FPGA-Accelerated Handwritten Digit Recognition Using an INT8 LeNet CNN on the DE1-SoC

Yizheng Tang (yt2992)
Weiwei Wu (ww2766)
Sirui Chen (sc5746)
Chenxi Shen(cs4634)
Tian Li (tl3468)

May 2026

Contents

1 Introduction

1.1 Project Background

Convolutional neural networks (CNNs) have become one of the most widely used neural network architectures for image classification tasks. Compared with traditional image processing algorithms, CNNs can automatically extract spatial features from input images and perform classification based on learned parameters. However, CNN inference requires a large number of multiply-accumulate operations, especially in convolutional and fully connected layers. When inference is executed only on a general-purpose processor, the computation can become inefficient in terms of latency and energy usage.

This project focuses on accelerating CNN inference on a DE1-SoC embedded platform. The target application is handwritten digit recognition, where a user draws or uploads a digit image and the system predicts one of ten digit classes, from 0 to 9. In the final version, the work was not just the CNN datapath itself; it also included the browser interface, preprocessing, Ethernet transfer, HPS Linux server, `/dev/mem` register access, and the FPGA accelerator that had to agree with the quantized software reference.

1.2 Motivation for FPGA-Based CNN Acceleration

FPGAs are well suited for CNN acceleration because they allow customized parallel datapaths, on-chip memory reuse, and fixed-point arithmetic. Instead of executing CNN operations sequentially on a general-purpose CPU, an FPGA can implement multiply-accumulate units, local buffers, and control logic directly in hardware. This makes it possible to reduce computation overhead and improve system efficiency.

For this project, fixed-point integer arithmetic is used instead of floating-point operations. Model parameters and intermediate activations are quantized to integer format, which avoids the cost of floating-point units and keeps the design small enough for the Cyclone V device. This choice also made debugging easier because the same integer operations could be reproduced in the Python reference model.

1.3 Project Objective

The objective of this project is to implement a complete FPGA-based CNN accelerator system for handwritten digit recognition. A working classifier alone would not have been enough for the demo; the system also needed a reliable path from browser input to HPS software, from HPS software to FPGA registers, and then back to the browser with prediction, logits, and timing information.

The complete system follows a software-hardware co-design approach. On the software side, a browser-based interface or uploaded image is processed by a Flask backend. The image is converted into a 32×32 grayscale pixel array and then sent to the DE1-SoC board through Ethernet using a TCP-based communication protocol. On the HPS side, a C program runs under Linux and receives the input data from the TCP socket. The HPS then accesses the FPGA accelerator through memory-mapped I/O using `/dev/mem` and the Lightweight HPS-to-FPGA AXI bridge.

1.4 Target Application: Handwritten Digit Recognition

The target application is to classify a 32×32 grayscale input image into one of ten digit classes, from 0 to 9. The CNN model is inspired by the LeNet-5 architecture, which is a classic convolutional neural network structure for digit recognition.

On the hardware side, the FPGA fabric contains a custom CNN inference core and a memory-mapped register interface. The register interface provides control, status, result, logit, and pixel input registers for communication between the HPS and FPGA. The input pixels are written into a memory-mapped register window, and the accelerator is started through a control register. After computation is complete, the HPS reads back the predicted digit and output logits from the FPGA registers and sends the result back to the software client.

The CNN accelerator performs the main inference operations, including convolution, pooling, flattening, fully connected computation, quantization, and output comparison.

1.5 Main Contributions

The major contributions of this project are as follows:

- A LeNet-style CNN model was trained and quantized for handwritten digit recognition.
- A software preprocessing pipeline was developed to convert browser input or uploaded images into FPGA-compatible 32×32 pixel data.
- A TCP-based communication path was implemented between the Flask software backend and the HPS running on the DE1-SoC board.
- A C-based HPS inference server was developed to receive image data, access FPGA registers, start the accelerator, and return results.
- A memory-mapped register interface was designed for HPS-FPGA communication through the Lightweight AXI bridge.
- A custom INT8 CNN accelerator was implemented in FPGA fabric.
- The complete system was verified through software reference comparison, register readback tests, RTL simulation, and end-to-end hardware testing.

1.6 Report Organization

The remainder of this report is organized as follows. Section 2 introduces the overall system block diagram and end-to-end dataflow. Section 3 describes the CNN algorithm and model structure. Section 4 explains the quantization method and parameter preparation flow. Section 5 presents the software input processing pipeline. Section 6 discusses the Ethernet TCP communication protocol. Section 7 describes the HPS software and MMIO interface. Section 8 provides the detailed register map design. Sections 9 and 10 describe the FPGA top-level architecture and CNN accelerator hardware design. Section 11 discusses memory organization. Section 12 presents the verification methodology. Section 13 evaluates the system results and performance. Finally, Sections 14 to 17 discuss limitations, future work, division of work, lessons learned, and conclusion.

2 System Block Diagram

2.1 Overall System Architecture

The complete system is an end-to-end software-hardware co-design for handwritten digit recognition. It connects a browser-based user interface, a Flask software backend, Ethernet TCP communication, an HPS Linux inference server, and a custom FPGA CNN accelerator.

The high-level system flow is shown in Figure 1. The path begins with browser-side image input, moves through software preprocessing and Ethernet transfer, and then enters the HPS-to-FPGA memory-mapped control path.

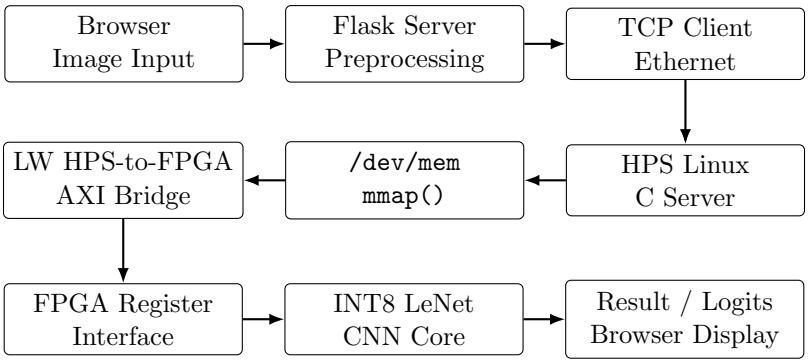


Figure 1: End-to-end software and hardware inference flow.

The browser provides the user input. The Flask server converts the input image into a 32×32 grayscale pixel array. The processed pixels are sent to the HPS over Ethernet using TCP. The HPS C server writes the pixels into FPGA registers through memory-mapped I/O and starts the CNN accelerator. After the FPGA finishes computation, the HPS reads back the predicted digit and logits and returns them to the web application.

2.2 Software-Hardware Co-Design Flow

The project uses a software-hardware co-design structure. Software handles the parts that changed often during debugging, such as image input, preprocessing, network communication, and result visualization. Hardware is reserved for the repeated CNN computation where fixed control flow and local memory reuse are useful.

The software side includes the browser drawing pad and image upload interface, Flask backend server, image preprocessing pipeline, Python TCP client, and software reference inference paths for comparison. The hardware and HPS side includes the HPS Linux C inference server, Lightweight HPS-to-FPGA AXI bridge, memory-mapped register interface, FPGA CNN accelerator core, and on-chip memories for pixels, feature maps, and parameters.

This separation also made debugging more manageable. When the browser preview looked wrong, the issue was usually in preprocessing; when the quantized software and FPGA outputs disagreed, the issue was more likely in parameter packing, register transfer, or RTL arithmetic.

Figure 2 provides an implementation-oriented view of this boundary. It highlights how the browser, Flask server, TCP client, HPS server, `/dev/mem` mapping, register interface, and RTL control logic are connected in the working system.

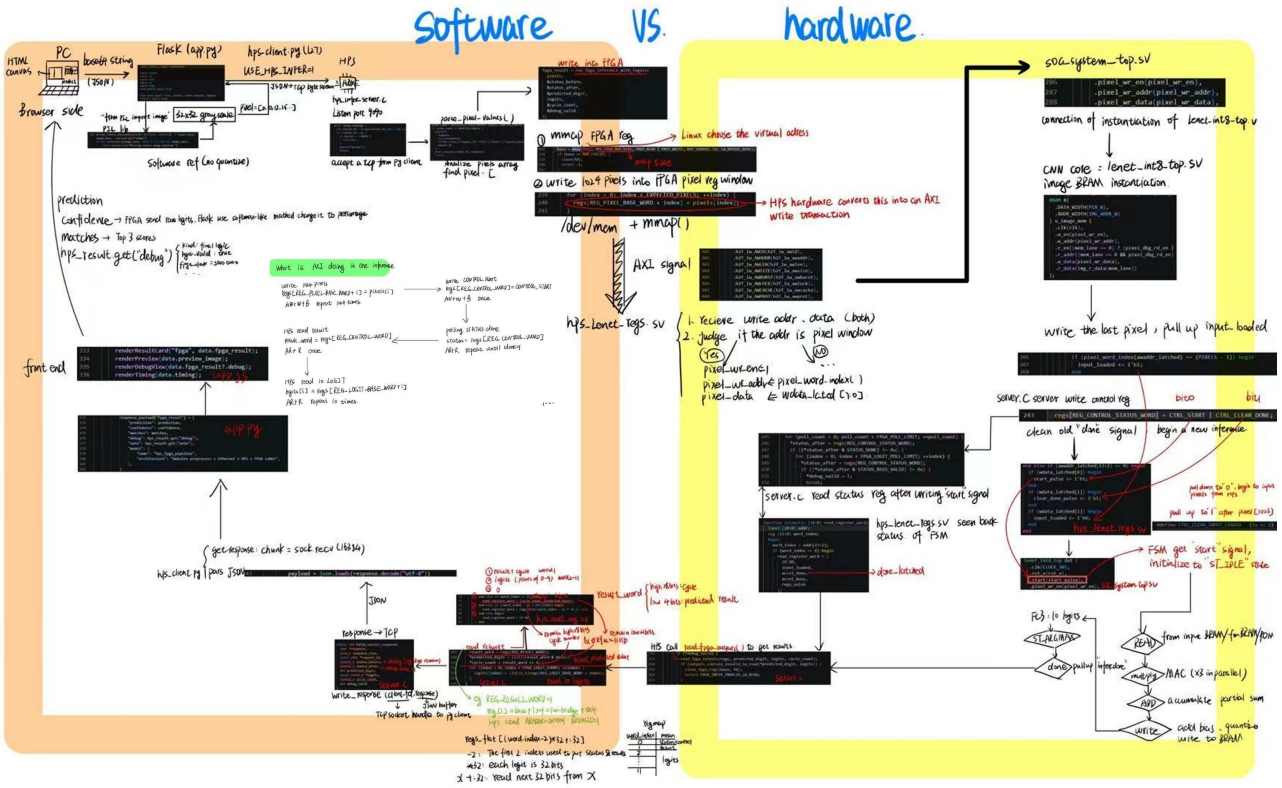


Figure 2: Annotated software-to-hardware integration flow used during system debugging.

2.3 Browser / Image Input Interface

The browser interface allows the user to either draw a digit on a canvas or upload an image. The drawing pad provides a black background and white stroke, which matches the expected digit format used by the preprocessing pipeline.

The user interface also includes controls such as stroke width, image inversion, prediction, upload, clear, and label saving. The browser does not directly communicate with the FPGA. Instead, it sends the canvas image to the Flask backend through an HTTP request.

The browser input stage is important because the quality of the handwritten digit directly affects recognition accuracy. A poorly centered or unclear drawing may lead to incorrect classification even if the FPGA hardware is functioning correctly.

Figure 3 shows the final web interface. The input canvas, preprocessing preview, FPGA prediction, software reference outputs, final logits, and speed comparison are displayed together. This layout was useful during testing because a single run could show whether the drawn digit, processed 32×32 input, quantized reference, and FPGA result were all consistent.

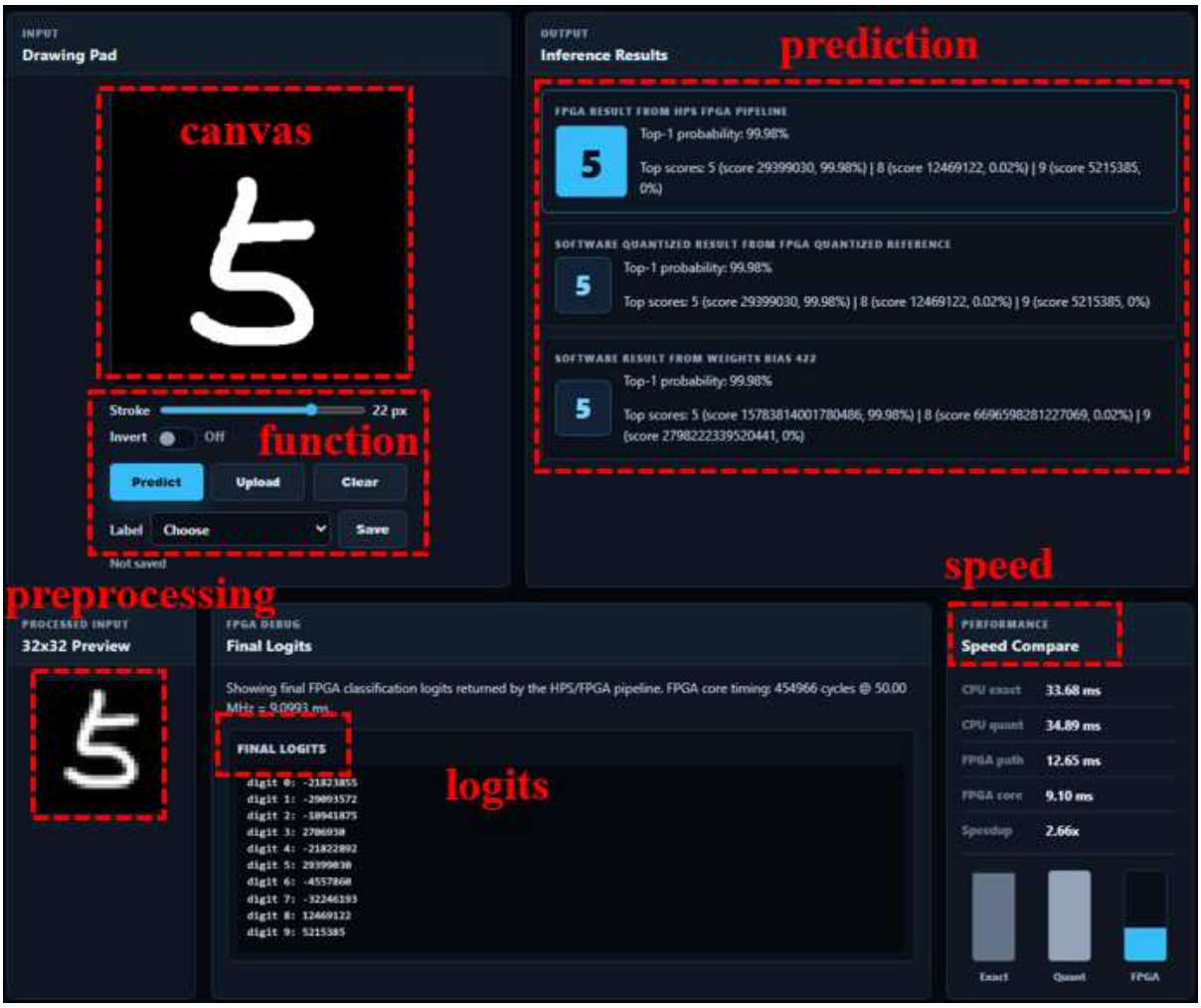


Figure 3: Annotated browser interface for drawing input, checking preprocessing, comparing predictions, and inspecting latency.

2.4 Flask Server and Preprocessing Module

The Flask server receives the image from the browser and performs preprocessing before sending data to the HPS. The preprocessing module converts the image into a format compatible with the FPGA accelerator.

The preprocessing pipeline is summarized in Table 1.

Table 1: Image Preprocessing Pipeline

Step	Operation	Output Purpose
1	RGBA/RGB to grayscale conversion	Single-channel image
2	Foreground detection and bounding-box crop	Remove unused background
3	Resize and center the digit	Match the LeNet input geometry
4	Optional smoothing and contrast adjustment	Improve robustness for drawings
5	Flatten to 32×32 unsigned 8-bit vector	FPGA input pixel window

The final output of preprocessing is a list of 1024 pixel values. Each pixel is an unsigned 8-bit grayscale value. This pixel vector is used both by the software reference model and by the FPGA

inference path. The final web interface, preprocessing code, software predictor, TCP client, and frontend files are included in Appendix A.1.

2.5 Ethernet TCP Communication Path

After preprocessing, the Flask backend sends the 1024-pixel input vector to the HPS board through Ethernet using TCP. TCP was selected because it provides reliable, ordered delivery. This is useful for this project because the complete image must arrive correctly and in order before inference starts.

The TCP client sends a request containing the image data. The HPS server receives the request, validates the input length, performs FPGA register access, and sends back a response containing the predicted digit and logits. Although TCP adds some communication overhead, it simplifies the software protocol and reduces the risk of corrupted or missing pixel data.

2.6 HPS Linux Inference Server

The HPS Linux inference server is implemented in C and runs on the ARM processor system of the DE1-SoC board. It listens for TCP connections from the Flask backend.

For each inference request, the server performs the following operations:

1. Receive 1024 input pixels through the TCP socket.
2. Open `/dev/mem`.
3. Map the Lightweight HPS-to-FPGA bridge address using `mmap()`.
4. Clear old FPGA status flags.
5. Write all input pixels into the FPGA pixel register window.
6. Write the control register to start inference.
7. Poll the status register until the FPGA asserts done.
8. Read the predicted digit and output logits.
9. Send the result back to the Flask client.

This C server acts as the bridge between network-level software communication and low-level FPGA register access.

2.7 Lightweight HPS-to-FPGA AXI Bridge

The Lightweight HPS-to-FPGA AXI bridge provides the memory-mapped communication path between the HPS and the FPGA fabric. From the HPS software perspective, FPGA registers appear as physical memory addresses starting at the lightweight bridge base address:

```
#define LW_BRIDGE_BASE 0xFF200000u
```

The HPS maps this region into user space through `/dev/mem` and `mmap()`. Once mapped, the C program accesses FPGA registers using normal pointer reads and writes. For example:

```
regs[REG_PIXEL_BASE_WORD + index] = pixels[index];
```

This C statement becomes a memory-mapped write. The HPS hardware converts it into an AXI write transaction across the lightweight bridge. On the FPGA side, the register interface receives the AXI signals, decodes the address, and stores the data.

2.8 FPGA CNN Accelerator Core

The FPGA fabric contains two major hardware blocks:

1. HPS-FPGA register interface
2. CNN accelerator core

The register interface is implemented in `hps_lenet_regs.sv`. It receives AXI read/write transactions from the HPS and exposes a simple control/status/data interface to the CNN core.

The CNN accelerator core is implemented in `lenet_int8_top.sv`. It performs the LeNet-style inference process using integer arithmetic. The main operations are:

```
Conv1 -> Pool1 -> Conv2 -> Pool2 -> FC1 -> FC2 -> FC3 -> Argmax
```

The accelerator uses on-chip BRAM/M10K memories for input pixels, intermediate feature maps, and packed parameter ROMs. Multiply-accumulate operations are reused across layers to reduce FPGA resource usage. Quantization is performed using shift-based rescaling and saturation logic.

2.9 End-to-End Inference Dataflow

The complete inference dataflow begins when the user draws or uploads a digit and ends when the predicted result is displayed in the browser.

1. The user draws a digit or uploads an image in the browser.
2. The browser sends the image to the Flask server.
3. Flask preprocesses the image into a 32×32 grayscale pixel vector.
4. The Python client sends the 1024 pixels to the HPS C server over TCP.
5. The HPS C server receives and validates the pixel data.
6. The HPS maps the lightweight bridge using `/dev/mem` and `mmap()`.
7. The HPS writes pixels into the FPGA `PIXEL_INPUT` register window.
8. The HPS writes the control register to start the accelerator.
9. The FPGA CNN core performs inference.
10. The FPGA asserts done and makes the result/logits available.
11. The HPS polls status, then reads `RESULT` and `LOGIT[0:9]`.

12. The HPS sends the prediction response back to Flask.
13. Flask returns the result to the browser.
14. The browser displays the FPGA result, software reference result, logits, and timing information.

This end-to-end structure demonstrates the complete integration of web software, network communication, embedded Linux, memory-mapped HPS-FPGA access, and FPGA-based neural network acceleration.

3 CNN Algorithm and Model Design

3.1 LeNet-5 Inspired CNN Structure

The hardware accelerator utilizes a convolutional neural network architecture based on LeNet-5, a pioneering structure developed for handwritten digit recognition. This implementation is tailored to classify MNIST-style digit images.

The design follows a sequential hierarchy consisting of two convolutional layers for feature extraction, two max-pooling stages for spatial downsampling, and three fully connected layers for classification. This structure provides a balance between accuracy and hardware cost. The model contains approximately 61,000 parameters, which is small enough to fit within the FPGA's on-chip memory while still providing strong classification accuracy for handwritten digits.

Figure 4 shows the layer dimensions used by the LeNet-style classifier. The input is expanded into convolutional feature maps, reduced by subsampling, and finally passed through fully connected layers to generate ten digit scores.

Layer Name	Input Dim	Operation	Output Dim	Weights (Stored as int8)
Input	32×32×1	Raw Pixel Data	32×32×1	0
Conv1	32×32×1	5×5 Kernel, 6 Filters	28×28×6	150 weights + 6 biases
Pool1	28×28×6	2×2 Max Pooling	14×14×6	0
Conv2	14×14×6	5×5 Kernel, 16 Filters	10×10×16	2,400 weights + 16 biases
Pool2	10×10×16	2×2 Max Pooling	5×5×16	0
Flatten	5×5×16	Unspool to 1D Vector	400×1	0
FC1	400×1	Matrix Mult (120 nodes)	120×1	48,000 weights + 120 biases
FC2	120×1	Matrix Mult (84 nodes)	84×1	10,080 weights + 84 biases
FC3 (Out)	84×1	Matrix Mult (10 nodes)	10×1	840 weights + 10 biases

Figure 4: LeNet-style layer dimensions for 32×32 handwritten digit classification.

3.2 Input Image Format: 32×32 Grayscale Pixels

The network consumes 32×32 grayscale images. Each input contains 1024 pixels. In the software preprocessing stage, pixels are represented as unsigned 8-bit values in the range 0 to 255. Inside the inference model, these values are interpreted in an integer-friendly format suitable for fixed-point computation.

Using a fixed 32×32 input size simplifies the hardware design because all layer dimensions and memory addresses can be statically determined. This also matches the LeNet-style structure, where the input image is progressively reduced by convolution and pooling operations.

3.3 Feature Extraction: Convolution and Pooling

Feature extraction is performed through two convolutional layers. The first convolutional layer uses six 5×5 filters to detect simple low-level features such as edges and stroke directions. The second convolutional layer uses sixteen 5×5 filters to capture more complex combinations of these features.

Between the convolutional layers, 2×2 max-pooling is applied to downsample the feature maps. Pooling reduces computational cost, lowers memory usage, and provides some tolerance to small input shifts. After the convolution and pooling stages, the spatial resolution is reduced to a final 5×5 feature map with 16 channels.

3.4 Classification: Flattening and Fully Connected Layers

After feature extraction, the final $5 \times 5 \times 16$ feature map is flattened into a 400-element vector. This vector is then passed through three fully connected layers. The first two fully connected layers learn higher-level digit representations, while the final layer maps these learned features to the ten possible digit classes.

To improve hardware efficiency, the model uses 8-bit signed weights and ReLU activation functions in the hidden layers. ReLU is especially suitable for FPGA implementation because it only requires a sign check and zero replacement for negative values.

3.5 Output Logits and Digit Prediction

The final fully connected layer produces ten output logits, with each logit corresponding to one digit class from 0 to 9. These logits are kept at 32-bit precision to preserve accuracy during the final decision stage. An argmax operation is then applied to select the index with the highest logit value. This index becomes the predicted digit and is transmitted back to the HPS as the final classification result.

3.6 Hardware-Aware Architecture Design

The CNN architecture was selected because it efficiently extracts spatial features using shared weights and local kernels. Compared with a standard multilayer perceptron, a CNN greatly reduces memory usage while still detecting important patterns such as edges, curves, and strokes.

Several design choices were made to simplify FPGA implementation. Traditional sigmoid activations and average pooling were replaced with ReLU and max pooling. ReLU only requires simple comparison logic, while max pooling can be implemented with a small set of comparators. Finally, the model was quantized from 32-bit floating-point arithmetic to 8-bit signed integer arithmetic. This reduces the size and resource cost of the MAC units while maintaining strong recognition accuracy.

4 Quantization and Parameter Preparation

4.1 Need for Integer Quantization

The primary motivation for moving away from standard 32-bit floating-point representation is the resource limitation of FPGA hardware. Storing a model in FP32 requires significant memory, and implementing floating-point multiply-accumulate units consumes many logic resources and DSP blocks.

To improve efficiency and maximize inference speed, this project uses INT8 quantization. This provides a four-times reduction in model parameter storage compared with FP32. By using 8-bit signed integers, the design can use simpler integer MAC datapaths and store all model parameters in on-chip memory. This keeps power consumption, area usage, and memory bandwidth requirements manageable.

4.2 Training and Quantization Strategy

The training and quantization strategy followed a two-stage pipeline: high-precision training followed by fixed-point integer conversion. The model was first trained in PyTorch using 32-bit floating-point precision to establish a baseline that was easy to evaluate. Only after the floating-point model behaved well were the weights converted into the integer files used by the software reference and FPGA ROMs.

After training, the learned kernels and fully connected weights were frozen and converted for FPGA inference using post-training quantization. The network stores all convolutional and fully connected parameters as signed 8-bit integers. Instead of using complex affine scaling, a shift-based quantization approach was used. For each layer, a scaling factor was selected so that the FP32 values could be mapped into the INT8 range while preserving classification accuracy.

4.3 Implementation of Quantization Logic

Quantization is required on the FPGA because arithmetic operations increase bit width. Multiplying two 8-bit signed values produces a 16-bit result, and accumulating many products can expand intermediate values to 32 bits. To pass data into the next layer, these 32-bit accumulations must be scaled back to a smaller activation range.

The FPGA implementation uses shift-based rescaling instead of division. The main rounding operation is:

$$y = \frac{x + 2^{n-1}}{2^n}$$

which is implemented as:

$$y = (x + (1 \ll (n-1))) \gg n$$

This approach performs round-to-nearest behavior before truncation. Dedicated clamping logic prevents wrap-around errors by saturating values that exceed the supported activation range.

The selected shift values are:

Table 2: Selected Layer Shift Values

Conv1 Shift	Conv2 Shift	FC1 Shift	FC2 Shift	Accuracy
6	8	8	7	98.20%

The output layer uses a shift value of zero so that the final logits remain in full 32-bit precision. This gives the argmax operation the highest possible resolution when selecting the predicted digit.

4.4 Parameter Files Used by Software and Hardware

The final project keeps trained parameters in `parameter/weights_bias_422`. The software reference path reads the unpacked INT8 hex files, while the FPGA accelerator reads lane-packed parameter files matched to `MAC_LANES = 3`. The relevant files are summarized in Table 3; the FPGA-ready packed files are included in Appendix B.

Table 3: Final Parameter Files Used by the Project

File Group	Files	Use
Software reference	`conv1.hex`, `conv2.hex`, `fc1.hex` `fc2.hex`, `fc3.hex`	Parsed by `predictor.py`
FPGA packed ROMs	`conv1_pack3.hex`, `conv2_pack3.hex` `fc1_pack3.hex`, `fc2_pack3.hex` `fc3_pack3.hex`	Loaded by `$readmemh`
Layer diagram	`layer dimension.png`	Report figure for model dimensions

4.5 Failure Analysis of Alternative Quantization Methodologies

Alternative methods such as affine post-training quantization and quantization-aware training were considered. Affine quantization can preserve numerical information through zero-point offsets and non-power-of-two scaling factors, but it increases hardware complexity because it requires additional offset correction and multiplication by scale factors.

Quantization-aware training was also explored, but it did not become the final path. In our experiments, the INT8 QAT version was harder to stabilize, and the extra training complexity was not worth the small benefit for this class project. An INT8-FP32 hybrid path was also tested, but the mixed precision made the software and hardware behavior harder to compare. For the final implementation, the simpler shift-based post-training quantization method was easier to verify layer by layer and matched the FPGA datapath more directly.

4.6 Static BRAM Configuration

To integrate the trained model into the hardware fabric, the finalized 8-bit signed weights and biases are exported from the Python environment into layer-specific hex memory initialization files. During Quartus compilation, these files are referenced by parameterized ROM modules through Verilog `INIT_FILE` parameters and loaded using `$readmemh`.

This static initialization makes the accelerator ready for inference immediately after FPGA configuration. By storing all parameters in on-chip BRAM or ROM-style memories, the CNN core can access weights and biases with deterministic low latency and avoid runtime parameter loading from the HPS.

5 Software Input Processing

5.1 Supported Input Sources

The handwritten digit recognition system supports two primary input modalities through its web-based interface. The first modality uses an interactive HTML5 canvas that allows the user to draw digits directly in the browser. The second modality allows the user to upload pre-existing image files in standard formats such as JPEG and PNG.

Both input paths are handled by the Flask backend. Regardless of input source, the image is converted into a common preprocessing pipeline. This ensures that both browser-drawn digits and uploaded images are normalized into the same 32×32 grayscale format expected by the CNN accelerator.

5.2 Browser Canvas Input

The browser-based input mechanism uses an HTML5 canvas element as the drawing surface. The canvas provides a high-resolution input area for users to sketch handwritten digits. The frontend JavaScript code supports configurable brush width, clearing, prediction, upload, and optional color inversion.

When the user draws on the canvas, JavaScript event handlers capture mouse or touch coordinates, convert them into canvas-relative coordinates, and render continuous line segments. The stroke rendering uses round line caps and joins to create smooth handwritten strokes. The final canvas image is serialized as a PNG-format data URL and sent to the Flask backend through an HTTP POST request.

5.3 Uploaded JPG / PNG Image Input

The image upload path supports common image formats, especially JPEG and PNG. When a user selects a file, the browser reads the file and converts it to a data URL representation. This data URL is transmitted to the backend in base64-encoded form.

Uploaded images can have arbitrary dimensions and aspect ratios. Therefore, the preprocessing pipeline must crop, resize, center, and normalize them before inference. This makes it possible to use the same CNN accelerator for both drawn and uploaded digit images.

5.4 Flask Backend Image Reception

The Flask backend implements a prediction endpoint that receives JSON-formatted requests from the frontend. Each request contains an image field with a base64-encoded image string. The backend decodes the data URL, converts the raw bytes into a PIL image object, and passes the image to the preprocessing pipeline.

The backend also performs error handling. If the image data is missing, malformed, or cannot be decoded, the server returns a JSON error response to the browser. This prevents invalid image payloads from being forwarded to the HPS or FPGA.

5.5 Grayscale Conversion

The first preprocessing step is grayscale conversion. The PIL `convert("L")` method converts the input image into an 8-bit grayscale representation. If the input image contains an alpha channel, it is first composited onto a solid background to avoid transparency artifacts.

This step ensures that the subsequent CNN input contains one intensity value per pixel rather than RGB color channels. Since handwritten digit recognition primarily depends on shape rather than color, grayscale input is sufficient.

5.6 Foreground Cropping and Background Removal

After grayscale conversion, the preprocessing pipeline detects the foreground digit by applying an intensity threshold. Pixels above the threshold are treated as foreground. The system then computes a bounding box around all foreground pixels and crops the image to this region.

A small margin is added around the detected bounding box to preserve antialiased stroke edges and avoid excessive cropping. If no foreground pixels are detected, the backend returns an error indicating that no digit was found. This prevents blank inputs from being sent to the FPGA.

5.7 Resizing to 32×32 Pixels

The cropped digit may have arbitrary dimensions. Directly resizing it to 32×32 could distort the aspect ratio, so the preprocessing pipeline uses aspect-ratio-preserving resizing. The digit is first scaled to fit inside a 24×24 interior region, then placed on a 32×32 black canvas.

This creates a border around the digit and makes the image more consistent with the training distribution. Centering and padding are important because the CNN model expects digit strokes to appear near the center of the image.

5.8 Conversion to 8-Bit Pixel Data

After spatial normalization, the image is converted into 8-bit pixel values. Optional smoothing and contrast adjustment are applied to improve robustness. The final image contains 1024 unsigned 8-bit values in row-major order.

These pixel values are directly used by the software reference model and also serialized for FPGA inference. Each pixel is represented as an integer between 0 and 255.

5.9 Generating FPGA-Compatible Input Format

The final 32×32 grayscale image is converted into a Python list of 1024 integers using row-major order. For FPGA inference, the list is serialized as JSON and sent to the HPS C server through the TCP client.

The request includes the command type, request ID, image width, image height, and pixel list. This format is easy to debug and makes it possible to inspect the full input payload during development.

5.10 Preprocessing Limitations and Error Sources

The preprocessing pipeline has several limitations. A fixed foreground threshold may fail for unusual contrast conditions. Very thin strokes may disappear during resizing, while very thick strokes may merge and change the apparent digit shape. Center-of-mass adjustment can also shift noisy or poorly cropped inputs incorrectly.

These limitations affect recognition accuracy even when the FPGA hardware is correct. If the processed 32×32 preview is distorted, the accelerator will classify that distorted input. Therefore, input quality and preprocessing robustness remain important parts of the overall system.

6 Software-Hardware Communication Protocol

6.1 Ethernet Architecture and Protocol Selection

The communication backbone of the system uses Ethernet to exchange data between the Flask application and the HPS backend. Ethernet was selected over lower-bandwidth serial protocols such as UART because each inference requires transmitting a 1024-pixel image payload. Ethernet provides sufficient bandwidth for interactive inference.

The system uses TCP/IP rather than UDP. TCP provides reliable, ordered delivery, which ensures that every pixel arrives correctly and in the proper sequence. This is important because the FPGA cannot begin inference until the full image has been received and validated.

6.2 TCP Implementation and JSON Payload Handling

The system uses a TCP client-server model. The HPS runs a C-based server on port 9090. The Flask backend acts as the client and sends a JSON request for each inference.

A typical request has the following form:

```
{
  "command": "infer",
  "request_id": "...",
  "width": 32,
  "height": 32,
  "pixels": [0, 0, ..., 255]
}
```

The request is sent as a newline-terminated UTF-8 JSON string:

```
sock.sendall((json.dumps(request) + "\n").encode("utf-8"))
```

The newline termination makes the protocol simple because the receiver can treat one line as one complete request.

6.3 Inference Result Reporting and Transmission

Once the FPGA asserts the done and valid status bits, the HPS reads the inference outputs from hardware registers. These outputs include the predicted digit, the ten output logits, and the execution cycle count. The server packages these values into a structured JSON response and sends the response back to the Flask client over the same TCP connection.

A successful response contains the prediction, source information, debug status values, FPGA cycle count, compute time, and logits. This allows the browser interface to display both the classification result and useful diagnostic information.

6.4 Latency Measurement and Fair Performance Comparison

The Flask client can measure the full round-trip latency of each inference request, but this value includes TCP communication, JSON serialization, HPS control, register access, FPGA execution, and response handling. Therefore, it represents end-to-end system latency rather than pure CNN execution time.

For fair comparison, the FPGA core compute time is calculated using the cycle count returned by the accelerator:

$$T_{\text{FPGA}} = \frac{\text{cycle count}}{50\,\text{MHz}}$$

This cycle-based timing isolates the CNN computation from communication and software overhead.

6.5 HPS-Side JSON Parsing and Input Validation

On the HPS side, the C server receives the TCP payload using `recv()` and stores it in a request buffer. The server checks that the request contains the `infer` command and extracts the request ID.

The pixel array is parsed into a `uint8_t pixels[1024]` buffer. The server validates that the request contains exactly 1024 pixel values and that each value is between 0 and 255. If validation fails, the server returns an error response instead of writing invalid data into FPGA registers.

6.6 One-Connection-Per-Inference Design

The current implementation uses one TCP connection for one inference request. The Python client opens a socket, sends one JSON request, waits for one response, and then closes the connection.

This design was simple and reliable for an interactive lab demo. Each inference transaction is independent, so a failed request does not corrupt future requests or leave the HPS server in a half-finished state. The tradeoff is that connection setup and teardown add extra latency. A persistent TCP connection could reduce overhead in a higher-throughput implementation.

6.7 Error Handling and Response Format

The HPS server returns structured JSON responses. Successful responses contain `"ok": true`, while failures contain `"ok": false` and an error string. Implemented error cases include unsupported commands, invalid pixel payloads, `/dev/mem` access failure, invalid lightweight bridge readback, FPGA timeout, and response buffer overflow.

This structured format makes debugging easier because the frontend can identify whether the failure happened during network communication, input parsing, MMIO access, or FPGA execution.

6.8 Separation Between Network Protocol and Hardware Protocol

The project uses two different protocols at different system layers. At the software network layer, Flask and HPS communicate using TCP and JSON. At the HPS-to-FPGA hardware layer, the HPS communicates with FPGA registers using MMIO through `/dev/mem` and the Lightweight AXI bridge.

This separation keeps the FPGA logic simple. The FPGA does not parse JSON and does not implement a network stack. The HPS C server acts as a translator between the network protocol and the hardware register protocol.

6.9 Protocol Limitations

Although TCP and JSON are easy to debug, they are not the most efficient possible format. JSON is human-readable but larger than binary data because pixel values are transmitted as text. The current design is acceptable because the image is small, but for larger images or real-time video, a compact binary protocol would be more efficient.

The main limitations are JSON parsing overhead, one connection per inference, variable TCP latency, sequential HPS request handling, and the need to receive and write the entire pixel array before starting the FPGA. These limitations are acceptable for a classroom demo, but they would matter more for batched inputs or real-time use.

6.10 Code Mapping

The software-hardware communication path is distributed across several files.

Table 4: Communication Path Code Mapping

Function	File
Flask receives browser request	`web_demo/app.py`
Python TCP client builds JSON request	`web_demo/hps_client.py`
HPS C server listens on port 9090	`hps_infer_server.c`
HPS parses pixel payload	`hps_infer_server.c`
HPS writes pixels to FPGA registers	`hps_infer_server.c`
FPGA register constants	`hps_fpga_regs.h`
FPGA AXI register interface	`hps_lenet_regs.sv`
CNN accelerator core	`lenet_int8_top.sv`

The final HPS inference server and register header are included in Appendix A.2.

7 HPS Software and MMIO Interface

7.1 HPS Linux Runtime Environment

The HPS side of the DE1-SoC runs an embedded Linux environment on the ARM processor system. This environment provides standard POSIX APIs, including file I/O, sockets, memory mapping, and signal handling. The inference server is implemented as a C program running in user space.

The HPS Linux environment is responsible for receiving input data from the network, accessing FPGA registers through memory-mapped I/O, and returning inference results to the client. The server also installs signal handlers so that it can shut down cleanly when interrupted.

7.2 C-Based Inference Server

The inference server is a single-threaded TCP server written in C. It follows the standard socket programming sequence: `socket()`, `bind()`, `listen()`, and `accept()`. The server listens on port 9090 and handles one client request at a time.

For each connection, the server reads the JSON payload, parses the input pixels, runs FPGA inference, builds a JSON response, sends it back to the client, and closes the socket. This single-threaded design is simple and sufficient for an interactive demonstration.

7.3 Receiving Pixel Data from TCP Socket

The server receives the request using `recv()` and stores it in a fixed-size buffer. It then checks for the expected `infer` command and parses the `pixels` array. Pixel values are converted from decimal strings into integers and stored in a 1024-element buffer.

The server verifies that exactly 1024 values are present. It also checks that every value is in the valid 8-bit range. These checks prevent incomplete or corrupted inputs from being written into the FPGA register window.

7.4 Memory-Mapped I/O Overview

Memory-mapped I/O allows software to access hardware registers as if they were normal memory locations. The FPGA accelerator exposes its control, status, result, logit, and pixel input registers

through the HPS-to-FPGA lightweight bridge. The HPS maps the physical bridge address into user space and then accesses the registers through a volatile pointer.

This avoids the need for a custom kernel driver, which simplifies development. However, using `/dev/mem` requires appropriate permissions and must be handled carefully because it gives access to physical memory.

7.5 Relationship Between `/dev/mem`, `mmap()`, and MMIO

The `/dev/mem` device file represents the physical memory address space. The HPS server opens this file and uses `mmap()` to map the lightweight bridge physical address into the process virtual address space.

The mapping uses the following key parameters:

- Physical base address: `0xFF200000`
- Mapping size: `0x1030` bytes
- Protection: read and write
- Mapping type: shared

The user-space mapping size is chosen to cover every accelerator register from the first control/status word through the final pixel input word. The last pixel register is `PIXEL_INPUT[1023]`, so the final used byte offset is:

$$\text{last\_offset} = 0x0030 + 1023 \times 4 = 0x102C$$

Because `PIXEL_INPUT[1023]` is a 32-bit register, the mapped user-space region must include four bytes starting at `0x102C`. Therefore, the required mapping length is:

$$\text{user\_space\_size} = \text{last\_offset} + 4 = 0x102C + 4 = 0x1030 \text{ bytes}$$

After `mmap()` succeeds, the physical FPGA register range:

$$0xFF200000 \leq \text{physical address} < 0xFF201030$$

is visible through a user-space virtual base pointer. A byte offset Δ from the lightweight bridge base is accessed as:

$$\text{user\_addr}(\Delta) = \text{user\_base} + \Delta$$

Equivalently, after casting the mapping to `volatile uint32_t *`, software accesses a register by word index:

$$\texttt{regs[word\_index]}, \qquad \texttt{word\_index} = \frac{\Delta}{4}$$

After mapping, the returned address is cast to `volatile uint32_t *`. The `volatile` keyword is essential because it prevents the compiler from optimizing away reads and writes to hardware registers.

7.6 Lightweight HPS-to-FPGA AXI Bridge

The lightweight bridge connects the HPS processor to custom FPGA logic. Each pointer read or write in the C program becomes an AXI transaction across this bridge. On the FPGA side, the register interface decodes the address and performs the corresponding register operation.

The bridge is used for pixel writes, control writes, status polling, result reads, and logit reads. This provides a compact and deterministic interface between software and hardware.

7.7 Physical Address to Virtual Address Mapping

The `open_fpga_regs()` function encapsulates the mapping process. It opens `/dev/mem`, maps the FPGA register region, and returns a pointer that can be used for register access.

For example, `regs[0]` corresponds to the control/status register at physical address `0xFF200000`. Similarly, `regs[12]` corresponds to the first pixel input register at byte offset `0x0030`.

7.8 Register Read Function

The HPS reads the result register to obtain the predicted digit and cycle count. The lower four bits contain the predicted digit, and the upper bits contain the FPGA cycle count.

```
uint32_t result_word = regs[REG_RESULT_WORD];
*predicted_digit = (int)(result_word & 0xfu);
*cycle_count = result_word >> 4;
```

The server also reads ten signed 32-bit logits from consecutive logit registers. These logits are returned to the browser for debugging and confidence analysis.

7.9 Register Write Function

Register writes are used to upload pixels and issue control commands. The HPS writes each pixel to the pixel input window:

```
for (index = 0; index < EXPECTED_PIXELS; ++index) {
    regs[REG_PIXEL_BASE_WORD + index] = pixels[index];
}
```

Only the lower 8 bits of each 32-bit word are used by the FPGA. Control commands are issued by writing to the control/status register.

7.10 Pixel Upload Sequence

The pixel upload sequence begins by clearing stale status flags. Then the HPS writes 1024 pixel values into the pixel window. The register interface tracks the number of pixels written and asserts the input-loaded status bit when the full image has been received.

This transfer method is simple and reliable. Its main limitation is that it requires 1024 separate MMIO writes for each image.

7.11 Accelerator Start Sequence

After pixel upload, the server starts the accelerator by writing:

```
regs[REG_CONTROL_STATUS_WORD] = CTRL_START | CTRL_CLEAR_DONE;
```

The FPGA register interface converts this write into a one-cycle `start_pulse`. The CNN core then begins executing the fixed inference sequence.

7.12 Done Polling Sequence

After issuing the start command, the server polls the status register until the done bit becomes high. The polling loop includes a timeout to avoid waiting forever if the hardware enters an invalid state.

Once the done bit is set, the server waits for the result-valid bit before reading the result and logits. This ensures that output registers are stable before readback.

7.13 Reading Prediction and Logits

After inference completes, the server reads the packed result word and the ten logit registers. The predicted digit is the argmax result computed by hardware. The logits are signed 32-bit scores corresponding to the ten digit classes.

Returning logits is useful because it allows the browser interface to display more than just the final predicted digit. If the top two logits are close, the model is less confident.

7.14 Returning Results to the Software Client

The server builds a JSON response containing the predicted digit, logits, FPGA cycle count, compute time, status values, and request ID. The response is sent back through the TCP connection.

This completes one inference transaction from browser input to FPGA computation and back to browser display.

8 Register Map Design

8.1 Register Map Overview

The HPS communicates with the FPGA accelerator through a memory-mapped register interface exposed on the Lightweight HPS-to-FPGA bridge. The register map is implemented in `hps_lenet_regs.sv` and accessed by the HPS C inference server through `/dev/mem` and `mmap()`.

The base physical address of the lightweight bridge is `0xFF200000`. All accelerator registers are placed as offsets from this base address.

Table 5: HPS-FPGA Register Map Overview

Start–End Offset	Name	Access	Comment
0x0000–0x0004	CONTROL_STATUS	RW	Write control commands; read accelerator status
0x0004–0x0008	RESULT	RO	Predicted digit and FPGA cycle count
0x0008–0x0030	LOGIT_0–LOGIT_9	RO	Ten signed 32-bit final logit values
0x0030–0x1030	PIXEL_INPUT[0:1023]	RW	Runtime input image window

8.2 Register Address Space Allocation

The total mapped register space is `0x1030` bytes. The address space is allocated as shown in Table 6.

Table 6: Register Address Space Allocation

Offset Range	Register Region
0x0000	CONTROL_STATUS
0x0004	RESULT
0x0008--0x002C	LOGIT_0 to LOGIT_9
0x0030--0x102C	PIXEL_INPUT[0] to PIXEL_INPUT[1023]

The final used register is PIXEL_INPUT[1023] at offset:

$$0x0030 + 1023 \times 4 = 0x102C$$

Since each register is 4 bytes, the mapped region must extend to:

$$0x102C + 4 = 0x1030$$

8.3 Address Alignment Requirement

All registers are 32-bit aligned. This means every register offset is a multiple of 4 bytes. In hardware, the byte address is converted to a word index by dropping the lower two address bits:

$$\text{word\_index} = \text{addr}[13 : 2]$$

Using aligned 32-bit registers simplifies both HPS software pointer arithmetic and FPGA address decoding logic.

8.4 Reason for Using 32-Bit Registers

The register interface uses 32-bit registers because the HPS lightweight bridge and CPU MMIO accesses naturally operate on 32-bit words. Although each input pixel only needs 8 bits, using one 32-bit register per pixel keeps the interface simple and reliable.

For pixel writes, only the lower 8 bits are used. For logits and result values, the full 32-bit register is used.

8.5 Reason for Separating Control, Status, Result, Logit, and Pixel Regions

The register map separates different functions into different regions to make the hardware/software protocol clear. The control/status register is accessed frequently, so it is placed at the beginning. The result register follows immediately after it. The ten logit registers are grouped together, and the large pixel input window is placed after the control and output region.

This separation improves readability, prevents accidental overlap, and makes address calculation simple.

8.6 Base Address Selection

The HPS software accesses the FPGA register interface through the lightweight bridge base address:

```
#define LW_BRIDGE_BASE 0xFF200000u
```

The C server maps this physical address using `/dev/mem` and `mmap()`, then accesses FPGA registers by adding word offsets.

8.7 Offset Address Definition

The software defines register addresses as word offsets:

```
#define REG_CONTROL_STATUS_WORD 0u
#define REG_RESULT_WORD         1u
#define REG_LOGIT_BASE_WORD     2u
#define REG_PIXEL_BASE_WORD     12u
```

Since each word is 4 bytes, the pixel input window starts at byte offset:

$$12 \times 4 = 0x0030$$

8.8 Control Register

The control register is written by the HPS at offset `0x0000`. It is used to send command pulses to the FPGA register interface.

Table 7: Control Register Bit Fields

Bit	Name	Function
bit 0	CTRL_START	Start one CNN inference
bit 1	CTRL_CLEAR_DONE	Clear the done flag inside the CNN core
bit 2	CTRL_CLEAR_INPUT_LOADED	Clear the input loaded flag
bit 31:3	Reserved	Unused

8.9 Status Register

The same offset `0x0000` is read as a status register. This means the register is write-control and read-status.

Table 8: Status Register Bit Fields

Bit	Name	Function
bit 0	STATUS_REGS_VALID	Output registers and logits are valid
bit 1	STATUS_BUSY	Accelerator is currently running
bit 2	STATUS_DONE	Accelerator has finished inference
bit 3	STATUS_INPUT_LOADED	All 1024 input pixels have been written
bit 31:4	Reserved	Read as zero

8.10 Result Register

The result register is located at offset `0x0004` and is read-only from the HPS side.

Table 9: Result Register Bit Fields

Bits	Field	Function
bit 3:0	predicted_digit	Final predicted digit, 0 to 9
bit 31:4	cycle_count	FPGA core cycle count

The HPS C server unpacks it using:

```
*predicted_digit = (int)(result_word & 0xfu);
*cycle_count = result_word >> 4;
```

Figure 5 gives the same protocol in bit-field form. It highlights that the write-side control bits and read-side status bits share the same base offset, while the result register packs the predicted digit with the cycle counter. This view is useful when checking that the software constants and RTL bit assignments agree exactly.

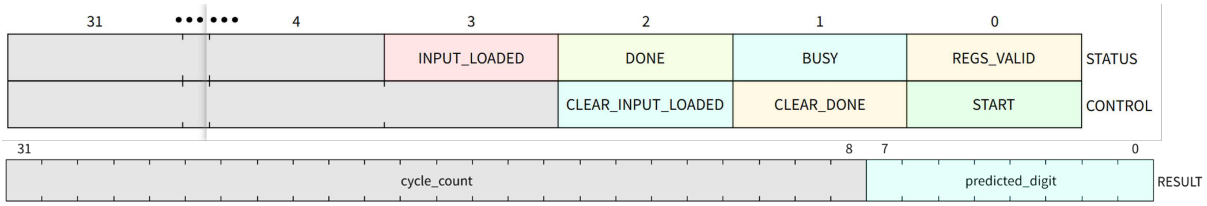


Figure 5: Detailed control, status, and result register bit-field layout.

8.11 LOGIT[0:9] Registers

The ten logit registers start at offset `0x0008`. Each register stores one signed 32-bit final score.

Table 10: Logit Register Address Table

Register	Offset	Function
LOGIT_0	0x0008	Signed score for digit 0
LOGIT_1	0x000C	Signed score for digit 1
LOGIT_2	0x0010	Signed score for digit 2
LOGIT_3	0x0014	Signed score for digit 3
LOGIT_4	0x0018	Signed score for digit 4
LOGIT_5	0x001C	Signed score for digit 5
LOGIT_6	0x0020	Signed score for digit 6
LOGIT_7	0x0024	Signed score for digit 7
LOGIT_8	0x0028	Signed score for digit 8
LOGIT_9	0x002C	Signed score for digit 9

8.12 PIXEL_INPUT[0:1023] Register Window

The pixel input window starts at offset `0x0030`. It contains 1024 registers, one for each pixel of the 32×32 input image. Each pixel register is 32 bits wide, but only bits $[7:0]$ are used.

The HPS writes pixels using:

```
regs[REG_PIXEL_BASE_WORD + index] = pixels[index];
```

The FPGA register interface writes the lower byte into the accelerator input memory.

8.13 Pixel Register Address Calculation

The pixel register byte offset is:

$$\text{PIXEL\_INPUT}[i] = 0x0030 + 4i$$

where $i = 0, 1, \ldots, 1023$. In hardware, the byte address is converted into a word index, and the pixel index is calculated by subtracting the pixel base word.

8.14 Register Reset Values

On reset, the register interface clears command pulses, read/write state, and input status. The result and logit values are generated by the CNN accelerator core, so software should rely on `STATUS_DONE` and `STATUS_REGS_VALID` before treating result registers as meaningful.

8.15 Register Read / Write Permissions

Table 11: Register Read and Write Permissions

Region	Access	Description
CONTROL_STATUS	RW	Write commands, read status bits
RESULT	RO	Read predicted digit and cycle count
LOGIT_0–LOGIT_9	RO	Read final output logits
PIXEL_INPUT[0:1023]	RW	Write input pixels, optional debug readback

8.16 Start / Done Handshake Register Protocol

The HPS software follows a polling-based start/done protocol:

1. Read current status from `CONTROL_STATUS`.
2. Write `CTRL_CLEAR_DONE | CTRL_CLEAR_INPUT_LOADED`.
3. Write 1024 input pixels into `PIXEL_INPUT[0:1023]`.
4. Wait for the FPGA to set `STATUS_INPUT_LOADED`.
5. Write `CTRL_START | CTRL_CLEAR_DONE`.
6. Poll `CONTROL_STATUS` until `STATUS_DONE` becomes 1.
7. Continue checking until `STATUS_REGS_VALID` becomes 1.
8. Read `RESULT`.
9. Read `LOGIT_0` to `LOGIT_9`.

8.17 Complete Register Address Table

Table 12: Complete Register Address Table

Offset	Name	Function
0x0000	CONTROL_STATUS	Write control commands; read status
0x0004	RESULT	Predicted digit and cycle count
0x0008	LOGIT_0	Logit score for digit 0
0x000C	LOGIT_1	Logit score for digit 1
0x0010	LOGIT_2	Logit score for digit 2
0x0014	LOGIT_3	Logit score for digit 3
0x0018	LOGIT_4	Logit score for digit 4
0x001C	LOGIT_5	Logit score for digit 5
0x0020	LOGIT_6	Logit score for digit 6
0x0024	LOGIT_7	Logit score for digit 7
0x0028	LOGIT_8	Logit score for digit 8
0x002C	LOGIT_9	Logit score for digit 9
0x0030–0x102C	PIXEL_INPUT[0:1023]	Runtime input image window

8.18 Register Map Verification

The register map was verified from both the software and hardware sides. On the software side, the HPS C server uses constants from `hps_fpga_regs.h`. On the hardware side, `hps_lenet_regs.sv` decodes AXI addresses into word indices using `addr[13:2]`.

End-to-end testing confirmed that 1024 pixels can be written into the pixel window, the accelerator can be started through the control register, the HPS can poll the done bit, and all result/logit registers can be read back correctly. The final FPGA register-interface RTL is included with the accelerator source code in Appendix A.3.

9 FPGA Hardware Architecture

9.1 Top-Level FPGA Design

The FPGA hardware design integrates the HPS subsystem, a memory-mapped register interface, and a custom CNN accelerator core. The top-level structure connects the HPS lightweight AXI bridge to FPGA logic so that software running on the HPS can control the accelerator through memory-mapped registers.

The main FPGA-side components are summarized in Table 13.

Table 13: Top-Level FPGA Components

Component	Role
`soc_system_top`	Board-level integration wrapper
Platform Designer HPS subsystem	ARM, DDR, Ethernet, and lightweight bridge
`hps_lenet_regs`	Memory-mapped register interface
`lenet_int8_top`	CNN accelerator datapath and control FSM
BRAM / packed ROM memories	Runtime pixels, feature maps, weights, and biases

Figure 6 shows the detailed hardware organization. The diagram emphasizes the separation between the HPS subsystem, the memory-mapped register interface, and the `lenet_int8_top` inference core.

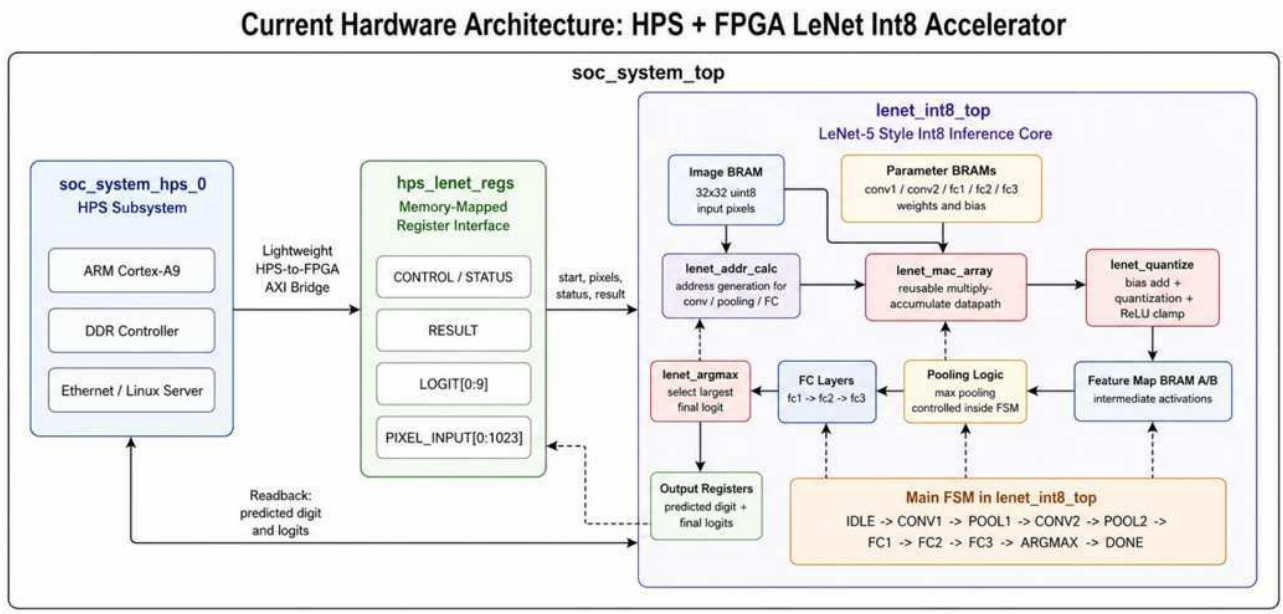


Figure 6: Current HPS and FPGA LeNet INT8 accelerator architecture.

9.2 `soc_system_top` Module

`soc_system_top.sv` is the board-level top module. It connects the physical DE1-SoC board pins, HPS subsystem signals, lightweight AXI bridge signals, and custom FPGA accelerator logic.

Its main responsibilities are to instantiate the Platform Designer HPS system, connect HPS peripherals, route lightweight AXI signals into the FPGA register interface, instantiate `hps_lenet_regs.sv`, instantiate `lenet_int8_top.sv`, and connect control/status/pixel/result/logit signals between the register interface and CNN core.

9.3 HPS Subsystem Integration

The HPS subsystem is generated by Platform Designer and instantiated inside the top-level design. It contains the ARM processor system and HPS peripherals required to run Linux and communicate with the outside world.

The HPS subsystem provides the ARM processor, DDR interface, Ethernet interface, SD card interface, UART/debug interface, and Lightweight HPS-to-FPGA bridge. In this project, the most important connection is the lightweight bridge, which allows the HPS C server to access FPGA registers.

9.4 `hps_lenet_regs.sv` Register Interface

`hps_lenet_regs.sv` implements the FPGA-side memory-mapped register interface. It receives AXI read and write transactions from the lightweight bridge and converts them into simple accelerator control signals.

The register interface decodes AXI byte addresses into 32-bit word indices, handles AXI read/write responses, generates `start_pulse`, writes input pixels into the CNN input buffer, and returns status, result, logits, and pixel readback data to the HPS.

9.5 `lenet_int8_top.sv` CNN Accelerator Core

`lenet_int8_top.sv` is the main CNN inference core. It receives input pixels from the register interface and executes the LeNet-style inference sequence:

$$\text{Conv1} \rightarrow \text{Pool1} \rightarrow \text{Conv2} \rightarrow \text{Pool2} \rightarrow \text{FC1} \rightarrow \text{FC2} \rightarrow \text{FC3}$$

The core uses integer arithmetic. Weights are stored as 8-bit values, activations are stored in fixed-width integer format, and MAC operations use wider accumulators to reduce overflow risk.

9.6 Clock and Reset Design

The accelerator uses the board 50 MHz clock as the main FPGA logic clock. This clock drives the register interface, CNN core, BRAMs, and control FSMs. The reset signal initializes the accelerator state, clears output registers, resets counters, and places the FSM into the idle state.

The design meets the 50 MHz target clock. The reported Fmax for `clock_50_1` is 61.94 MHz, giving positive timing margin.

9.7 Input Pixel Buffer

The input image is stored in an on-chip BRAM-based pixel buffer. The input size is $32 \times 32 = 1024$ pixels. Each pixel is an unsigned 8-bit grayscale value.

The HPS writes pixels through the register interface, which converts AXI writes into `pixel_wr_en`, `pixel_wr_addr`, and `pixel_wr_data`. These signals are connected to the CNN core.

9.8 Weight and Bias Memory

Weights and biases are stored in packed ROMs initialized from hex files. The design uses five parameter ROM groups: Conv1, Conv2, FC1, FC2, and FC3. Each packed ROM word contains multiple 8-bit parameters aligned with the MAC lane structure.

9.9 Intermediate Feature Map Storage

Intermediate activations are stored in on-chip feature map memories. The design uses two feature map buffer groups, allowing one buffer to act as source while another acts as destination. This ping-pong strategy reduces storage duplication and avoids off-chip intermediate memory.

9.10 Output Logit Storage

The final fully connected layer produces ten output logits. These logits are stored as signed 32-bit values and flattened into a packed bus called `logits_flat`. The register interface maps this bus into ten read-only registers.

9.11 Top-Level Signal Connections

Table 14: Top-Level Signal Connections Between Register Interface and CNN Core

Signal	Direction	Purpose
start_pulse	hps_lenet_regs → lenet_int8_top	Start one inference
clear_done_pulse	hps_lenet_regs → lenet_int8_top	Clear previous done state
pixel_wr_en	hps_lenet_regs → lenet_int8_top	Input pixel write enable
pixel_wr_addr	hps_lenet_regs → lenet_int8_top	Input pixel write address
pixel_wr_data	hps_lenet_regs → lenet_int8_top	Input pixel write data
busy	lenet_int8_top → hps_lenet_regs	Accelerator is running
done	lenet_int8_top → hps_lenet_regs	Accelerator finished
predicted_digit	lenet_int8_top → hps_lenet_regs	Final predicted digit
logits_flat	lenet_int8_top → hps_lenet_regs	Ten final logits
cycle_count	lenet_int8_top → hps_lenet_regs	Number of inference cycles
regs_valid	lenet_int8_top → hps_lenet_regs	Result/logit registers are valid

9.12 FPGA Hardware Block Diagram

The simplified accelerator view in Figure 7 shows the main execution order inside the FPGA. The HPS writes pixels and control commands through the lightweight bridge, while the FPGA core advances through convolution, pooling, and fully connected stages before exposing the final answer and logits.

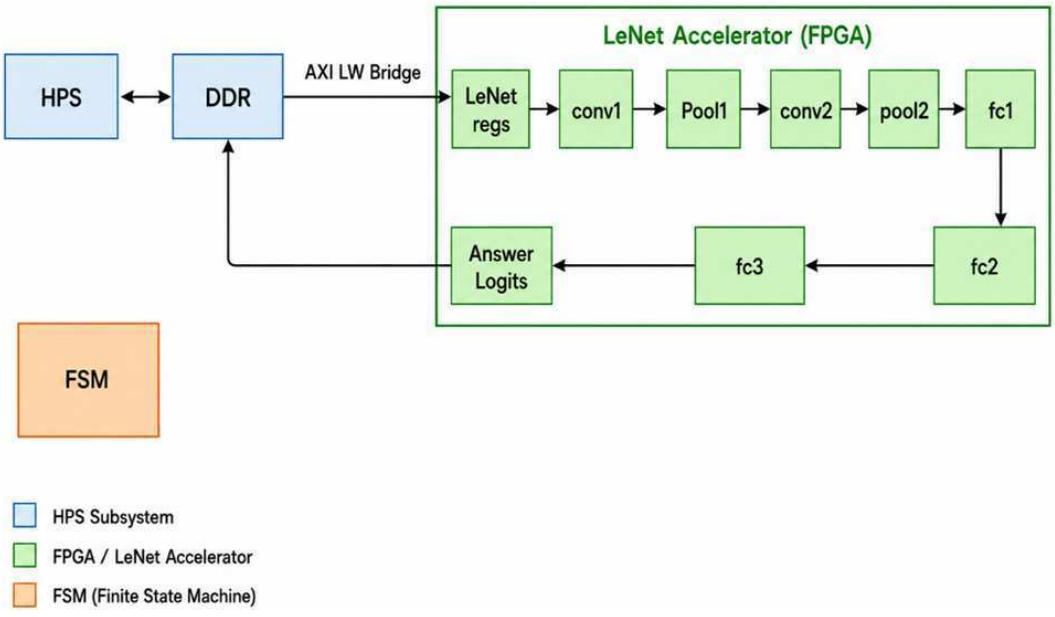


Figure 7: Simplified HPS-to-FPGA accelerator block diagram.

The final top-level FPGA wrapper and CNN RTL source files are listed in Appendix A.3.

10 CNN Accelerator Hardware Design

10.1 Execution Flow and Global FSM

The accelerator executes a fixed LeNet-style inference sequence:

$$\text{Conv1} \rightarrow \text{Pool1} \rightarrow \text{Conv2} \rightarrow \text{Pool2} \rightarrow \text{FC1} \rightarrow \text{FC2} \rightarrow \text{FC3} \rightarrow \text{Argmax}$$

This flow is implemented in `lenet_int8_top.sv` using a single global finite-state machine. The FSM contains explicit per-stage states for initialization, memory reads, accumulation, bias addition, quantization, writeback, argmax, and completion.

```
assign busy = (state != ST_IDLE) && (state != ST_DONE);
assign done = (state == ST_DONE);
```

A start pulse launches one full inference pass, and the FSM returns to idle after completion.

Table 15 summarizes the major state-group transitions in the global CNN FSM. Each stage advances when the corresponding loop counters reach the final output element and the final MAC group for that output has completed.

Table 15: Overall CNN FSM State-Group Transitions

Current State Group	Transition Condition	Next State Group
`ST_IDLE`	`start` == 1	Conv1 states
Conv1 states	`out_ch` == 5, `out_y` == 27, `out_x` == 27, `term_idx` + 3 >= 25	Pool1 states
Pool1 states	`pool_ch` == 5, `pool_y` == 13, `pool_x` == 13, `pool_read_idx` == 3	Conv2 states
Conv2 states	`out_ch` == 15, `out_y` == 9, `out_x` == 9, `term_idx` + 3 >= 150	Pool2 states
Pool2 states	`pool_ch` == 15, `pool_y` == 4, `pool_x` == 4, `pool_read_idx` == 3	FC1 states
FC1 states	`neuron_idx` == 119, `term_idx` + 3 >= 400	FC2 states
FC2 states	`neuron_idx` == 83, `term_idx` + 3 >= 120	FC3 states
FC3 states	`neuron_idx` == 9, `term_idx` + 3 >= 84	`ST_ARGMAX`
`ST_ARGMAX`	fixed 1 cycle	`ST_DONE`
`ST_DONE`	fixed 1 cycle	`ST_IDLE`

10.2 Convolution Datapath

Both convolution layers reuse the same datapath template. During read states, activation data and packed weight data are fetched from BRAM and ROM. During accumulation, lane inputs are fed into the MAC array. During writeback, bias and quantization are applied before storing the output activation.

Conv1 reads from image memory and writes to one feature map bank. Conv2 reads from a feature map bank and writes to another. Address generation is delegated to `lenet_addr_calc.sv`, avoiding hard-coded address logic inside the FSM.

Figure 8 records the earlier convolution and pooling datapath sketch. It is included as design context because it shows the same core ideas used in the final RTL: input BRAM, pixel buffering, kernel memory, MAC accumulation, bias/ReLU, pooling, and feature-map memory writeback.

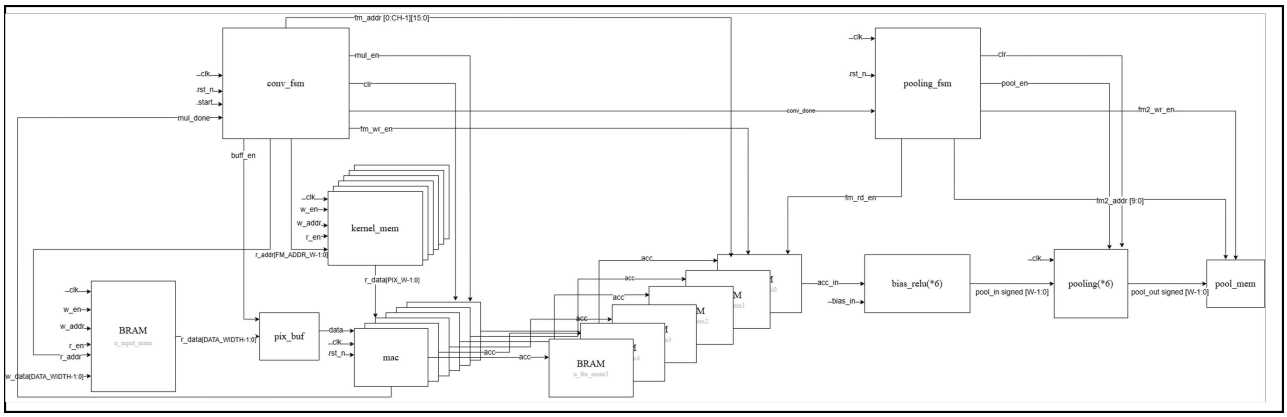


Figure 8: Datapath sketch for convolution, activation, pooling, and feature-map memory organization.

10.3 PE and MAC Design

The processing element is implemented as a lane-parallel MAC unit in `lenet_mac_array.sv`. The design uses `MAC_LANES = 3`. Each lane multiplies one activation value and one signed weight value per cycle. The lane products are then reduced into a shared 32-bit partial sum.

```
for (lane = 0; lane < MAC_LANES; lane = lane + 1) begin
  if (valid_mask[lane]) begin
    partial_sum = partial_sum + (act_value * weight_value);
  end
end
```

A valid mask disables out-of-range tail lanes when the vector length is not divisible by the lane count.

10.4 Bias Addition and Quantization

Bias values are stored in the same packed parameter memory space as weights. For each output channel or neuron, the controller fetches the bias after the final MAC segment and adds it to the accumulated value.

The quantizer applies right-shift scaling with rounding and saturation. Negative values are clamped to zero for ReLU-style activation. Layer-specific shift constants are defined in `lenet_int8_pkg.sv`.

```
assign acc_with_bias = acc_reg +
    {{(ACC_W-BIAS_W){current_bias[BIAS_W-1]}}, current_bias};

if (value_in <= 0)
    value_out = '0;
```

10.5 Max Pooling Datapath

Pooling is implemented as 2×2 max pooling. For each pooled output, four source activations are read sequentially, compared against a running maximum, and written to the next feature map bank.

This implementation is simple and deterministic. It reduces control complexity and matches the staged FSM structure.

10.6 Flatten / Fully Connected Datapath

There is no separate flatten module in hardware. Flattening is realized implicitly through linear indexing when FC layers read pooled feature maps. FC1, FC2, and FC3 reuse the same MAC and accumulator pattern as convolution, with different loop bounds and parameter address formulas.

FC1 and FC2 outputs are quantized activations written back to BRAM, while FC3 keeps full accumulated logits for final classification.

10.7 Output and Completion Signaling

Final prediction is produced by an argmax module that scans all ten logits and outputs the index of the maximum score.

```
if (score > best_score) begin
  best_score = score;
  best_idx = idx;
end
digit = best_idx[3:0];
```

The logits are exposed through register-mapped outputs for software-side inspection. Completion is indicated by entering `ST_DONE`; system-level logic latches completion so HPS polling remains robust.

10.8 Hardware Latency Analysis

Latency is measured in hardware using a cycle counter driven by the 50 MHz system clock. The HPS reads this counter and converts it to compute time. This number is the cleanest measurement of the CNN core itself. End-to-end response latency is reported separately because it also includes pixel writes, polling, TCP communication, JSON parsing, and browser-side handling.

Table 16 shows the cycle-level breakdown of the CNN core. The cycle estimate follows the staged datapath structure: each output element requires setup cycles, repeated three-lane MAC groups, and a final write or transition cycle. The convolution and fully connected stages dominate runtime because they perform the largest number of multiply-accumulate operations.

Table 16: CNN Core Stage Cycle Breakdown

Stage	Output Count	Cycles Needed per Output	Total Cycles
Start	1	1	1
Conv1	$6 \times 28 \times 28 = 4704$	$1 + \lceil 25/3 \rceil \times 3 + 1 = 29$	136,416
Pool1	$6 \times 14 \times 14 = 1176$	$1 + 4 \times 2 + 1 = 10$	11,760
Conv2	$16 \times 10 \times 10 = 1600$	$1 + \lceil 150/3 \rceil \times 3 + 1 = 152$	243,200
Pool2	$16 \times 5 \times 5 = 400$	10	4,000
FC1	120	$1 + \lceil 400/3 \rceil \times 3 + 1 = 404$	48,480
FC2	84	$1 + \lceil 120/3 \rceil \times 3 + 1 = 122$	10,248
FC3	10	$1 + \lceil 84/3 \rceil \times 3 + 1 = 86$	860

The table sums to 454,965 cycles. The measured counter value is 454,966 cycles because the implementation also includes the final completion/accounting cycle before `done` is observed by the surrounding control logic.

11 Memory Architecture

11.1 Memory Requirement Analysis

The design targets a 32×32 grayscale input and multiple intermediate feature maps for convolution, pooling, and fully connected processing. Layer dimensions and parameter counts are fixed in `lenet_int8_pkg.sv`. The design uses on-chip memory for input pixels, feature maps, and parameters.

11.2 Input Pixel Storage

Input pixels are written by the HPS through the lightweight AXI memory-mapped pixel window. The pixel input window contains 1024 words for one complete input frame.

On the FPGA side, these writes drive the image BRAM interface through `pixel_wr_en`, `pixel_wr_addr`, and `pixel_wr_data`. This runtime loading path removes dependence on static input files and enables online inference requests.

11.3 Kernel / Bias Parameter Memory

Layer parameters are stored in dedicated packed ROM blocks. The ROMs use one initialization file per layer group. Data are lane-packed to match the MAC lane count, so one memory read returns multiple weights aligned to MAC consumption. Biases are appended after the weight region in the same address space.

```
(* ramstyle = "M10K" *) reg [LANES*DATA_WIDTH-1:0] mem [0:WORD_COUNT-1];

initial begin
  if (INIT_FILE != "") begin
    $readmemh(INIT_FILE, mem);
  end
end
```

11.4 Feature Map Memory Organization

Intermediate activations are stored in two on-chip feature map banks using a ping-pong strategy. One bank acts as producer destination while the other acts as consumer source, then roles swap at stage boundaries. This minimizes conflicts and avoids off-chip intermediate storage.

11.5 BRAM / M10K Usage

Both writable memories and parameter ROMs request M10K mapping through synthesis attributes. This provides predictable timing and local bandwidth for the staged read-accumulate-write schedule.

```
(* ramstyle = "M10K" *) reg [DATA_WIDTH-1:0] mem [0:(1<<ADDR_WIDTH)-1];

always @(posedge clk) begin
  if (w_en)
    mem[w_addr] <= w_data;
  if (r_en)
    r_data <= mem[r_addr];
end
```

11.6 Parameter Initialization Using `$readmemh` and `.memh` File Format

Parameters are initialized using `$readmemh`, enabling deterministic startup without runtime weight transfer from the HPS. The files are hex text files arranged to match the hardware packing format. The software reference pipeline follows the same signed-INT8 interpretation, supporting cross-validation between software and hardware behavior.

11.7 Addressing and Access Scheduling

Address generation for convolution and pooling is centralized in `lenet_addr_calc.sv`. Fully connected stages use linear indexing with per-neuron row offsets. Memory access is explicitly staged to match synchronous memory behavior. This improves correctness, readability, and cycle-level traceability.

11.8 Reuse Strategy and Bandwidth Limitations

The architecture emphasizes reuse. One MAC datapath serves all layers, two feature-map banks serve intermediate tensors, and preloaded ROMs serve parameter reads. This is area-efficient and robust for an educational FPGA platform.

The tradeoff is throughput. Bandwidth is constrained by sequential FSM scheduling, synchronous memory reads, and modest lane parallelism. Future speedups would require more lanes, deeper pipelining, or more memory banking.

12 Verification and Testing

12.1 Verification Overview

Verification was performed at multiple levels: software model verification, quantized model verification, RTL simulation, HPS register interface testing, FPGA on-board testing, and final browser-to-FPGA end-to-end testing.

The main purpose was to avoid debugging the whole system as one black box. The flow was checked in layers: first the software reference and preprocessing, then the quantized model, then RTL simulation, then register readback on the board, and finally the full browser-to-FPGA path.

12.2 Python Golden Model

A Python reference model was implemented in the web demo backend. It uses the same trained parameter files and performs the same LeNet-style structure: Conv1, Pool1, Conv2, Pool2, FC1, FC2, and FC3.

This model is used as the first-level reference because it is easier to inspect and debug than hardware. It allows preprocessing, parameter loading, layer dimensions, and final classification to be checked before running the FPGA.

12.3 Layer-by-Layer Output Comparison

Layer-by-layer verification was used during development to check whether each CNN stage produced reasonable intermediate values before moving on to the next stage. The main stages checked were input image, Conv1 output, Pool1 output, Conv2 output, Pool2 output, FC1 output, FC2 output, and final logits. This was especially helpful when distinguishing a quantization mismatch from an address-generation or parameter-packing bug.

For RTL simulation, internal signals such as FSM state, counters, accumulator, output channel, row/column index, and final logits can be observed in ModelSim waveforms. This helped confirm that the dataflow through the CNN follows the expected order.

12.4 Quantized Model Verification

Since the FPGA accelerator uses integer arithmetic, a quantized Python reference path was implemented. This path applies integer convolution and fully connected computation, followed by layer-specific shift-based rescaling and ReLU-style clamping.

The shift values match the hardware configuration:

CONV1_SHIFT	6
CONV2_SHIFT	8
FC1_SHIFT	8
FC2_SHIFT	7

The FPGA result should be compared against this quantized reference rather than only against the non-quantized software result.

12.5 Register Interface Test

The HPS-to-FPGA register interface was tested to confirm that software register offsets match FPGA address decoding logic. The test focused on verifying that control writes generate correct command pulses, status reads return valid busy/done/input-loaded bits, pixel writes go into the correct pixel index, and result/logit reads return correct data.

12.6 HPS Register Readback Test

HPS register readback was used to verify that the HPS can correctly read FPGA-side registers through the lightweight bridge. The C server reads the result register and extracts the predicted digit and cycle count. It also reads ten final logits from consecutive registers.

12.7 Pixel Register Write Test

The pixel write path was tested by sending a full 32×32 image from the HPS to the FPGA. The HPS writes 1024 pixel values, one 32-bit MMIO word per pixel. The FPGA uses only the lower 8 bits of each write.

When the last pixel is written, the FPGA register interface sets the input-loaded status bit. This provides a simple way to confirm that the full image was transferred.

12.8 Start / Done Handshake Test

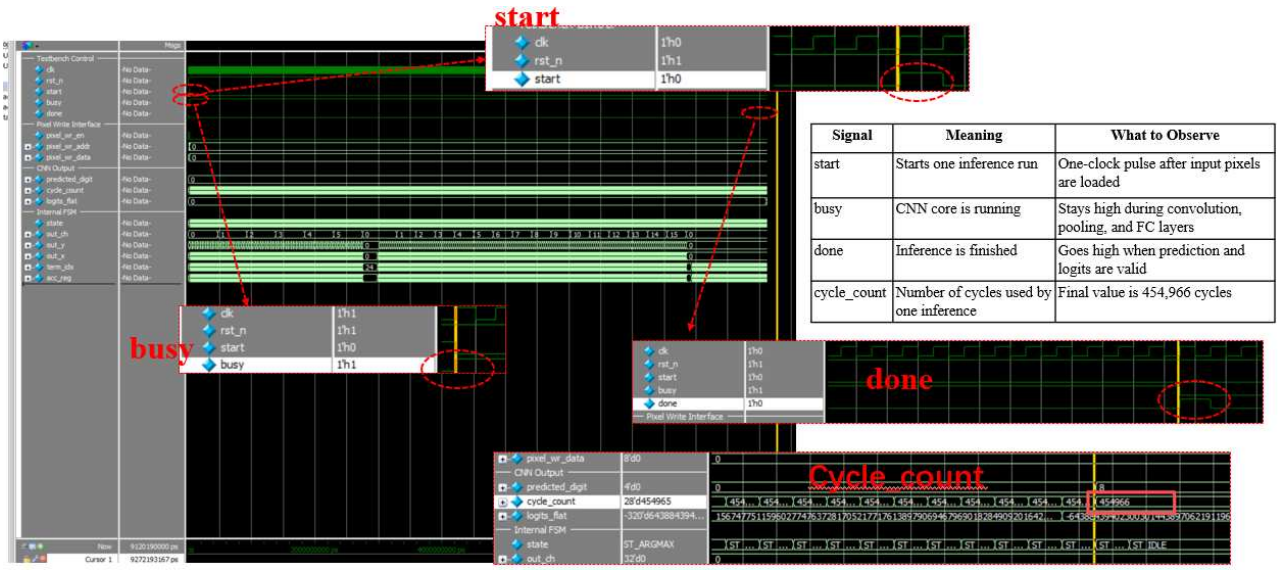
The start/done protocol was verified both in software and in RTL simulation. After writing all pixels, the C server writes the control register with `CTRL_START | CTRL_CLEAR_DONE`. The FPGA register interface converts this into a one-cycle start pulse. The HPS then polls the status register until the done bit becomes high.

12.9 ModelSim RTL Simulation

RTL simulation was performed using `tb_lenet_int8_top.sv`. This testbench directly instantiates the CNN core without the HPS or AXI bridge. The testbench generates a 50 MHz clock, applies reset, writes a simple 32×32 input image, pulses start, waits for done, and prints the predicted digit, cycle count, and logits.

The purpose of this simulation is not to test recognition accuracy, but to confirm that the core can complete a full inference pass without deadlock or timeout. For this project, seeing the expected start, busy, done, and cycle-count behavior in ModelSim was the fastest way to catch FSM and counter mistakes before testing on the board. The final ModelSim testbench and run script are included in Appendix A.4.

Figure 9 shows the key waveform observations. The highlighted regions confirm the one-cycle `start` pulse, the asserted `busy` interval, the final `done` signal, and the measured cycle count of 454,966 cycles.



Signal	Meaning	What to Observe
start	Starts one inference run	One-clock pulse after input pixels are loaded
busy	CNN core is running	Stays high during convolution, pooling, and FC layers
done	Inference is finished	Goes high when prediction and logits are valid
cycle_count	Number of cycles used by one inference	Final value is 454,966 cycles

Figure 9: ModelSim waveform used to verify start, busy, done, and cycle-count behavior.

12.10 FPGA On-Board Testing

After RTL simulation, the design was compiled in Quartus and programmed onto the DE1-SoC board. The HPS Linux system was booted, Ethernet was configured, and the C inference server was executed on the board.

On-board testing verified that the bitstream can be programmed successfully, the HPS can access the lightweight bridge, pixel data can be written through MMIO, the accelerator can be started from software, and the final prediction/logits can be read back.

Figure 10 shows the DE1-SoC board during an on-board test. The seven-segment display is used as a visible sanity check for the predicted digit while the HPS/server path reports the full logits and

timing values.

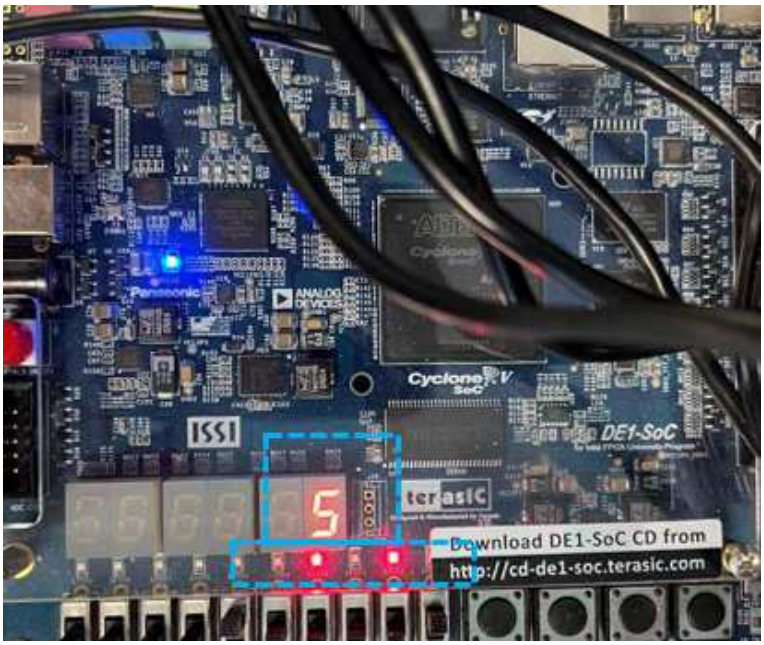


Figure 10: DE1-SoC board test showing a visible predicted digit on the seven-segment display.

12.11 End-to-End Browser-to-FPGA Test

The final system test was performed through the browser interface. The user draws or uploads a digit, Flask preprocesses the image, the TCP client sends pixels to the HPS C server, the HPS writes FPGA registers, the CNN core performs inference, and the result is displayed in the browser.

The web page displays the FPGA result, software quantized reference result, and software floating-point reference result. This allows the hardware result to be compared with software references during live testing.

12.12 Test Cases for Digits 0–9

Test cases were created by drawing or uploading digits from 0 to 9. The tests included clear digits, thin and thick strokes, centered and shifted digits, similar-looking digits such as 3/5 and 4/9, and uploaded image examples.

For each test, the predicted digit, top logits, and processed 32×32 preview were observed.

Figure 11 shows representative digit test samples. The grid is useful for checking whether the preprocessing output still resembles the intended class and for identifying ambiguous handwritten shapes.

Figure 11: Representative processed digit samples used for functional testing.

12.13 Misclassification Case Analysis

Misclassification cases were mainly caused by preprocessing sensitivity and similarity between handwritten digit shapes. Common causes included poor centering, very thin or thick strokes, ambiguous digit shapes, mismatch with the training distribution, and close logits between similar classes.

If the FPGA result differs from the quantized software reference, the issue is likely in hardware computation, register transfer, or parameter initialization. If both agree but the prediction is wrong, the issue is likely caused by model accuracy or input quality.

13 Results and Performance Evaluation

13.1 Functional Correctness Results

The completed system successfully performs end-to-end handwritten digit inference from the browser interface to the FPGA accelerator. The user can draw or upload a digit image, and the software backend preprocesses it into a 32×32 grayscale image. The processed pixel array is transmitted to the HPS, written into the FPGA input register window, and then processed by the CNN accelerator.

The FPGA returns the predicted digit, ten final logits, FPGA cycle count, and status/debug information. The web interface displays the FPGA result together with two software reference paths: the quantized software reference and the regular software model.

13.2 Recognition Accuracy

The trained LeNet-style model was evaluated using a digit test set containing digits 0 through 9. Based on the confusion matrix obtained during parameter testing, the model achieved approximately:

$$\frac{968}{1000} = 96.8\%$$

accuracy. Most digits were classified correctly, especially clear and centered examples. Accuracy was lower for ambiguous handwritten shapes and poorly preprocessed inputs.

13.3 Example Correct Predictions

Correct predictions were observed when the input digit was clearly drawn, centered, and had a reasonable stroke thickness. In these cases, the FPGA result, quantized software result, and regular software result usually agreed.

A correct prediction typically has the following behavior:

- The top logit corresponds to the correct digit.
- The top logit is much larger than the second-highest logit.
- The FPGA result matches the quantized software reference.
- The processed 32×32 preview still resembles the intended digit.

13.4 Example Incorrect Predictions

Incorrect predictions occurred mainly for unclear or ambiguous inputs. Common failure cases included 3 confused with 5, 4 confused with 9, 8 confused with 9, and 0 confused with 6.

These errors were often caused by handwritten shape ambiguity, poor centering, or stroke thickness. The final logits are useful for analyzing these cases. If the top two logits are close, the model is uncertain.

13.5 FPGA Core Runtime

The FPGA core runtime is measured using the cycle counter inside the accelerator. The cycle count is packed into the result register together with the predicted digit.

A representative measured FPGA cycle count was:

$$454,966 \text{ cycles}$$

At a 50 MHz FPGA clock, the core runtime is:

$$\frac{454,966}{50,000,000} = 9.10 \text{ ms}$$

This value represents pure FPGA CNN execution time after input pixels have already been written to the FPGA. It is lower than the browser-observed latency because it excludes the software and HPS-FPGA transfer work around the accelerator.

13.6 HPS Register Access Time

The HPS communicates with the FPGA through memory-mapped register accesses over the lightweight bridge. The largest register access overhead comes from writing 1024 pixel registers. Other overheads include writing the control register, polling the status register, reading the result register, and reading ten logit registers.

The register-window approach is simple and reliable but slower than a DMA-based transfer.

13.7 Ethernet Transmission Time

The browser/Flask side communicates with the HPS through Ethernet using TCP. Each inference request sends a JSON payload containing the command, request ID, image dimensions, and 1024 pixel values.

Because the image size is small, Ethernet communication is acceptable for an interactive demo. However, Ethernet transmission time can vary due to socket overhead, JSON serialization, operating system scheduling, and network buffering.

13.8 End-to-End Inference Latency

The end-to-end latency includes all software and hardware stages: Flask preprocessing, TCP transmission, HPS socket receiving, JSON parsing, MMIO pixel writes, FPGA computation, status polling, result/logit readback, TCP response, and browser display.

Table 17: Representative Inference Latency Comparison

Path	Runtime
CPU exact software path	33.68 ms
CPU quantized software path	34.89 ms
FPGA end-to-end path	12.65 ms
FPGA core only	9.10 ms

The FPGA end-to-end path is slower than pure FPGA core time because it includes communication and HPS control overhead. However, it is still faster than the software-only inference path in the measured example.

13.9 Runtime Variation Analysis

The FPGA core runtime is deterministic for a fixed design because the CNN accelerator follows a fixed FSM sequence. For the same bitstream, the number of stages and loop iterations does not change with the digit value, so the cycle count stays stable across inference runs.

The end-to-end latency is more variable because it includes software and communication overhead. Variation can come from Flask scheduling, Python JSON serialization, TCP connection setup, HPS Linux scheduling, socket receive timing, MMIO polling interval, and browser-side handling.

13.10 FPGA Resource Utilization

The final Quartus compilation was successful on the Cyclone V device 5CSEMA5F31C6. The fitted resource usage is shown in Table 18.

Table 18: FPGA Resource Utilization

Resource	Usage
Logic utilization	1,532 / 32,070 ALMs, 5%
Total registers	1,272
Total pins	362 / 457, 79%
Block memory bits	1,310,448 / 4,065,280, 32%
RAM blocks	165 / 397, 42%
DSP blocks	5 / 87, 6%
PLLs	0 / 6, 0%

The design uses relatively little logic and DSP resource, but a larger portion of memory resources. This is expected because the accelerator stores input images, intermediate feature maps, and packed parameter ROMs in on-chip memories.

Figure 12 visualizes the same resource profile. The chart makes the design tradeoff clear: the implementation is not logic- or DSP-limited, but it consumes a more noticeable fraction of the available M10K memory blocks.

Figure 12: Quartus FPGA resource utilization summary for the fitted accelerator design.

13.11 Timing Analysis and Fmax

The design meets timing at the target 50 MHz clock. The Quartus Timing Analyzer reports the following timing results.

Table 19: Timing Analysis Summary

Clock Domain / Metric	Result
clock_50_1 Fmax	61.94 MHz
clock_50_1 setup slack	+3.855 ns
clock_50_1 hold slack	+0.314 ns
Worst overall setup slack	+1.730 ns

The Quartus Fmax report in Figure 13 confirms that the `clock_50_1` domain has a restricted Fmax of 61.94 MHz, which is above the 50 MHz target used by the design.

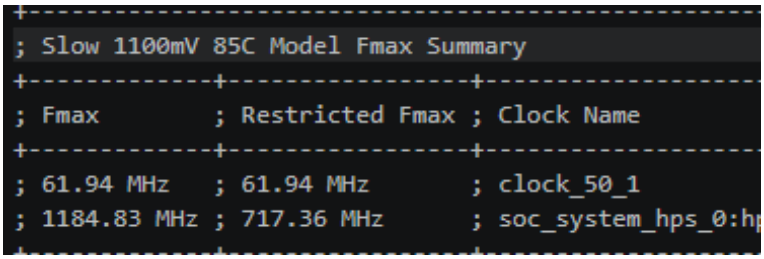


Figure 13: Quartus Fmax summary for the 50 MHz FPGA clock domain.

An earlier critical path came from address-generation logic in the convolution datapath. The original expression computed the kernel row, kernel column, input-channel index, and packed ROM tap index directly from `term_idx` using division and modulo operations. Although this code was compact, synthesis mapped the divide/modulo expressions into relatively expensive combinational logic.

```
ky = term_idx / 5;
kx = term_idx % 5;
conv2_ic = term_idx / 25;
conv2_tap = term_idx % 25;
addr = channel_offset + (y + ky) * width + (x + kx);
```

Figure 14: Original convolution address-generation logic using division and modulo operations.

The optimized version avoids the expensive modulo-by-5 path for the kernel column by replacing it with a small lookup-style `case` structure. Since the kernel index pattern is fixed for a 5×5 convolution, the hardware can decode the tap position with simple comparisons and constants instead of a general arithmetic remainder circuit.

```
case (idx)
    0, 5, 10, 15, 20: tap5_x = 3'd0;
    1, 6, 11, 16, 21: tap5_x = 3'd1;
    2, 7, 12, 17, 22: tap5_x = 3'd2;
    3, 8, 13, 18, 23: tap5_x = 3'd3;
    default:          tap5_x = 3'd4;
endcase
```

Figure 15: Optimized tap lookup logic used to reduce the old convolution address-generation critical path.

This change shortened the combinational path in the convolution address logic. A separate timing issue in the final output comparison was also improved by replacing a wide combinational argmax path with a sequential argmax scan.

13.12 Comparison with Software-Only Inference

Using the representative latency values:

$$\text{End-to-end speedup} = \frac{33.68}{12.65} = 2.66\times$$

$$\text{Compute speedup} = \frac{33.68}{9.10} = 3.70\times$$

The compute speedup is higher because it excludes Ethernet, HPS, and register access overhead.

13.13 Performance Bottleneck Analysis

The main computational bottleneck in the CNN core comes from multiply-accumulate operations in convolution and fully connected layers. To reduce resource usage, the design reuses a small number of MAC lanes instead of fully unrolling all operations.

At the system level, the main bottlenecks are 1024 separate MMIO pixel writes, TCP/JSON communication overhead, HPS polling, sequential MAC reuse, and on-chip memory bandwidth. Future improvements could include DMA-based input transfer, interrupt-based completion, persistent TCP connections, binary payload format, and increased CNN parallelism.

14 Failure Modes and Limitations

14.1 Image Preprocessing Errors

One major limitation is that the final prediction depends heavily on image preprocessing. If the crop, resize, centering, or contrast adjustment fails, the FPGA may receive a distorted 32×32 input image. This type of failure is not caused by the FPGA computation itself.

14.2 Stroke Thickness and Digit Shape Sensitivity

The recognition result is sensitive to stroke thickness and digit shape. Very thin strokes may disappear after resizing, while very thick strokes may merge parts of the digit. The same digit may produce different logits if the stroke width, drawing speed, or shape changes.

14.3 Similar Digit Misclassification

Some digits are naturally difficult to distinguish when handwritten. Common confusing pairs include 3 and 5, 4 and 9, 7 and 1, 8 and 9, and 0 and 6. If two digits have close logits, small input changes or quantization differences can change the predicted class.

14.4 Quantization Error

The FPGA implementation uses integer arithmetic instead of floating-point arithmetic. This reduces hardware cost but introduces quantization error. If the FPGA result matches the quantized software reference but differs from the non-quantized reference, the difference is likely caused by quantization rather than an RTL bug.

14.5 Overflow Risk

CNN layers perform many multiply-accumulate operations. Even if inputs and weights are 8-bit, accumulated sums can become much larger. The design reduces overflow risk by using 32-bit accumulation and applying shift-based rescaling after each major layer. Saturation logic prevents wrap-around but may lose information if too many values saturate.

14.6 TCP Transmission Delay Variation

TCP provides reliable delivery but does not provide deterministic latency. End-to-end runtime can vary due to operating system scheduling, Ethernet buffering, socket overhead, and network conditions. Therefore, cycle-count-based FPGA timing is a better measure of pure hardware compute time.

14.7 Register Access Overhead

The current design transfers input pixels through a memory-mapped register window. Each pixel is written as one 32-bit word, even though only the lower 8 bits are used. This was a deliberate choice because it made address decoding and readback testing straightforward, but it is not the fastest possible transfer method. DMA or shared memory could reduce this overhead.

14.8 Polling-Based Control Limitation

The HPS currently waits for the FPGA to finish by polling the status register. Polling is simple but consumes CPU time and introduces latency depending on polling interval. An interrupt-based completion mechanism would be more efficient.

14.9 FPGA Resource Limitation

The DE1-SoC Cyclone V FPGA has limited logic, DSP blocks, and on-chip memory. This limits the amount of CNN parallelism that can be implemented. The design reuses MAC hardware to save resources, but this increases the number of cycles per inference.

14.10 Timing Closure Limitation

Timing closure is another limitation of FPGA implementation. Long combinational paths can reduce the maximum clock frequency. In this design, address arithmetic and output comparison both had to be watched carefully. The tap-lookup and sequential argmax changes helped reduce critical paths, but further improvements may require more pipelining and memory access optimization.

14.11 Scalability Limitation

The current system is designed for a small LeNet-style classifier with 32×32 grayscale input and ten output classes. Scaling to larger images, larger CNNs, or real-time video would require DMA transfer, more memory bandwidth, more MAC lanes, and a more efficient communication protocol.

15 Future Work

The current system is functional and complete, but several parts could still be improved if the project were continued. The first direction would be reducing system-level latency. At the moment, the HPS writes 1024 pixels into the FPGA through individual MMIO accesses, which is simple and reliable but not very efficient. A DMA-based input path could move the full image as a bulk transfer. Similarly, replacing status polling with an FPGA interrupt would reduce wasted CPU time while the HPS waits for inference to finish.

The second direction would be improving compute throughput inside the FPGA. The current accelerator reuses a small number of MAC lanes to save resources, so convolution and fully connected layers take many cycles. Increasing the number of MAC lanes, adding more parallel processing elements, or

banking memories more aggressively could reduce the FPGA core cycle count, especially in Conv2 and FC1 where most of the multiply-accumulate work occurs.

The third direction would be improving the software-facing interface and recognition robustness. A persistent TCP connection or compact binary payload could reduce JSON serialization and connection setup overhead. On the accuracy side, more robust cropping, centering, contrast handling, and data augmentation would help the model handle off-center digits, unusual stroke widths, and ambiguous handwriting. After those improvements, the same pipeline could be extended from browser drawing and image upload to real-time camera input, although that would require faster preprocessing and more efficient data transfer.

16 Division of Work

16.1 Yizheng Tang

- **Software part:** built the web demo, Flask backend integration, sample saving, HPS client connection, and result/logit/timing display.
- **FPGA RTL part:** implemented and maintained the CNN core FSM, BRAM storage, MAC datapath integration, quantization connection, address calculation, and argmax logic.
- **Timing optimization part:** optimized the FPGA critical path in the later stage of the project, including replacing the long combinational argmax comparison with a sequential argmax scan, reducing timing pressure, improving Fmax, and checking the updated Quartus timing report.
- **HPS-FPGA interface part:** designed the compact register map and implemented the AXI register interface in `hps_lenet_regs.sv`.
- **HPS software part:** updated the C server logic for `/dev/mem`, `mmap()`, pixel upload, start/done polling, result readback, and logit readback.
- **Verification part:** created the RTL testbench, ran simulation, checked Quartus timing/resource reports, and debugged the full browser-to-FPGA demo.
- **Report and presentation part:** contributed to the report/PPT and took the main role in presentation and Q&A preparation.

16.2 Chenxi Shen

- **FPGA RTL part:** implemented key RTL modules for the later pipeline, including Conv2, quantization, and the FC1/FC2/FC3 computation flow, and completed their integration in the LeNet inference datapath.
- **Display/debug part:** implemented the seven-segment HEX display path to show prediction results directly on the board for hardware-level verification and demo usability.
- **Presentation part:** prepared the PPT sections for the above hardware modules, including architecture explanation and dataflow details.
- **Report writing part:** contributed to the final report by writing Sections 10 and 11, covering hardware-related content and implementation details.

16.3 Sirui Chen

- **CNN training part:** trained the network and exported the final weights and biases for FPGA use.
- **Hardware alignment part:** adjusted training and quantization so the exported parameters better aligned with hardware inference behavior.
- **Shift search part:** built a Python hardware-simulation testbench to evaluate different layer shift values and select suitable shifts for Conv1, Conv2, FC1, and FC2.
- **Verification part:** verified software/hardware consistency by comparing PyTorch inference results with hardware-style simulation and measuring prediction mismatch.
- **Report and slides part:** contributed to kernel parameter training, quantization, and shift-scaling documentation.

16.4 Weiwei Wu

- **Software part:** improved Flask-to-HPS data transfer and response return for prediction, logits, and timing display.
- **HPS-FPGA interface part:** improved the register map and AXI-side interface behavior for reliable control/status signaling and data exchange between HPS and FPGA.
- **HPS software part:** optimized the C-side MMIO workflow, including `/dev/mem` and `mmap()` access, pixel upload, start/done polling, and result/logit readback.
- **Report part:** contributed to the Software Input Processing and HPS Interface sections.

16.5 Tian Li

- **Second convolution layer RTL part:** implemented the hardware logic for the second convolution layer, including convolution computation and data control.
- **Hardware optimization part:** optimized the convolution datapath and control logic to reduce resource usage and improve timing.
- **FPGA integration part:** helped connect and debug the convolution module within the overall FPGA system.
- **Draw.io diagram part:** created the system architecture and hardware block diagrams for the report.
- **Verification part:** ran RTL simulation and checked hardware output correctness.
- **Report part:** organized the final report and wrote the hardware implementation and optimization sections.

17 Conclusion

17.1 Project Summary

This project implemented a complete FPGA-based CNN accelerator system for handwritten digit recognition on the DE1-SoC platform. The final demo accepts a digit drawn in a browser or uploaded as

an image, preprocesses it into a 32×32 grayscale pixel array, transfers the data to the HPS through Ethernet TCP communication, and performs CNN inference on the FPGA fabric.

17.2 Main Technical Achievements

The main achievements include training and quantizing a LeNet-style classifier, building a browser-based digit input interface, implementing TCP communication between Flask and HPS, developing a C inference server, designing a compact memory-mapped register map, implementing a custom INT8 CNN accelerator, and verifying the design through software reference comparison, RTL simulation, register testing, and end-to-end hardware testing. The final system also exposes logits and cycle counts, which made it easier to understand a prediction rather than only displaying a single digit.

The final design meets timing at the 50 MHz target clock, with a reported Fmax of 61.94 MHz. The measured FPGA path also achieves lower latency than the software-only reference path.

17.3 Main Challenges

The main challenges were HPS-FPGA communication, quantization, timing closure, and full system integration. The design required correct coordination between Linux software, `/dev/mem`, `mmap()`, AXI register decoding, and the FPGA accelerator. Matching FPGA behavior with the quantized software reference also required careful handling of scaling, saturation, and layer output formats.

17.4 Final Remarks

Overall, the project achieved its goal of building a complete FPGA-based CNN inference system for handwritten digit recognition. The design is still limited by register-based input transfer, polling-based control, and modest CNN parallelism, but those limitations are now clearly isolated. That makes the project a useful foundation for future improvements such as DMA transfer, interrupt-based completion, higher MAC parallelism, improved preprocessing, and real-time camera input.

18 References

1. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
2. Intel, "DE1-SoC User Manual," Terasic Technologies.
3. Intel, "Cyclone V Hard Processor System Technical Reference Manual."
4. Intel, "Quartus Prime Standard Edition Handbook."

A Final Source Code

This appendix includes the final hand-written project source files from `project_main`. Quartus-generated HDL, synthesis outputs, HPS handoff files, ModelSim work libraries, waveforms, transcripts, and runtime logs are intentionally excluded.

A.1 Web Demo and Software Reference Code

Listing 1: Flask web server and request handler.

```
from __future__ import annotations

import base64
import io
import math
import os
import time
from pathlib import Path

from flask import Flask, jsonify, render_template, request
from PIL import Image

try:
    from .predictor import DigitPredictor
    from .hps_client import HpsInferClient
except ImportError:
    from predictor import DigitPredictor
    from hps_client import HpsInferClient


BASE_DIR = Path(__file__).resolve().parent
SAMPLE_DIR = BASE_DIR / "saved_samples"

app = Flask(__name__)
predictor = DigitPredictor(project_root=BASE_DIR.parent)
USE_HPS_INFER = os.getenv("USE_HPS_INFER", "0") == "1"
HPS_HOST = os.getenv("HPS_HOST", "127.0.0.1")
HPS_PORT = int(os.getenv("HPS_PORT", "9090"))
FPGA_CLOCK_HZ = 50_000_000
FPGA_INFERENCE_CYCLES = 312_909
FPGA_COMPUTE_MS = FPGA_INFERENCE_CYCLES * 1000.0 / FPGA_CLOCK_HZ
hps_client = HpsInferClient(host=HPS_HOST, port=HPS_PORT)

print(
    f"[startup] USE_HPS_INFER={USE_HPS_INFER} "
    f"HPS_HOST={HPS_HOST} HPS_PORT={HPS_PORT}",
    flush=True,
)


def summarize_fpga_logits(logits: list[int]) -> tuple[float, list[dict[str, float | int]]]:
    if not logits:
        return 0.0, []

    max_logit = max(logits)
    max_abs_logit = max(abs(value) for value in logits) or 1
    temperature = max(1.0, max_abs_logit / 16.0)
    exps = [math.exp((value - max_logit) / temperature) for value in logits]
    total = sum(exps) or 1.0
    probabilities = [(value / total) * 100.0 for value in exps]

    ranked = sorted(
        (
            {
                "digit": digit,
                "score": logits[digit],
```

```
                "probability": round(probabilities[digit], 2),
            }
            for digit in range(len(logits))
        ),
        key=lambda item: item["score"],
        reverse=True,
    )
    return ranked[0]["probability"], ranked[:3]


@app.get("/")
def index() -> str:
    return render_template("index.html")


def decode_canvas_payload(payload: dict[str, object]) -> Image.Image:
    image_data = payload.get("image")
    if not isinstance(image_data, str) or "," not in image_data:
        raise ValueError("Missing canvas image payload.")

    _, encoded = image_data.split(",", 1)
    raw = base64.b64decode(encoded)
    return Image.open(io.BytesIO(raw)).convert("RGBA")


def elapsed_ms(start: float, end: float) -> float:
    return round((end - start) * 1000.0, 2)


@app.post("/predict")
def predict():
    payload = request.get_json(silent=True) or {}

    try:
        image = decode_canvas_payload(payload)
    except Exception as exc:  # pragma: no cover - defensive API response
        print(f"[predict] Error: {exc}", flush=True)
        return jsonify({"error": str(exc)}), 400

    software_total_start = time.perf_counter()
    processed = predictor.preprocess_canvas(image)
    pixels = list(processed.getdata())
    preview_image = predictor.render_preview_data_url(processed)

    exact_start = time.perf_counter()
    exact_logits = predictor._forward(pixels)
    exact_end = time.perf_counter()

    quantized_start = time.perf_counter()
    quantized_logits = predictor._forward_quantized(pixels)
    quantized_end = time.perf_counter()
    software_total_end = time.perf_counter()

    software_result = predictor._format_prediction(exact_logits, preview_image, "weights_bias_422")
    software_result["quantized_reference"] = predictor._format_prediction(
        quantized_logits,
        preview_image,
        "fpga_quantized_reference",
```

```
    )
    timing = {
        "software_exact_ms": elapsed_ms(exact_start, exact_end),
        "software_quantized_ms": elapsed_ms(quantized_start, quantized_end),
        "software_total_ms": elapsed_ms(software_total_start, software_total_end),
        "fpga_roundtrip_ms": None,
        "fpga_compute_ms": None,
        "fpga_cycles": None,
        "fpga_clock_hz": None,
        "speedup_vs_software": None,
        "compute_speedup_vs_software": None,
    }
    response_payload = {
        "preview_image": preview_image,
        "software_result": software_result,
        "software_quantized_result": software_result.get("quantized_reference"),
        "timing": timing,
    }

    if USE_HPS_INFER:
        print("[predict] Using HPS inference path.", flush=True)
        try:
            fpga_start = time.perf_counter()
            hps_result = hps_client.infer(pixels)
            fpga_end = time.perf_counter()
            timing["fpga_roundtrip_ms"] = elapsed_ms(fpga_start, fpga_end)
            debug_payload = hps_result.get("debug", {}) or {}
            fpga_compute_ms = debug_payload.get("fpga_compute_ms")
            fpga_cycles = debug_payload.get("fpga_cycles")
            fpga_clock_hz = debug_payload.get("fpga_clock_hz")
            if isinstance(fpga_compute_ms, (int, float)):
                timing["fpga_compute_ms"] = round(float(fpga_compute_ms), 4)
            else:
                timing["fpga_compute_ms"] = round(FPGA_COMPUTE_MS, 4)
            if isinstance(fpga_cycles, int):
                timing["fpga_cycles"] = fpga_cycles
            else:
                timing["fpga_cycles"] = FPGA_INFERENCE_CYCLES
            if isinstance(fpga_clock_hz, int):
                timing["fpga_clock_hz"] = fpga_clock_hz
            else:
                timing["fpga_clock_hz"] = FPGA_CLOCK_HZ
            if timing["fpga_roundtrip_ms"] and timing["fpga_roundtrip_ms"] > 0:
                timing["speedup_vs_software"] = round(
                    timing["software_exact_ms"] / timing["fpga_roundtrip_ms"],
                    2,
                )
            if timing["fpga_compute_ms"] and timing["fpga_compute_ms"] > 0:
                timing["compute_speedup_vs_software"] = round(
                    timing["software_exact_ms"] / timing["fpga_compute_ms"],
                    2,
                )
            print(f"[predict] HPS response: {hps_result}", flush=True)
            logits = debug_payload.get("logits", []) or []
            confidence, matches = summarize_fpga_logits(logits)
            prediction = hps_result.get("prediction", "-")
            if isinstance(prediction, int) and prediction < 0:
                prediction = "-"
```

```
            response_payload["fpga_result"] = {
                "prediction": prediction,
                "confidence": confidence,
                "matches": matches,
                "debug": hps_result.get("debug"),
                "note": hps_result.get("note"),
                "model": {
                    "name": "hps_fpga_pipeline",
                    "architecture": "Website␣preprocess␣+␣Ethernet␣+␣HPS␣+␣FPGA␣LeNet",
                },
            }
        except Exception as exc:  # pragma: no cover - defensive API response
            print(f"[predict]␣FPGA␣path␣error:␣{exc}", flush=True)
            response_payload["fpga_result"] = {
                "prediction": "!",
                "confidence": 0.0,
                "matches": [],
                "debug": None,
                "note": str(exc),
                "model": {
                    "name": "fpga_error",
                    "architecture": "HPS/FPGA␣path␣failed␣during␣this␣request.",
                },
            }
    else:
        print("[predict]␣Using␣local␣software␣predictor␣path.", flush=True)
        response_payload["fpga_result"] = {
            "prediction": "-",
            "confidence": 0.0,
            "matches": [],
            "debug": None,
            "note": "FPGA␣path␣disabled.",
            "model": {
                "name": "fpga_path_disabled",
                "architecture": "Enable␣USE_HPS_INFER␣to␣compare␣hardware␣output.",
            },
        }

    return jsonify(response_payload)


@app.post("/save_sample")
def save_sample():
    payload = request.get_json(silent=True) or {}
    label = payload.get("label")

    if label not in [str(digit) for digit in range(10)]:
        return jsonify({"error": "Choose␣a␣digit␣label␣before␣saving."}), 400

    try:
        image = decode_canvas_payload(payload)
    except Exception as exc:  # pragma: no cover - defensive API response
        print(f"[save_sample]␣Error:␣{exc}", flush=True)
        return jsonify({"error": str(exc)}), 400

    target_dir = SAMPLE_DIR / f"digit_{label}"
    target_dir.mkdir(parents=True, exist_ok=True)
    filename = f"sample_{time.strftime('%Y%m%d_%H%M%S')}_{time.time_ns()␣%␣1000000:06d}.png"
```

```
    target_path = target_dir / filename
    image.save(target_path)

    return jsonify({
        "saved": True,
        "label": label,
        "path": str(target_path.relative_to(BASE_DIR)),
    })


if __name__ == "__main__":
    app.run(debug=False)
```

Listing 2: TCP client used to send inference requests to the HPS server.

```
from __future__ import annotations

import json
import socket
import uuid
from typing import Any


class HpsInferClient:
    def __init__(self, host: str, port: int, timeout_seconds: float = 35.0):
        self.host = host
        self.port = port
        self.timeout_seconds = timeout_seconds

    def infer(self, pixels: list[int], width: int = 32, height: int = 32) -> dict[str, Any]:
        if len(pixels) != width * height:
            raise ValueError(f"Expected {width * height} pixels, got {len(pixels)}.")

        request = {
            "command": "infer",
            "request_id": str(uuid.uuid4()),
            "width": width,
            "height": height,
            "pixels": pixels,
        }

        with socket.create_connection((self.host, self.port), timeout=self.timeout_seconds) as sock:
            sock.sendall((json.dumps(request) + "\n").encode("utf-8"))
            chunks: list[bytes] = []
            while True:
                chunk = sock.recv(16384)
                if not chunk:
                    break
                chunks.append(chunk)
                if b"\n" in chunk:
                    break

        response = b"".join(chunks).split(b"\n", 1)[0]

        if not response:
            raise RuntimeError("No response from HPS inference service.")

        payload = json.loads(response.decode("utf-8"))
```

```
        if not payload.get("ok"):
            error = payload.get("error", "unknown_hps_error")
            stage = payload.get("stage")
            if stage:
                raise RuntimeError(f"{error}␣({stage})")
            raise RuntimeError(error)
        return payload
```

Listing 3: Software LeNet reference and quantized reference implementation.

```
from __future__ import annotations

import base64
import io
import math
from dataclasses import dataclass
from pathlib import Path

from PIL import Image, ImageFilter


def stretch_contrast(img: Image.Image) -> Image.Image:
    pixels = list(img.getdata())
    lo = min(pixels)
    hi = max(pixels)
    if hi <= lo:
        return img.copy()
    scale = 255.0 / (hi - lo)
    return img.point(lambda value: int(round((value - lo) * scale)))


def denoise_image(img: Image.Image) -> Image.Image:
    return img.filter(ImageFilter.MedianFilter(size=3))


def find_foreground_bbox(
    img: Image.Image,
    threshold: int,
) -> tuple[int, int, int, int] | None:
    pixels = img.load()
    width, height = img.size
    left, top = width, height
    right, bottom = -1, -1

    for y in range(height):
        for x in range(width):
            if pixels[x, y] >= threshold:
                left = min(left, x)
                top = min(top, y)
                right = max(right, x)
                bottom = max(bottom, y)

    if right == -1:
        return None
    return left, top, right + 1, bottom + 1


def add_bbox_margin(
```

```
    bbox: tuple[int, int, int, int],
    image_size: tuple[int, int],
    margin: int,
) -> tuple[int, int, int, int]:
    left, top, right, bottom = bbox
    width, height = image_size
    return (
        max(0, left - margin),
        max(0, top - margin),
        min(width, right + margin),
        min(height, bottom + margin),
    )


def center_by_mass(img: Image.Image) -> Image.Image:
    pixels = list(img.getdata())
    width, height = img.size
    total = sum(pixels)
    if total == 0:
        return img

    cx = sum((idx % width) * value for idx, value in enumerate(pixels)) / total
    cy = sum((idx // width) * value for idx, value in enumerate(pixels)) / total

    shift_x = int(round((width - 1) / 2.0 - cx))
    shift_y = int(round((height - 1) / 2.0 - cy))

    shifted = Image.new("L", (width, height), color=0)
    shifted.paste(img, (shift_x, shift_y))
    return shifted


def preprocess_digit(
    img: Image.Image,
    out_size: int = 32,
    inner_size: int = 24,
    bbox_threshold: int = 24,
    crop_margin: int = 2,
) -> Image.Image:
    working = img.convert("L")
    working = denoise_image(working)
    working = stretch_contrast(working)

    bbox = find_foreground_bbox(working, bbox_threshold)
    if bbox is None:
        raise ValueError("No␣foreground␣digit␣detected.␣Please␣draw␣a␣number␣first.")

    cropped = working.crop(add_bbox_margin(bbox, working.size, crop_margin))

    scale = min(inner_size / cropped.width, inner_size / cropped.height)
    new_width = max(1, int(round(cropped.width * scale)))
    new_height = max(1, int(round(cropped.height * scale)))
    resized = cropped.resize((new_width, new_height), Image.Resampling.LANCZOS)

    canvas = Image.new("L", (out_size, out_size), color=0)
    offset_x = (out_size - resized.width) // 2
    offset_y = (out_size - resized.height) // 2
    canvas.paste(resized, (offset_x, offset_y))
```

```
107
108     canvas = center_by_mass(canvas)
109     canvas = canvas.filter(ImageFilter.GaussianBlur(radius=0.35))
110     return stretch_contrast(canvas)
111
112
113 def image_to_vector(img: Image.Image) -> list[int]:
114     return list(img.getdata())
115
116
117 def parse_int8_hex_file(path: Path) -> list[int]:
118     values: list[int] = []
119     for line in path.read_text(encoding="utf-8").splitlines():
120         item = line.strip()
121         if not item:
122             continue
123         raw = int(item, 16)
124         if raw >= 128:
125             raw -= 256
126         values.append(raw)
127     return values
128
129
130 def max_pool_2x2(
131     feature_map: list[int],
132     channels: int,
133     input_size: int,
134 ) -> tuple[list[int], int]:
135     output_size = input_size // 2
136     pooled = [0] * (channels * output_size * output_size)
137
138     for channel in range(channels):
139         channel_offset = channel * input_size * input_size
140         pooled_offset = channel * output_size * output_size
141         for out_y in range(output_size):
142             in_y = out_y * 2
143             for out_x in range(output_size):
144                 in_x = out_x * 2
145                 base = channel_offset + in_y * input_size + in_x
146                 pooled[pooled_offset + out_y * output_size + out_x] = max(
147                     feature_map[base],
148                     feature_map[base + 1],
149                     feature_map[base + input_size],
150                     feature_map[base + input_size + 1],
151                 )
152     return pooled, output_size
153
154
155 def conv_valid(
156     inputs: list[int],
157     input_channels: int,
158     input_size: int,
159     output_channels: int,
160     kernel_size: int,
161     weights: list[int],
162     biases: list[int],
163 ) -> tuple[list[int], int]:
164     output_size = input_size - kernel_size + 1
```

```
    outputs = [0] * (output_channels * output_size * output_size)
    kernel_area = kernel_size * kernel_size

    for out_channel in range(output_channels):
        out_channel_offset = out_channel * output_size * output_size
        weight_base = out_channel * input_channels * kernel_area
        bias = biases[out_channel]

        for out_y in range(output_size):
            for out_x in range(output_size):
                acc = bias
                for in_channel in range(input_channels):
                    in_channel_offset = in_channel * input_size * input_size
                    channel_weight_base = weight_base + in_channel * kernel_area
                    for ky in range(kernel_size):
                        input_row = in_channel_offset + (out_y + ky) * input_size + out_x
                        kernel_row = channel_weight_base + ky * kernel_size
                        for kx in range(kernel_size):
                            acc += inputs[input_row + kx] * weights[kernel_row + kx]
                outputs[out_channel_offset + out_y * output_size + out_x] = max(0, acc)

    return outputs, output_size


def dense(
    inputs: list[int],
    outputs_count: int,
    weights: list[int],
    biases: list[int],
    apply_relu: bool,
) -> list[int]:
    input_count = len(inputs)
    outputs = [0] * outputs_count

    for out_index in range(outputs_count):
        weight_base = out_index * input_count
        acc = biases[out_index]
        for input_index, input_value in enumerate(inputs):
            acc += input_value * weights[weight_base + input_index]
        outputs[out_index] = max(0, acc) if apply_relu else acc

    return outputs


def softmax_percentages(logits: list[int]) -> list[float]:
    max_logit = max(logits)
    max_abs_logit = max(abs(value) for value in logits) or 1
    temperature = max(1.0, max_abs_logit / 16.0)
    exps = [math.exp((value - max_logit) / temperature) for value in logits]
    total = sum(exps) or 1.0
    return [(value / total) * 100.0 for value in exps]


def quantize_relu(value: int, shift_right: int, clamp_max: int = 255) -> int:
    if value <= 0:
        return 0
    if shift_right:
        value += 1 << (shift_right - 1)
```

```
        value >>= shift_right
    return max(0, min(clamp_max, value))


@dataclass(frozen=True)
class Int8Layer:
    weights: list[int]
    biases: list[int]


@dataclass(frozen=True)
class QuantizationConfig:
    act_bits: int = 16
    conv1_shift: int = 6
    conv2_shift: int = 8
    fc1_shift: int = 8
    fc2_shift: int = 7

    @property
    def act_max(self) -> int:
        return (1 << self.act_bits) - 1


@dataclass(frozen=True)
class Int8LeNetModel:
    conv1: Int8Layer
    conv2: Int8Layer
    fc1: Int8Layer
    fc2: Int8Layer
    fc3: Int8Layer


class DigitPredictor:
    def __init__(self, project_root: Path):
        self.project_root = project_root
        self.model_dir = self._resolve_model_dir()
        self.model = self._load_model()
        self.quant = QuantizationConfig()

    def _resolve_model_dir(self) -> Path:
        candidates = [
            self.project_root.parent / "parameter" / "weights_bias_422",
            self.project_root / "parameter" / "weights_bias_422",
        ]
        for candidate in candidates:
            if candidate.exists():
                return candidate
        raise FileNotFoundError("Could not find parameter/weights_bias_422 for the web demo.")

    def _split_layer(self, values: list[int], weight_count: int, bias_count: int) -> Int8Layer:
        expected = weight_count + bias_count
        if len(values) != expected:
            raise ValueError(f"Expected {expected} values but found {len(values)}.")
        return Int8Layer(
            weights=values[:weight_count],
            biases=values[weight_count:],
        )

```

```
    def _load_model(self) -> Int8LeNetModel:
        conv1 = self._split_layer(parse_int8_hex_file(self.model_dir / "conv1.hex"), 6 * 5 * 5, 6)
        conv2 = self._split_layer(parse_int8_hex_file(self.model_dir / "conv2.hex"), 16 * 6 * 5 * 5,
     16)
        fc1 = self._split_layer(parse_int8_hex_file(self.model_dir / "fc1.hex"), 120 * 400, 120)
        fc2 = self._split_layer(parse_int8_hex_file(self.model_dir / "fc2.hex"), 84 * 120, 84)
        fc3 = self._split_layer(parse_int8_hex_file(self.model_dir / "fc3.hex"), 10 * 84, 10)
        return Int8LeNetModel(conv1=conv1, conv2=conv2, fc1=fc1, fc2=fc2, fc3=fc3)

    def _forward(self, vector: list[int]) -> list[int]:
        conv1_out, conv1_size = conv_valid(
            inputs=vector,
            input_channels=1,
            input_size=32,
            output_channels=6,
            kernel_size=5,
            weights=self.model.conv1.weights,
            biases=self.model.conv1.biases,
        )
        pool1_out, pool1_size = max_pool_2x2(conv1_out, channels=6, input_size=conv1_size)

        conv2_out, conv2_size = conv_valid(
            inputs=pool1_out,
            input_channels=6,
            input_size=pool1_size,
            output_channels=16,
            kernel_size=5,
            weights=self.model.conv2.weights,
            biases=self.model.conv2.biases,
        )
        pool2_out, _ = max_pool_2x2(conv2_out, channels=16, input_size=conv2_size)

        fc1_out = dense(pool2_out, 120, self.model.fc1.weights, self.model.fc1.biases, apply_relu=
    True)
        fc2_out = dense(fc1_out, 84, self.model.fc2.weights, self.model.fc2.biases, apply_relu=True)
        return dense(fc2_out, 10, self.model.fc3.weights, self.model.fc3.biases, apply_relu=False)

    def _forward_quantized(self, vector: list[int]) -> list[int]:
        quant = self.quant
        conv1_out, conv1_size = conv_valid(
            inputs=vector,
            input_channels=1,
            input_size=32,
            output_channels=6,
            kernel_size=5,
            weights=self.model.conv1.weights,
            biases=self.model.conv1.biases,
        )
        conv1_q = [quantize_relu(value, quant.conv1_shift, quant.act_max) for value in conv1_out]
        pool1_out, pool1_size = max_pool_2x2(conv1_q, channels=6, input_size=conv1_size)

        conv2_out, conv2_size = conv_valid(
            inputs=pool1_out,
            input_channels=6,
            input_size=pool1_size,
            output_channels=16,
            kernel_size=5,
            weights=self.model.conv2.weights,
```

```
            biases=self.model.conv2.biases,
        )
        conv2_q = [quantize_relu(value, quant.conv2_shift, quant.act_max) for value in conv2_out]
        pool2_out, _ = max_pool_2x2(conv2_q, channels=16, input_size=conv2_size)

        fc1_raw = dense(pool2_out, 120, self.model.fc1.weights, self.model.fc1.biases, apply_relu=False)
        fc1_out = [quantize_relu(value, quant.fc1_shift, quant.act_max) for value in fc1_raw]
        fc2_raw = dense(fc1_out, 84, self.model.fc2.weights, self.model.fc2.biases, apply_relu=False)
        fc2_out = [quantize_relu(value, quant.fc2_shift, quant.act_max) for value in fc2_raw]
        return dense(fc2_out, 10, self.model.fc3.weights, self.model.fc3.biases, apply_relu=False)

    def _format_prediction(self, logits: list[int], preview_image: str, model_name: str) -> dict[str, object]:
        probabilities = softmax_percentages(logits)

        ranked = sorted(
            (
                {
                    "digit": digit,
                    "score": logits[digit],
                    "probability": round(probabilities[digit], 2),
                }
                for digit in range(10)
            ),
            key=lambda item: item["score"],
            reverse=True,
        )

        best = ranked[0]
        return {
            "prediction": best["digit"],
            "confidence": best["probability"],
            "matches": ranked[:3],
            "preview_image": preview_image,
            "logits": logits,
            "model": {
                "name": model_name,
                "architecture": f"LeNet-style␣quantized␣CNN␣({self.quant.act_bits}-bit␣activations,␣8-bit␣weights)",
            },
        }

    def preprocess_canvas(self, canvas_image: Image.Image) -> Image.Image:
        black_bg = Image.new("RGBA", canvas_image.size, (0, 0, 0, 255))
        black_bg.alpha_composite(canvas_image)
        return preprocess_digit(black_bg.convert("RGB"))

    def render_preview_data_url(self, processed: Image.Image) -> str:
        preview = processed.resize((160, 160), Image.Resampling.NEAREST)
        preview_buffer = io.BytesIO()
        preview.save(preview_buffer, format="PNG")
        preview_base64 = base64.b64encode(preview_buffer.getvalue()).decode("ascii")
        return f"{preview_base64}"

    def predict(self, canvas_image: Image.Image) -> dict[str, object]:
        processed = self.preprocess_canvas(canvas_image)
```

```
        vector = image_to_vector(processed)
        preview_image = self.render_preview_data_url(processed)
        exact = self._format_prediction(self._forward(vector), preview_image, "weights_bias_422")
        exact["quantized_reference"] = self._format_prediction(
            self._forward_quantized(vector),
            preview_image,
            "fpga_quantized_reference",
        )
        return exact
```

Listing 4: Browser interface HTML template.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>FPGA Handwritten Digit Recognition</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}" />
  </head>
  <body>
    <main class="page">
      <header class="topbar">
        <div>
          <h1>FPGA Digit Lab</h1>
        </div>
        <div class="status-strip" aria-label="Pipeline status">
          <span>Web preprocess</span>
          <span>Python reference</span>
          <span>HPS FPGA</span>
        </div>
      </header>

      <section class="console-grid">
        <section class="panel input-panel">
          <div class="panel-heading">
            <p class="eyebrow">Input</p>
            <h2>Drawing Pad</h2>
          </div>

          <div class="canvas-shell">
            <canvas id="draw-canvas" width="360" height="360" aria-label="Digit drawing canvas"></
    canvas>
          </div>

          <div class="control-row brush-control">
            <label for="brush-size-slider">Stroke</label>
            <input id="brush-size-slider" type="range" min="4" max="28" step="1" value="12" />
            <span id="brush-size-value">12 px</span>
          </div>

          <div class="control-row invert-control">
            <span>Invert</span>
            <label class="switch" for="invert-toggle">
              <input id="invert-toggle" type="checkbox" />
              <span class="switch-track"></span>
            </label>
            <span id="invert-status">Off</span>
```

```
          </div>

          <div class="actions">
            <button id="predict-btn" class="primary-btn">Predict</button>
            <label class="upload-btn" for="upload-image-input">Upload</label>
            <input id="upload-image-input" type="file" accept="image/*" />
            <button id="clear-btn" class="ghost-btn">Clear</button>
          </div>

          <div class="label-save-row">
            <label for="sample-label-select">Label</label>
            <select id="sample-label-select">
              <option value="">Choose</option>
              <option value="0">0</option>
              <option value="1">1</option>
              <option value="2">2</option>
              <option value="3">3</option>
              <option value="4">4</option>
              <option value="5">5</option>
              <option value="6">6</option>
              <option value="7">7</option>
              <option value="8">8</option>
              <option value="9">9</option>
            </select>
            <button id="save-sample-btn" class="ghost-btn" type="button">Save</button>
          </div>
          <p id="save-status" class="save-status">Not saved</p>

        </section>

        <section class="panel results-panel">
          <div class="panel-heading">
            <p class="eyebrow">Output</p>
            <h2>Inference Results</h2>
          </div>

          <div class="result-stack">
            <article class="result-block">
              <p id="fpga-result-label" class="result-label">FPGA Result</p>
              <div class="prediction-row">
                <div id="fpga-prediction-value" class="prediction-value">-</div>
                <div class="prediction-meta">
                  <p id="fpga-confidence-text">Top-1 probability: --</p>
                  <p id="fpga-top-match-text">Top scores: --</p>
                </div>
              </div>
            </article>

            <article class="result-block compact-result">
              <p id="software-quant-result-label" class="result-label">Quantized Software Reference
    </p>
              <div class="prediction-row">
                <div id="software-quant-prediction-value" class="prediction-value">-</div>
                <div class="prediction-meta">
                  <p id="software-quant-confidence-text">Top-1 probability: --</p>
                  <p id="software-quant-top-match-text">Top scores: --</p>
                </div>
              </div>
```

```
            </article>

            <article class="result-block compact-result">
              <p id="software-result-label" class="result-label">Software Result</p>
              <div class="prediction-row">
                <div id="software-prediction-value" class="prediction-value">-</div>
                <div class="prediction-meta">
                  <p id="software-confidence-text">Top-1 probability: --</p>
                  <p id="software-top-match-text">Top scores: --</p>
                </div>
              </div>
            </article>
          </div>
        </section>
      </section>

      <section class="analysis-grid">
        <section class="panel preview-block">
          <div class="panel-heading">
            <p class="eyebrow">Processed Input</p>
            <h2>32x32 Preview</h2>
          </div>
          <div class="preview-frame">
            <canvas id="preview-canvas" width="160" height="160"></canvas>
          </div>
        </section>

        <section class="panel debug-block">
          <div class="panel-heading">
            <p class="eyebrow">FPGA Output</p>
            <h2>Final Logits</h2>
          </div>
          <p id="debug-summary" class="debug-summary">Waiting for FPGA debug data.</p>
          <div id="debug-grid" class="debug-grid"></div>
        </section>

        <section class="panel performance-panel">
          <div class="panel-heading">
            <p class="eyebrow">Performance</p>
            <h2>Speed Compare</h2>
          </div>
          <div class="metric-grid">
            <div>
              <span>CPU exact</span>
              <strong id="software-exact-time">--</strong>
            </div>
            <div>
              <span>CPU quant</span>
              <strong id="software-quant-time">--</strong>
            </div>
            <div>
              <span>FPGA path</span>
              <strong id="fpga-time">--</strong>
            </div>
            <div>
              <span>FPGA core</span>
              <strong id="fpga-core-time">--</strong>
            </div>
```

```
            <div>
              <span>Speedup</span>
              <strong id="speedup-value">--</strong>
            </div>
          </div>
          <div class="speed-chart" aria-label="Speed comparison chart">
            <div class="speed-bar-group">
              <div class="speed-bar-track">
                <div id="software-exact-bar" class="speed-bar cpu-exact-bar"></div>
              </div>
              <span>Exact</span>
            </div>
            <div class="speed-bar-group">
              <div class="speed-bar-track">
                <div id="software-quant-bar" class="speed-bar cpu-quant-bar"></div>
              </div>
              <span>Quant</span>
            </div>
            <div class="speed-bar-group">
              <div class="speed-bar-track">
                <div id="fpga-bar" class="speed-bar fpga-bar"></div>
              </div>
              <span>FPGA</span>
            </div>
          </div>
        </section>
      </section>
    </main>

    <script src="{{ url_for('static', filename='app.js') }}"></script>
  </body>
</html>
```

Listing 5: Browser-side JavaScript for drawing, upload, prediction requests, and UI updates.

```
const drawCanvas = document.getElementById("draw-canvas");
const drawCtx = drawCanvas.getContext("2d");
const previewCanvas = document.getElementById("preview-canvas");
const previewCtx = previewCanvas.getContext("2d");
const predictBtn = document.getElementById("predict-btn");
const clearBtn = document.getElementById("clear-btn");
const uploadImageInput = document.getElementById("upload-image-input");
const invertToggle = document.getElementById("invert-toggle");
const invertStatus = document.getElementById("invert-status");
const brushSizeSlider = document.getElementById("brush-size-slider");
const brushSizeValue = document.getElementById("brush-size-value");
const sampleLabelSelect = document.getElementById("sample-label-select");
const saveSampleBtn = document.getElementById("save-sample-btn");
const saveStatus = document.getElementById("save-status");
const softwareResultLabel = document.getElementById("software-result-label");
const softwarePredictionValue = document.getElementById("software-prediction-value");
const softwareConfidenceText = document.getElementById("software-confidence-text");
const softwareTopMatchText = document.getElementById("software-top-match-text");
const softwareQuantResultLabel = document.getElementById("software-quant-result-label");
const softwareQuantPredictionValue = document.getElementById("software-quant-prediction-value");
const softwareQuantConfidenceText = document.getElementById("software-quant-confidence-text");
const softwareQuantTopMatchText = document.getElementById("software-quant-top-match-text");
const fpgaResultLabel = document.getElementById("fpga-result-label");
```

```
const fpgaPredictionValue = document.getElementById("fpga-prediction-value");
const fpgaConfidenceText = document.getElementById("fpga-confidence-text");
const fpgaTopMatchText = document.getElementById("fpga-top-match-text");
const softwareExactTime = document.getElementById("software-exact-time");
const softwareQuantTime = document.getElementById("software-quant-time");
const fpgaTime = document.getElementById("fpga-time");
const fpgaCoreTime = document.getElementById("fpga-core-time");
const speedupValue = document.getElementById("speedup-value");
const softwareExactBar = document.getElementById("software-exact-bar");
const softwareQuantBar = document.getElementById("software-quant-bar");
const fpgaBar = document.getElementById("fpga-bar");
const debugSummary = document.getElementById("debug-summary");
const debugGrid = document.getElementById("debug-grid");

let drawing = false;
let brushSize = Number(brushSizeSlider.value);

function applyInvertStatus() {
  invertStatus.textContent = invertToggle.checked ? "On" : "Off";
}

function applyBrushSize() {
  brushSize = Number(brushSizeSlider.value);
  drawCtx.lineWidth = brushSize;
  brushSizeValue.textContent = `${brushSize} px`;
}

function resetCanvas() {
  drawCtx.fillStyle = "#000000";
  drawCtx.fillRect(0, 0, drawCanvas.width, drawCanvas.height);
  drawCtx.lineCap = "round";
  drawCtx.lineJoin = "round";
  drawCtx.strokeStyle = "#ffffff";
  drawCtx.lineWidth = brushSize;
}

function resetSaveStatus() {
  saveStatus.textContent = "Not saved";
  saveStatus.style.color = "";
}

function clearPreview() {
  previewCtx.fillStyle = "#000000";
  previewCtx.fillRect(0, 0, previewCanvas.width, previewCanvas.height);
}

function getCanvasPayload() {
  if (!invertToggle.checked) {
    return drawCanvas.toDataURL("image/png");
  }

  const tempCanvas = document.createElement("canvas");
  tempCanvas.width = drawCanvas.width;
  tempCanvas.height = drawCanvas.height;
  const tempCtx = tempCanvas.getContext("2d");
  tempCtx.drawImage(drawCanvas, 0, 0);
  const imageData = tempCtx.getImageData(0, 0, tempCanvas.width, tempCanvas.height);
  const data = imageData.data;
```

```
82
83    for (let index = 0; index < data.length; index += 4) {
84      data[index] = 255 - data[index];
85      data[index + 1] = 255 - data[index + 1];
86      data[index + 2] = 255 - data[index + 2];
87      data[index + 3] = 255;
88    }
89
90    tempCtx.putImageData(imageData, 0, 0);
91    return tempCanvas.toDataURL("image/png");
92  }
93
94  function resetResultCards() {
95    softwarePredictionValue.textContent = "-";
96    softwareConfidenceText.textContent = "Top-1 probability: --";
97    softwareTopMatchText.textContent = "Top scores: --";
98    softwareResultLabel.textContent = "Software Result";
99    softwareQuantPredictionValue.textContent = "-";
100   softwareQuantConfidenceText.textContent = "Top-1 probability: --";
101   softwareQuantTopMatchText.textContent = "Top scores: --";
102   softwareQuantResultLabel.textContent = "Quantized Software Reference";
103   fpgaPredictionValue.textContent = "-";
104   fpgaConfidenceText.textContent = "Top-1 probability: --";
105   fpgaTopMatchText.textContent = "Top scores: --";
106   fpgaResultLabel.textContent = "FPGA Result";
107   softwareExactTime.textContent = "--";
108   softwareQuantTime.textContent = "--";
109   fpgaTime.textContent = "--";
110   fpgaCoreTime.textContent = "--";
111   speedupValue.textContent = "--";
112   softwareExactBar.style.height = "0%";
113   softwareQuantBar.style.height = "0%";
114   fpgaBar.style.height = "0%";
115   clearDebugView();
116 }
117
118 function getPoint(event) {
119   const rect = drawCanvas.getBoundingClientRect();
120   const touch = event.touches ? event.touches[0] : event;
121   const scaleX = drawCanvas.width / rect.width;
122   const scaleY = drawCanvas.height / rect.height;
123   return {
124     x: (touch.clientX - rect.left) * scaleX,
125     y: (touch.clientY - rect.top) * scaleY,
126   };
127 }
128
129 function startDrawing(event) {
130   drawing = true;
131   const { x, y } = getPoint(event);
132   drawCtx.beginPath();
133   drawCtx.moveTo(x, y);
134   event.preventDefault();
135 }
136
137 function draw(event) {
138   if (!drawing) {
139     return;
```

```
  }
  const { x, y } = getPoint(event);
  drawCtx.lineTo(x, y);
  drawCtx.stroke();
  event.preventDefault();
}

function stopDrawing() {
  drawing = false;
}

function renderPreview(dataUrl) {
  const previewImage = new Image();
  previewImage.onload = () => {
    previewCtx.imageSmoothingEnabled = false;
    previewCtx.clearRect(0, 0, previewCanvas.width, previewCanvas.height);
    previewCtx.drawImage(previewImage, 0, 0, previewCanvas.width, previewCanvas.height);
  };
  previewImage.src = dataUrl;
}

function toTitleCase(text) {
  return text
    .replace(/[_-]+/g, " ")
    .replace(/\b\w/g, (char) => char.toUpperCase());
}

function clearDebugView(message = "Waiting for FPGA debug data.") {
  debugSummary.textContent = message;
  debugGrid.innerHTML = "";
}

function renderLogitDebug(logits) {
  debugSummary.textContent = "Showing final FPGA classification logits returned by the HPS/FPGA
    pipeline.";
  debugGrid.innerHTML = "";

  const card = document.createElement("article");
  card.className = "debug-card";

  const title = document.createElement("p");
  title.className = "debug-card-title";
  title.textContent = "Final Logits";

  const dataBlock = document.createElement("pre");
  dataBlock.className = "debug-card-data";
  dataBlock.textContent = logits
    .map((value, index) => `digit ${index}: ${value}`)
    .join("\n");

  card.appendChild(title);
  card.appendChild(dataBlock);
  debugGrid.appendChild(card);
}

function formatHardwareTiming(debug) {
  if (typeof debug?.fpga_cycles !== "number") {
    return "FPGA core timing: unavailable";
```

```
197   }
198
199   const clockMhz =
200     typeof debug?.fpga_clock_hz === "number"
201       ? (debug.fpga_clock_hz / 1000000).toFixed(2)
202       : "--";
203   const computeMs =
204     typeof debug?.fpga_compute_ms === "number"
205       ? debug.fpga_compute_ms.toFixed(4)
206       : "--";
207   return `FPGA core timing: ${debug.fpga_cycles} cycles @ ${clockMhz} MHz = ${computeMs} ms`;
208 }
209
210 function renderResultCard(prefix, result) {
211   let labelEl;
212   let predictionEl;
213   let confidenceEl;
214   let matchesEl;
215
216   if (prefix === "software") {
217     labelEl = softwareResultLabel;
218     predictionEl = softwarePredictionValue;
219     confidenceEl = softwareConfidenceText;
220     matchesEl = softwareTopMatchText;
221   } else if (prefix === "software_quantized") {
222     labelEl = softwareQuantResultLabel;
223     predictionEl = softwareQuantPredictionValue;
224     confidenceEl = softwareQuantConfidenceText;
225     matchesEl = softwareQuantTopMatchText;
226   } else {
227     labelEl = fpgaResultLabel;
228     predictionEl = fpgaPredictionValue;
229     confidenceEl = fpgaConfidenceText;
230     matchesEl = fpgaTopMatchText;
231   }
232
233   labelEl.textContent = `${toTitleCase(prefix)} Result From ${toTitleCase(result?.model?.name || "
        unknown_source")}`;
234   predictionEl.textContent = result?.prediction ?? "-";
235   confidenceEl.textContent = `Top-1 probability: ${result?.confidence ?? 0}%`;
236
237   const nearest = (result?.matches || [])
238     .map((match) => `${match.digit} (score ${match.score}, ${match.probability}%)`)
239     .join(" | ");
240
241   if (nearest) {
242     matchesEl.textContent = `Top scores: ${nearest}`;
243   } else if (result?.note) {
244     matchesEl.textContent = `Status: ${result.note}`;
245   } else {
246     matchesEl.textContent = "Top scores: unavailable";
247   }
248 }
249
250 function renderDebugView(debug) {
251   const logits = debug?.logits;
252   if (debug?.logits_valid === false) {
```

```
253     clearDebugView(`FPGA completed the run, but final logits were not marked valid yet. ${
        formatHardwareTiming(debug)}`);
254     return;
255   }
256
257   if (Array.isArray(logits) && logits.length) {
258     renderLogitDebug(logits);
259     debugSummary.textContent = `Showing final FPGA classification logits returned by the HPS/FPGA
        pipeline. ${formatHardwareTiming(debug)}`;
260     return;
261   }
262
263   clearDebugView("No final FPGA logits returned.");
264 }
265
266 function formatMs(value) {
267   if (typeof value !== "number") {
268     return "--";
269   }
270   return `${value.toFixed(2)} ms`;
271 }
272
273 function renderTiming(timing) {
274   softwareExactTime.textContent = formatMs(timing?.software_exact_ms);
275   softwareQuantTime.textContent = formatMs(timing?.software_quantized_ms);
276   fpgaTime.textContent = formatMs(timing?.fpga_roundtrip_ms);
277   fpgaCoreTime.textContent = formatMs(timing?.fpga_compute_ms);
278
279   if (typeof timing?.speedup_vs_software === "number") {
280     speedupValue.textContent = `${timing.speedup_vs_software.toFixed(2)}x`;
281   } else if (timing?.fpga_roundtrip_ms === null) {
282     speedupValue.textContent = "FPGA off";
283   } else {
284     speedupValue.textContent = "--";
285   }
286
287   const values = [
288     timing?.software_exact_ms,
289     timing?.software_quantized_ms,
290     timing?.fpga_roundtrip_ms,
291   ].filter((value) => typeof value === "number" && value > 0);
292   const maxValue = values.length ? Math.max(...values) : 0;
293   const bars = [
294     [softwareExactBar, timing?.software_exact_ms],
295     [softwareQuantBar, timing?.software_quantized_ms],
296     [fpgaBar, timing?.fpga_roundtrip_ms],
297   ];
298
299   bars.forEach(([bar, value]) => {
300     if (typeof value !== "number" || value <= 0 || maxValue <= 0) {
301       bar.style.height = "0%";
302       return;
303     }
304     bar.style.height = `${Math.max(8, (value / maxValue) * 100)}%`;
305   });
306 }
307
308 async function predictDigit() {
```

```
  softwarePredictionValue.textContent = "...";
  softwareConfidenceText.textContent = "Top-1 probability: running";
  softwareTopMatchText.textContent = "Top scores: calculating";
  softwareQuantPredictionValue.textContent = "...";
  softwareQuantConfidenceText.textContent = "Top-1 probability: running";
  softwareQuantTopMatchText.textContent = "Top scores: calculating";
  fpgaPredictionValue.textContent = "...";
  fpgaConfidenceText.textContent = "Top-1 probability: running";
  fpgaTopMatchText.textContent = "Top scores: calculating";
  softwareExactTime.textContent = "running";
  softwareQuantTime.textContent = "running";
  fpgaTime.textContent = "running";
  fpgaCoreTime.textContent = "running";
  speedupValue.textContent = "--";
  softwareExactBar.style.height = "0%";
  softwareQuantBar.style.height = "0%";
  fpgaBar.style.height = "0%";

  try {
    const response = await fetch("/predict", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ image: getCanvasPayload() }),
    });

    const data = await response.json();
    if (!response.ok) {
      throw new Error(data.error || "Prediction failed.");
    }

    renderResultCard("software", data.software_result);
    renderResultCard("software_quantized", data.software_quantized_result);
    renderResultCard("fpga", data.fpga_result);
    renderPreview(data.preview_image);
    renderDebugView(data.fpga_result?.debug);
    renderTiming(data.timing);
  } catch (error) {
    softwarePredictionValue.textContent = "!";
    softwareConfidenceText.textContent = error.message;
    softwareTopMatchText.textContent = "Top scores: unavailable";
    softwareResultLabel.textContent = "Software Result From Error";
    softwareQuantPredictionValue.textContent = "!";
    softwareQuantConfidenceText.textContent = error.message;
    softwareQuantTopMatchText.textContent = "Top scores: unavailable";
    softwareQuantResultLabel.textContent = "Quantized Software Reference From Error";
    fpgaPredictionValue.textContent = "!";
    fpgaConfidenceText.textContent = error.message;
    fpgaTopMatchText.textContent = "Top scores: unavailable";
    fpgaResultLabel.textContent = "FPGA Result From Error";
    softwareExactTime.textContent = "--";
    softwareQuantTime.textContent = "--";
    fpgaTime.textContent = "--";
    fpgaCoreTime.textContent = "--";
    speedupValue.textContent = "--";
    clearPreview();
    clearDebugView("FPGA debug data unavailable.");
  }
}
```

```
367
368 async function saveSample() {
369   const label = sampleLabelSelect.value;
370   if (!label) {
371     saveStatus.textContent = "Choose a label first";
372     saveStatus.style.color = "#fbbf24";
373     return;
374   }
375
376   saveSampleBtn.disabled = true;
377   saveStatus.textContent = "Saving...";
378   saveStatus.style.color = "";
379
380   try {
381     const response = await fetch("/save_sample", {
382       method: "POST",
383       headers: { "Content-Type": "application/json" },
384       body: JSON.stringify({
385         image: getCanvasPayload(),
386         label,
387       }),
388     });
389
390     const data = await response.json();
391     if (!response.ok) {
392       throw new Error(data.error || "Save failed.");
393     }
394
395     saveStatus.textContent = `Saved label ${data.label}`;
396     saveStatus.style.color = "#7dd3fc";
397   } catch (error) {
398     saveStatus.textContent = error.message;
399     saveStatus.style.color = "#f87171";
400   } finally {
401     saveSampleBtn.disabled = false;
402   }
403 }
404
405 function drawUploadedImage(file) {
406   if (!file) {
407     return;
408   }
409
410   const reader = new FileReader();
411   reader.onload = () => {
412     const uploadedImage = new Image();
413     uploadedImage.onload = () => {
414       const tempCanvas = document.createElement("canvas");
415       tempCanvas.width = drawCanvas.width;
416       tempCanvas.height = drawCanvas.height;
417       const tempCtx = tempCanvas.getContext("2d");
418
419       tempCtx.fillStyle = "#000000";
420       tempCtx.fillRect(0, 0, tempCanvas.width, tempCanvas.height);
421
422       const scale = Math.min(
423         tempCanvas.width / uploadedImage.width,
424         tempCanvas.height / uploadedImage.height,
```

```
      );
      const width = uploadedImage.width * scale;
      const height = uploadedImage.height * scale;
      const x = (tempCanvas.width - width) / 2;
      const y = (tempCanvas.height - height) / 2;

      tempCtx.imageSmoothingEnabled = true;
      tempCtx.imageSmoothingQuality = "high";
      tempCtx.drawImage(uploadedImage, x, y, width, height);

      const imageData = tempCtx.getImageData(0, 0, tempCanvas.width, tempCanvas.height);
      const data = imageData.data;
      for (let index = 0; index < data.length; index += 4) {
        const gray = Math.round((data[index] + data[index + 1] + data[index + 2]) / 3);
        data[index] = gray;
        data[index + 1] = gray;
        data[index + 2] = gray;
        data[index + 3] = 255;
      }

      drawCtx.putImageData(imageData, 0, 0);
      applyBrushSize();
      clearPreview();
      resetResultCards();
      resetSaveStatus();
    };
    uploadedImage.src = reader.result;
  };
  reader.readAsDataURL(file);
}

drawCanvas.addEventListener("mousedown", startDrawing);
drawCanvas.addEventListener("mousemove", draw);
window.addEventListener("mouseup", stopDrawing);
drawCanvas.addEventListener("mouseleave", stopDrawing);

drawCanvas.addEventListener("touchstart", startDrawing, { passive: false });
drawCanvas.addEventListener("touchmove", draw, { passive: false });
window.addEventListener("touchend", stopDrawing);

brushSizeSlider.addEventListener("input", applyBrushSize);
invertToggle.addEventListener("change", applyInvertStatus);
predictBtn.addEventListener("click", predictDigit);
saveSampleBtn.addEventListener("click", saveSample);
uploadImageInput.addEventListener("change", (event) => {
  drawUploadedImage(event.target.files?.[0]);
  event.target.value = "";
});
clearBtn.addEventListener("click", () => {
  resetCanvas();
  clearPreview();
  resetResultCards();
  resetSaveStatus();
});

resetCanvas();
applyInvertStatus();
applyBrushSize();
```

```
clearPreview();
clearDebugView();
resetSaveStatus();
```

Listing 6: Web interface CSS styling.

```
:root {
  --bg: #080c12;
  --panel: #101722;
  --panel-strong: #141f2d;
  --ink: #e6edf7;
  --muted: #8fa1b7;
  --subtle: #627184;
  --accent: #38bdf8;
  --accent-strong: #7dd3fc;
  --success: #22c55e;
  --line: #243244;
  --line-strong: #33465f;
  --black: #020408;
  --shadow: 0 22px 50px rgba(0, 0, 0, 0.34);
}

* {
  box-sizing: border-box;
}

body {
  margin: 0;
  min-height: 100vh;
  font-family: "Segoe UI", "PingFang SC", "Microsoft YaHei", sans-serif;
  color: var(--ink);
  background: var(--bg);
}

.page {
  width: min(1440px, calc(100% - 32px));
  margin: 0 auto;
  padding: 24px 0 32px;
}

.topbar {
  min-height: 88px;
  display: flex;
  align-items: center;
  justify-content: space-between;
  gap: 18px;
  border: 1px solid var(--line);
  border-radius: 8px;
  padding: 18px 20px;
  background: #0d131d;
  box-shadow: var(--shadow);
}

.eyebrow,
.result-label {
  margin: 0 0 7px;
  color: var(--muted);
  font-size: 0.78rem;
```

```
  font-weight: 800;
  letter-spacing: 0.08em;
  text-transform: uppercase;
}

h1,
h2,
h3,
p {
  margin-top: 0;
}

h1 {
  margin-bottom: 0;
  font-size: clamp(2rem, 3vw, 3rem);
  line-height: 1;
  letter-spacing: 0;
}

h2 {
  margin: 0;
  font-size: 1.2rem;
  line-height: 1.2;
  letter-spacing: 0;
}

.status-strip {
  display: flex;
  flex-wrap: wrap;
  justify-content: flex-end;
  gap: 8px;
}

.status-strip span {
  border: 1px solid var(--line-strong);
  border-radius: 999px;
  padding: 8px 11px;
  background: #0a1018;
  color: #bed0e4;
  font-size: 0.86rem;
  font-weight: 700;
}

.console-grid {
  margin-top: 18px;
  display: grid;
  grid-template-columns: minmax(420px, 0.86fr) minmax(520px, 1.14fr);
  gap: 18px;
  align-items: stretch;
}

.analysis-grid {
  margin-top: 18px;
  display: grid;
  grid-template-columns: 230px minmax(420px, 1fr) 300px;
  gap: 18px;
  align-items: stretch;
}
```

```
111
112 .panel {
113   border: 1px solid var(--line);
114   border-radius: 8px;
115   background: var(--panel);
116   box-shadow: var(--shadow);
117 }
118
119 .panel-heading {
120   min-height: 64px;
121   padding: 16px 18px 12px;
122   border-bottom: 1px solid var(--line);
123   background: var(--panel-strong);
124   border-radius: 8px 8px 0 0;
125 }
126
127 .input-panel {
128   display: grid;
129   justify-items: center;
130   align-content: start;
131   padding-bottom: 18px;
132 }
133
134 .input-panel .panel-heading,
135 .results-panel .panel-heading,
136 .debug-block .panel-heading,
137 .preview-block .panel-heading,
138 .performance-panel .panel-heading {
139   width: 100%;
140 }
141
142 .canvas-shell {
143   width: min(360px, calc(100% - 36px));
144   aspect-ratio: 1;
145   margin: 18px;
146   background: var(--black);
147   border: 1px solid #2d3c50;
148   border-radius: 8px;
149   overflow: hidden;
150   box-shadow:
151     inset 0 0 0 1px rgba(255, 255, 255, 0.05),
152     0 18px 30px rgba(0, 0, 0, 0.35);
153 }
154
155 #draw-canvas {
156   width: 100%;
157   height: 100%;
158   display: block;
159 }
160
161 #preview-canvas {
162   display: block;
163   image-rendering: pixelated;
164   background: var(--black);
165   border-radius: 6px;
166 }
167
168 .control-row {
```

```
  width: min(360px, calc(100% - 36px));
  display: grid;
  gap: 10px;
  color: #c5d5e7;
  font-weight: 700;
}

.brush-control {
  grid-template-columns: auto 1fr auto;
}

.invert-control {
  margin-top: 10px;
  grid-template-columns: auto auto 1fr;
}

.brush-control label,
.brush-control span {
  white-space: nowrap;
}

.brush-control input[type="range"] {
  width: 100%;
  accent-color: var(--accent);
}

.switch {
  width: 50px;
  height: 28px;
  display: inline-grid;
  align-items: center;
  cursor: pointer;
}

.switch input {
  position: absolute;
  opacity: 0;
  pointer-events: none;
}

.switch-track {
  position: relative;
  width: 50px;
  height: 28px;
  border: 1px solid var(--line-strong);
  border-radius: 999px;
  background: #0a1018;
  transition: background 0.16s ease, border-color 0.16s ease;
}

.switch-track::after {
  content: "";
  position: absolute;
  top: 4px;
  left: 4px;
  width: 18px;
  height: 18px;
  border-radius: 50%;
```

```
  background: #8fa1b7;
  transition: transform 0.16s ease, background 0.16s ease;
}

.switch input:checked + .switch-track {
  border-color: var(--accent);
  background: #123247;
}

.switch input:checked + .switch-track::after {
  transform: translateX(22px);
  background: var(--accent);
}

#invert-status {
  color: var(--muted);
}

.actions {
  width: min(360px, calc(100% - 36px));
  margin: 16px 0 12px;
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 10px;
}

.label-save-row {
  width: min(360px, calc(100% - 36px));
  display: grid;
  grid-template-columns: auto minmax(0, 1fr) 72px;
  align-items: center;
  gap: 8px;
  color: #d8e6f5;
  font-size: 0.9rem;
  font-weight: 800;
}

.label-save-row select,
#sample-label-select {
  appearance: none;
  -webkit-appearance: none;
  min-width: 0;
  width: 100%;
  height: 34px;
  border: 1px solid var(--line-strong);
  border-radius: 6px;
  padding: 0 34px 0 12px;
  background:
    linear-gradient(45deg, transparent 50%, #cfe0f1 50%) right 13px center / 6px 6px no-repeat,
    linear-gradient(135deg, #cfe0f1 50%, transparent 50%) right 8px center / 6px 6px no-repeat,
    #0a1018;
  color: var(--ink);
  font: inherit;
}

.label-save-row select option,
#sample-label-select option {
  background: #0a1018;
```

```
  color: var(--ink);
}

.label-save-row button {
  width: 72px;
  min-height: 34px;
  padding: 7px 12px;
  font-size: 0.86rem;
}

.save-status {
  width: min(360px, calc(100% - 36px));
  margin: 8px 0 0;
  color: var(--muted);
  font-size: 0.86rem;
  font-weight: 700;
}

button,
.upload-btn {
  min-height: 42px;
  border: 1px solid transparent;
  border-radius: 6px;
  padding: 10px 12px;
  font-size: 0.95rem;
  font-weight: 800;
  cursor: pointer;
  transition: transform 0.14s ease, border-color 0.14s ease, background 0.14s ease;
}

button:hover,
.upload-btn:hover {
  transform: translateY(-1px);
}

button:disabled {
  cursor: wait;
  opacity: 0.68;
  transform: none;
}

.primary-btn {
  background: var(--accent);
  color: #041019;
  box-shadow: 0 10px 24px rgba(56, 189, 248, 0.2);
}

.primary-btn:hover {
  background: var(--accent-strong);
}

.ghost-btn {
  background: #172232;
  color: #d8e6f5;
  border-color: #2f425b;
}

.ghost-btn:hover,
```

```
.upload-btn:hover {
  border-color: var(--accent);
}

.upload-btn {
  display: grid;
  place-items: center;
  background: #172232;
  color: #d8e6f5;
  border-color: #2f425b;
  text-align: center;
}

#upload-image-input {
  display: none;
}

.results-panel {
  display: flex;
  flex-direction: column;
}

.result-stack {
  padding: 18px;
  display: grid;
  gap: 12px;
}

.result-block {
  border: 1px solid var(--line);
  border-radius: 8px;
  padding: 16px;
  background: #0c131d;
}

.result-block:first-child {
  border-color: rgba(56, 189, 248, 0.55);
  background: #0d1722;
}

.prediction-row {
  display: grid;
  grid-template-columns: 88px 1fr;
  align-items: center;
  gap: 15px;
}

.prediction-value {
  width: 88px;
  height: 88px;
  border-radius: 8px;
  display: grid;
  place-items: center;
  background: #132337;
  border: 1px solid #28506c;
  color: var(--accent-strong);
  font-size: 3rem;
  font-weight: 900;
```

```
  line-height: 1;
}

.result-block:first-child .prediction-value {
  background: var(--accent);
  color: #041019;
}

.compact-result .prediction-value {
  width: 68px;
  height: 68px;
  font-size: 2.2rem;
}

.compact-result .prediction-row {
  grid-template-columns: 68px 1fr;
}

.prediction-meta {
  color: #c6d5e7;
  line-height: 1.6;
  overflow-wrap: anywhere;
}

.prediction-meta p:last-child {
  margin-bottom: 0;
}

.preview-block,
.debug-block,
.performance-panel {
  overflow: hidden;
}

.preview-frame {
  padding: 18px;
  display: grid;
  place-items: center;
}

.debug-block {
  max-height: 520px;
  overflow: auto;
}

.debug-summary {
  margin: 14px 18px;
  color: #c2d1e2;
  line-height: 1.55;
}

.debug-grid {
  padding: 0 18px 18px;
  display: grid;
  gap: 12px;
}

.debug-card {
```

```
  border: 1px solid var(--line);
  border-radius: 8px;
  padding: 12px;
  background: #0b111a;
}

.debug-card-title {
  margin: 0 0 10px;
  color: #b9c8da;
  font-size: 0.9rem;
  font-weight: 800;
  text-transform: uppercase;
  letter-spacing: 0.05em;
}

.debug-card-stats {
  margin: 10px 0 0;
  color: #c3d0df;
  font-size: 0.92rem;
  line-height: 1.5;
}

.debug-card-data {
  margin: 0;
  padding: 12px;
  max-height: 280px;
  overflow: auto;
  border-radius: 6px;
  background: #05080d;
  color: #e5eef8;
  font-size: 0.84rem;
  line-height: 1.5;
  font-family: Consolas, "Courier New", monospace;
  white-space: pre;
}

.metric-grid {
  padding: 18px;
  display: grid;
  gap: 10px;
}

.metric-grid div {
  display: grid;
  grid-template-columns: 92px 1fr;
  align-items: center;
  gap: 10px;
  border-bottom: 1px solid var(--line);
  padding-bottom: 10px;
}

.metric-grid div:last-child {
  border-bottom: 0;
  padding-bottom: 0;
}

.metric-grid span {
  color: var(--subtle);
```

```
  font-size: 0.9rem;
  font-weight: 800;
}

.metric-grid strong {
  color: var(--ink);
  font-size: 0.98rem;
  overflow-wrap: anywhere;
}

.speed-chart {
  margin: 0 18px 18px;
  min-height: 150px;
  border-top: 1px solid var(--line);
  padding-top: 18px;
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 14px;
  align-items: end;
}

.speed-bar-group {
  min-width: 0;
  display: grid;
  grid-template-rows: 112px auto;
  gap: 9px;
  justify-items: center;
  color: var(--muted);
  font-size: 0.82rem;
  font-weight: 800;
}

.speed-bar-track {
  width: 100%;
  max-width: 54px;
  height: 112px;
  display: flex;
  align-items: end;
  border: 1px solid var(--line);
  border-radius: 6px;
  background: #0a1018;
  overflow: hidden;
}

.speed-bar {
  width: 100%;
  height: 0%;
  min-height: 0;
  transition: height 0.28s ease;
}

.cpu-exact-bar {
  background: #64748b;
}

.cpu-quant-bar {
  background: #94a3b8;
}
```

```
575
576 .fpga-bar {
577   background: var(--accent);
578 }
579
580 @media (max-width: 1100px) {
581   .console-grid,
582   .analysis-grid {
583     grid-template-columns: 1fr;
584   }
585
586   .debug-block {
587     max-height: none;
588   }
589 }
590
591 @media (max-width: 720px) {
592   .page {
593     width: min(100% - 20px, 1440px);
594     padding-top: 10px;
595   }
596
597   .topbar {
598     align-items: flex-start;
599     flex-direction: column;
600   }
601
602   .status-strip {
603     justify-content: flex-start;
604   }
605
606   .actions {
607     grid-template-columns: 1fr 1fr;
608   }
609
610   .prediction-row,
611   .compact-result .prediction-row {
612     grid-template-columns: 64px 1fr;
613   }
614
615   .prediction-value,
616   .compact-result .prediction-value {
617     width: 64px;
618     height: 64px;
619     font-size: 2rem;
620   }
621 }
```

Listing 7: Python package requirements for the web demo.

```
Flask>=3.0.0
Pillow>=10.0.0
```

A.2 HPS Software Code

Listing 8: HPS FPGA register constants shared with the C server.

```
#ifndef HPS_FPGA_REGS_H
#define HPS_FPGA_REGS_H

#define LW_BRIDGE_BASE            0xFF200000u
#define HPS_FPGA_MAP_SIZE         0x1030u

#define REG_CONTROL_STATUS_WORD   0u
#define REG_RESULT_WORD           1u
#define REG_LOGIT_BASE_WORD       2u
#define REG_PIXEL_BASE_WORD       12u

#define REG_CONTROL_STATUS_OFFSET (REG_CONTROL_STATUS_WORD * 4u)
#define REG_RESULT_OFFSET         (REG_RESULT_WORD * 4u)
#define REG_LOGIT_BASE_OFFSET     (REG_LOGIT_BASE_WORD * 4u)
#define REG_PIXEL_BASE_OFFSET     (REG_PIXEL_BASE_WORD * 4u)

/* CONTROL write bits */
#define CTRL_START                (1u << 0)
#define CTRL_CLEAR_DONE           (1u << 1)
#define CTRL_CLEAR_INPUT_LOADED   (1u << 2)

/* STATUS read bits */
#define STATUS_REGS_VALID         (1u << 0)
#define STATUS_BUSY               (1u << 1)
#define STATUS_DONE               (1u << 2)
#define STATUS_INPUT_LOADED       (1u << 3)

#define FPGA_INPUT_PIXELS         1024u

#endif
```

Listing 9: HPS inference server for TCP receive, MMIO register access, polling, and result response.

```
#include <arpa/inet.h>
#include <errno.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <signal.h>
#include <stdint.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

```

```
16 #include "hps_fpga_regs.h"
17
18 #define SERVER_PORT 9090
19 #define MAX_REQUEST_BYTES 16384
20 #define EXPECTED_PIXELS FPGA_INPUT_PIXELS
21 #define FPGA_POLL_LIMIT 30000
22 #define FPGA_POLL_DELAY_US 1000
23 #define FPGA_LOGIT_POLL_LIMIT 1000
24 #define FPGA_LOGIT_COUNT 10
25 #define FPGA_CLOCK_HZ 50000000.0
26 #define MAX_RESPONSE_BYTES 65536
27 #define INVALID_LW_BRIDGE_WORD 0xADADADADu
28
29 enum fpga_infer_result {
30   FPGA_INFER_OK = 0,
31   FPGA_INFER_DEVMEM_ERROR = -1,
32   FPGA_INFER_TIMEOUT = 1,
33   FPGA_INFER_INVALID_LW_READ = 2
34 };
35
36 static volatile sig_atomic_t keep_running = 1;
37
38 static void handle_sigint(int sig)
39 {
40   (void)sig;
41   keep_running = 0;
42 }
43
44 static int request_contains_infer_command(const char *text)
45 {
46   return strstr(text, "\"command\"") && strstr(text, "\"infer\"");
47 }
48
49 static int extract_request_id(const char *text, char *out, size_t out_size)
50 {
51   const char *tag = strstr(text, "\"request_id\"");
52   const char *colon;
53   const char *first_quote;
54   const char *second_quote;
55   size_t len;
56
57   if (!tag) {
58     out[0] = '\0';
59     return 0;
60   }
61
62   colon = strchr(tag, ':');
63   if (!colon) {
64     out[0] = '\0';
65     return 0;
66   }
67
68   first_quote = strchr(colon, '"');
69   if (!first_quote) {
70     out[0] = '\0';
71     return 0;
72   }
73   ++first_quote;
```

```
  second_quote = strchr(first_quote, '"');
  if (!second_quote) {
    out[0] = '\0';
    return 0;
  }

  len = (size_t)(second_quote - first_quote);
  if (len >= out_size) {
    len = out_size - 1;
  }

  memcpy(out, first_quote, len);
  out[len] = '\0';
  return 1;
}

static int parse_pixel_values(const char *text, uint8_t *pixels, int max_pixels)
{
  int count = 0;
  const char *pixels_tag = strstr(text, "\"pixels\"");
  const char *p;

  if (!pixels_tag) {
    return -1;
  }

  p = strchr(pixels_tag, '[');
  if (!p) {
    return -1;
  }
  ++p;

  while (*p && *p != ']') {
    char *endptr;
    long value;

    while (*p == ' ' || *p == '\n' || *p == '\r' || *p == '\t' || *p == ',') {
      ++p;
    }
    if (*p == ']') {
      break;
    }

    errno = 0;
    value = strtol(p, &endptr, 10);
    if (endptr == p || errno != 0 || value < 0 || value > 255 || count >= max_pixels) {
      return -1;
    }

    pixels[count] = (uint8_t)value;
    ++count;
    p = endptr;
  }

  return count;
}
```

```
132 static void write_response(int client_fd, const char *response)
133 {
134   size_t remaining = strlen(response);
135   const char *p = response;
136 
137   while (remaining > 0) {
138     ssize_t written = send(client_fd, p, remaining, 0);
139     if (written <= 0) {
140       return;
141     }
142     p += written;
143     remaining -= (size_t)written;
144   }
145 }
146 
147 static int open_fpga_regs(volatile uint32_t **regs_out, void **base_out, int *fd_out)
148 {
149   int fd;
150   void *base;
151 
152   fd = open("/dev/mem", O_RDWR | O_SYNC);
153   if (fd < 0) {
154     return -1;
155   }
156 
157   base = mmap(NULL, HPS_FPGA_MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, LW_BRIDGE_BASE);
158   if (base == MAP_FAILED) {
159     close(fd);
160     return -1;
161   }
162 
163   *regs_out = (volatile uint32_t *)base;
164   *base_out = base;
165   *fd_out = fd;
166   return 0;
167 }
168 
169 static void close_fpga_regs(void *base, int fd)
170 {
171   if (base && base != MAP_FAILED) {
172     munmap(base, HPS_FPGA_MAP_SIZE);
173   }
174   if (fd >= 0) {
175     close(fd);
176   }
177 }
178 
179 static void read_fpga_outputs(
180   volatile uint32_t *regs,
181   int *predicted_digit,
182   int32_t *logits,
183   uint32_t *cycle_count
184 )
185 {
186   int index;
187   uint32_t result_word;
188 
189   result_word = regs[REG_RESULT_WORD];
```

```
190   *predicted_digit = (int)(result_word & 0xfu);
191   *cycle_count = result_word >> 4;
192   for (index = 0; index < FPGA_LOGIT_COUNT; ++index) {
193     logits[index] = (int32_t)regs[REG_LOGIT_BASE_WORD + index];
194   }
195 }
196
197 static int outputs_contain_invalid_lw_read(int predicted_digit, const int32_t *logits)
198 {
199   int index;
200
201   if (predicted_digit == (int)(INVALID_LW_BRIDGE_WORD & 0xfu)) {
202     for (index = 0; index < FPGA_LOGIT_COUNT; ++index) {
203       if ((uint32_t)logits[index] != INVALID_LW_BRIDGE_WORD) {
204         return 0;
205       }
206     }
207     return 1;
208   }
209
210   return 0;
211 }
212
213 static int run_fpga_inference_with_logits(
214   const uint8_t *pixels,
215   uint32_t *status_before,
216   uint32_t *status_after,
217   int *predicted_digit,
218   int32_t *logits,
219   uint32_t *cycle_count,
220   int *debug_valid
221 )
222 {
223   volatile uint32_t *regs;
224   void *base = NULL;
225   int fd = -1;
226   int index;
227   int poll_count;
228
229   *debug_valid = 0;
230
231   if (open_fpga_regs(&regs, &base, &fd) != 0) {
232     return FPGA_INFER_DEVMEM_ERROR;
233   }
234
235   *status_before = regs[REG_CONTROL_STATUS_WORD];
236
237   regs[REG_CONTROL_STATUS_WORD] = CTRL_CLEAR_DONE | CTRL_CLEAR_INPUT_LOADED;
238
239   for (index = 0; index < EXPECTED_PIXELS; ++index) {
240     regs[REG_PIXEL_BASE_WORD + index] = pixels[index];
241   }
242
243   regs[REG_CONTROL_STATUS_WORD] = CTRL_START | CTRL_CLEAR_DONE;
244
245   for (poll_count = 0; poll_count < FPGA_POLL_LIMIT; ++poll_count) {
246     *status_after = regs[REG_CONTROL_STATUS_WORD];
247     if ((*status_after & STATUS_DONE) != 0u) {
```

```
      for (index = 0; index < FPGA_LOGIT_POLL_LIMIT; ++index) {
        *status_after = regs[REG_CONTROL_STATUS_WORD];
        if ((*status_after & STATUS_REGS_VALID) != 0u) {
          *debug_valid = 1;
          break;
        }
        usleep(FPGA_POLL_DELAY_US);
      }
      if (*debug_valid) {
        read_fpga_outputs(regs, predicted_digit, logits, cycle_count);
        if (outputs_contain_invalid_lw_read(*predicted_digit, logits)) {
          close_fpga_regs(base, fd);
          return FPGA_INFER_INVALID_LW_READ;
        }
      }
      close_fpga_regs(base, fd);
      return FPGA_INFER_OK;
    }
    usleep(FPGA_POLL_DELAY_US);
  }

  close_fpga_regs(base, fd);
  return FPGA_INFER_TIMEOUT;
}

static int append_text(char *buffer, size_t buffer_size, size_t *used, const char *text)
{
  int written = snprintf(buffer + *used, buffer_size - *used, "%s", text);
  if (written < 0 || (size_t)written >= buffer_size - *used) {
    return -1;
  }
  *used += (size_t)written;
  return 0;
}

static int append_format(char *buffer, size_t buffer_size, size_t *used, const char *fmt, ...)
{
  va_list args;
  int written;

  va_start(args, fmt);
  written = vsnprintf(buffer + *used, buffer_size - *used, fmt, args);
  va_end(args);

  if (written < 0 || (size_t)written >= buffer_size - *used) {
    return -1;
  }
  *used += (size_t)written;
  return 0;
}

static int build_success_response(
  char *response,
  size_t response_size,
  const char *request_id,
  uint32_t status_before,
  uint32_t status_after,
  int predicted_digit,
```

```
306   const int32_t *logits,
307   uint32_t cycle_count,
308   int debug_valid
309 )
310 {
311   size_t used = 0;
312   int index;
313   double fpga_compute_ms = ((double)cycle_count * 1000.0) / FPGA_CLOCK_HZ;
314
315   if (append_format(
316         response,
317         response_size,
318         &used,
319         "{\"ok\":true,\"request_id\":\"%s\",\"source\":\"fpga\",\"prediction\":%d,"
320         "\"confidence\":0.0,\"status_before\":%u,\"status_after\":%u,"
321         "\"note\":\"%s\",\"debug\":{\"kind\":\"final_logits\",\"logits_valid\":%s,"
322         "\"fpga_cycles\":%u,\"fpga_clock_hz\":50000000,\"fpga_compute_ms\":%.6f",
323         request_id,
324         predicted_digit,
325         status_before,
326         status_after,
327         debug_valid ? "hardware_run_completed_logits_available" : "
      hardware_run_completed_but_logits_not_valid",
328         debug_valid ? "true" : "false",
329         cycle_count,
330         fpga_compute_ms
331       ) != 0) {
332     return -1;
333   }
334
335   if (!debug_valid) {
336     if (append_text(response, response_size, &used, "}}\n") != 0) {
337       return -1;
338     }
339     return 0;
340   }
341
342   if (append_text(response, response_size, &used, ",\"logits\":[") != 0) {
343     return -1;
344   }
345
346   for (index = 0; index < FPGA_LOGIT_COUNT; ++index) {
347     if (append_format(
348           response,
349           response_size,
350           &used,
351           index == 0 ? "%d" : ",%d",
352           logits[index]
353         ) != 0) {
354       return -1;
355     }
356   }
357
358   if (append_text(response, response_size, &used, "]}}\n") != 0) {
359     return -1;
360   }
361
362   return 0;
```

```
}

static void handle_client(int client_fd)
{
  char buffer[MAX_REQUEST_BYTES + 1];
  char response[MAX_RESPONSE_BYTES];
  char request_id[128];
  uint8_t pixels[EXPECTED_PIXELS];
  int32_t logits[FPGA_LOGIT_COUNT];
  ssize_t received;
  int pixel_count;
  uint32_t status_before;
  uint32_t status_after;
  uint32_t cycle_count;
  int fpga_result;
  int debug_valid;
  int predicted_digit;

  received = recv(client_fd, buffer, MAX_REQUEST_BYTES, 0);
  if (received <= 0) {
    return;
  }

  buffer[received] = '\0';
  extract_request_id(buffer, request_id, sizeof(request_id));

  if (!request_contains_infer_command(buffer)) {
    write_response(
      client_fd,
      "{\"ok\":false,\"error\":\"unsupported_command\"}\n"
    );
    return;
  }

  pixel_count = parse_pixel_values(buffer, pixels, EXPECTED_PIXELS);
  if (pixel_count != EXPECTED_PIXELS) {
    snprintf(
      response,
      sizeof(response),
      "{\"ok\":false,\"request_id\":\"%s\",\"error\":\"invalid_pixel_payload\",\"expected_pixels
    \":1024}\n",
      request_id
    );
    write_response(client_fd, response);
    return;
  }

  printf("received infer request id=%s pixels=%d\n", request_id, pixel_count);
  fflush(stdout);

  predicted_digit = -1;
  cycle_count = 0u;
  memset(logits, 0, sizeof(logits));

  fpga_result = run_fpga_inference_with_logits(
    pixels,
    &status_before,
    &status_after,
```

```
    &predicted_digit,
    logits,
    &cycle_count,
    &debug_valid
  );
  if (fpga_result == FPGA_INFER_DEVMEM_ERROR) {
    snprintf(
      response,
      sizeof(response),
      "{\"ok\":false,\"request_id\":\"%s\",\"error\":\"devmem_access_failed\"}\n",
      request_id
    );
    write_response(client_fd, response);
    return;
  }

  if (fpga_result != FPGA_INFER_OK) {
    if (fpga_result == FPGA_INFER_INVALID_LW_READ) {
      snprintf(
        response,
        sizeof(response),
        "{\"ok\":false,\"request_id\":\"%s\",\"error\":\"lw_bridge_invalid_read\",\"detail\":\"
    read_0xADADADAD_from_fpga_regs\",\"status_before\":%u,\"status_after\":%u}\n",
        request_id,
        status_before,
        status_after
      );
      write_response(client_fd, response);
      return;
    }

    snprintf(
      response,
      sizeof(response),
      "{\"ok\":false,\"request_id\":\"%s\",\"error\":\"fpga_timeout\",\"status_before\":%u,\"
    status_after\":%u}\n",
      request_id,
      status_before,
      status_after
    );
    write_response(client_fd, response);
    return;
  }

  if (build_success_response(
        response,
        sizeof(response),
        request_id,
        status_before,
        status_after,
        predicted_digit,
        logits,
        cycle_count,
        debug_valid
  ) != 0) {
    snprintf(
      response,
      sizeof(response),
```

```
      "{\"ok\":false,\"request_id\":\"%s\",\"error\":\"response_buffer_overflow\"}\n",
      request_id
    );
  }

  printf(
    "completed request id=%s prediction=%d status_before=%u status_after=%u debug_valid=%d
    fpga_cycles=%u\n",
    request_id,
    predicted_digit,
    status_before,
    status_after,
    debug_valid,
    cycle_count
  );
  fflush(stdout);
  write_response(client_fd, response);
}

int main(void)
{
  int server_fd;
  int optval = 1;
  struct sockaddr_in addr;

  signal(SIGINT, handle_sigint);
  signal(SIGTERM, handle_sigint);

  server_fd = socket(AF_INET, SOCK_STREAM, 0);
  if (server_fd < 0) {
    perror("socket");
    return 1;
  }

  if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval)) < 0) {
    perror("setsockopt");
    close(server_fd);
    return 1;
  }

  memset(&addr, 0, sizeof(addr));
  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = htonl(INADDR_ANY);
  addr.sin_port = htons(SERVER_PORT);

  if (bind(server_fd, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
    perror("bind");
    close(server_fd);
    return 1;
  }

  if (listen(server_fd, 4) < 0) {
    perror("listen");
    close(server_fd);
    return 1;
  }

  printf("hps_infer_server listening on port %d\n", SERVER_PORT);
```

```
533
534   while (keep_running) {
535     int client_fd = accept(server_fd, NULL, NULL);
536     if (client_fd < 0) {
537       if (errno == EINTR) {
538         continue;
539       }
540       perror("accept");
541       break;
542     }
543
544     handle_client(client_fd);
545     close(client_fd);
546   }
547
548   close(server_fd);
549   return 0;
550 }
```

A.3 FPGA RTL Code

Listing 10: Board-level HPS and FPGA top module.

```
1  // ============================================================
2  // Copyright (c) 2013 by Terasic Technologies Inc.
3  // ============================================================
4  //
5  // Modified 2019 by Stephen A. Edwards
6  // Modified 2026 to expose full LeNet inference registers to HPS LW bridge
7  //
8  module soc_system_top(
9
10  ///////// ADC /////////
11  inout         ADC_CS_N,
12  output        ADC_DIN,
13  input         ADC_DOUT,
14  output        ADC_SCLK,
15
16  ///////// AUD /////////
17  input         AUD_ADCDAT,
18  inout         AUD_ADCLRCK,
19  inout         AUD_BCLK,
20  output        AUD_DACDAT,
21  inout         AUD_DACLRCK,
22  output        AUD_XCK,
23
24  ///////// CLOCK2 /////////
25  input         CLOCK2_50,
26
27  ///////// CLOCK3 /////////
28  input         CLOCK3_50,
29
30  ///////// CLOCK4 /////////
31  input         CLOCK4_50,
32
33  ///////// CLOCK /////////
34  input         CLOCK_50,
```

```
35
36  ///////// DRAM /////////
37  output [12:0] DRAM_ADDR,
38  output [1:0]  DRAM_BA,
39  output        DRAM_CAS_N,
40  output        DRAM_CKE,
41  output        DRAM_CLK,
42  output        DRAM_CS_N,
43  inout [15:0]  DRAM_DQ,
44  output        DRAM_LDQM,
45  output        DRAM_RAS_N,
46  output        DRAM_UDQM,
47  output        DRAM_WE_N,
48
49  ///////// FAN /////////
50  output        FAN_CTRL,
51
52  ///////// FPGA /////////
53  output        FPGA_I2C_SCLK,
54  inout         FPGA_I2C_SDAT,
55
56  ///////// GPIO /////////
57  inout [35:0]  GPIO_0,
58  inout [35:0]  GPIO_1,
59
60  ///////// HEX0 /////////
61  output [6:0]  HEX0,
62
63  ///////// HEX1 /////////
64  output [6:0]  HEX1,
65
66  ///////// HEX2 /////////
67  output [6:0]  HEX2,
68
69  ///////// HEX3 /////////
70  output [6:0]  HEX3,
71
72  ///////// HEX4 /////////
73  output [6:0]  HEX4,
74
75  ///////// HEX5 /////////
76  output [6:0]  HEX5,
77
78  ///////// HPS /////////
79  inout         HPS_CONV_USB_N,
80  output [14:0] HPS_DDR3_ADDR,
81  output [2:0]  HPS_DDR3_BA,
82  output        HPS_DDR3_CAS_N,
83  output        HPS_DDR3_CKE,
84  output        HPS_DDR3_CK_N,
85  output        HPS_DDR3_CK_P,
86  output        HPS_DDR3_CS_N,
87  output [3:0]  HPS_DDR3_DM,
88  inout [31:0]  HPS_DDR3_DQ,
89  inout [3:0]   HPS_DDR3_DQS_N,
90  inout [3:0]   HPS_DDR3_DQS_P,
91  output        HPS_DDR3_ODT,
92  output        HPS_DDR3_RAS_N,
```

```
93  output        HPS_DDR3_RESET_N,
94  input         HPS_DDR3_RZQ,
95  output        HPS_DDR3_WE_N,
96  output        HPS_ENET_GTX_CLK,
97  inout         HPS_ENET_INT_N,
98  output        HPS_ENET_MDC,
99  inout         HPS_ENET_MDIO,
100 input         HPS_ENET_RX_CLK,
101 input [3:0]   HPS_ENET_RX_DATA,
102 input         HPS_ENET_RX_DV,
103 output [3:0]  HPS_ENET_TX_DATA,
104 output        HPS_ENET_TX_EN,
105 inout         HPS_GSENSOR_INT,
106 inout         HPS_I2C1_SCLK,
107 inout         HPS_I2C1_SDAT,
108 inout         HPS_I2C2_SCLK,
109 inout         HPS_I2C2_SDAT,
110 inout         HPS_I2C_CONTROL,
111 inout         HPS_KEY,
112 inout         HPS_LED,
113 inout         HPS_LTC_GPIO,
114 output        HPS_SD_CLK,
115 inout         HPS_SD_CMD,
116 inout [3:0]   HPS_SD_DATA,
117 output        HPS_SPIM_CLK,
118 input         HPS_SPIM_MISO,
119 output        HPS_SPIM_MOSI,
120 inout         HPS_SPIM_SS,
121 input         HPS_UART_RX,
122 output        HPS_UART_TX,
123 input         HPS_USB_CLKOUT,
124 inout [7:0]   HPS_USB_DATA,
125 input         HPS_USB_DIR,
126 input         HPS_USB_NXT,
127 output        HPS_USB_STP,
128
129 ///////// IRDA /////////
130 input         IRDA_RXD,
131 output        IRDA_TXD,
132
133 ///////// KEY /////////
134 input [3:0]   KEY,
135
136 ///////// LEDR /////////
137 output [9:0]  LEDR,
138
139 ///////// PS2 /////////
140 inout         PS2_CLK,
141 inout         PS2_CLK2,
142 inout         PS2_DAT,
143 inout         PS2_DAT2,
144
145 ///////// SW /////////
146 input [9:0]   SW,
147
148 ///////// TD /////////
149 input         TD_CLK27,
150 input [7:0]   TD_DATA,
```

```
151  input         TD_HS,
152  output        TD_RESET_N,
153  input         TD_VS,
154
155
156  ///////// VGA /////////
157  output [7:0]  VGA_B,
158  output        VGA_BLANK_N,
159  output        VGA_CLK,
160  output [7:0]  VGA_G,
161  output        VGA_HS,
162  output [7:0]  VGA_R,
163  output        VGA_SYNC_N,
164  output        VGA_VS
165 );
166
167    localparam integer LOGIT_COUNT = 10;
168    localparam integer PIXELS      = 32 * 32;
169
170    localparam [6:0] HEX_OFF       = 7'b1111111;
171    localparam [35:0] GPIO_HIZ     = {36{1'bZ}};
172    localparam [15:0] DRAM_DQ_HIZ  = {16{1'bZ}};
173
174    // All active logic in this top level runs from the board 50 MHz clock.
175    wire rst_n;
176    wire start_pulse;
177    wire clear_done_pulse;
178    reg  done_latched;
179    reg  accel_busy;
180    wire infer_done;
181    wire result_regs_valid;
182    wire [LOGIT_COUNT*32-1:0] logit_regs_flat;
183    wire pixel_wr_en;
184    wire [$clog2(PIXELS)-1:0] pixel_wr_addr;
185    wire [7:0] pixel_wr_data;
186    wire pixel_rd_en;
187    wire [$clog2(PIXELS)-1:0] pixel_rd_addr;
188    wire [7:0] pixel_rd_data;
189    wire input_loaded;
190    wire h2f_rst_n;
191    wire [3:0] predicted_digit;
192    wire signed [10*32-1:0] logits_flat;
193    wire [27:0] infer_cycle_count;
194    reg  [27:0] infer_cycle_count_latched;
195
196    // HPS lightweight AXI master connected to the LeNet register block.
197    wire [11:0] h2f_lw_awid;
198    wire [20:0] h2f_lw_awaddr;
199    wire [3:0]  h2f_lw_awlen;
200    wire [2:0]  h2f_lw_awsize;
201    wire [1:0]  h2f_lw_awburst;
202    wire [1:0]  h2f_lw_awlock;
203    wire [3:0]  h2f_lw_awcache;
204    wire [2:0]  h2f_lw_awprot;
205    wire        h2f_lw_awvalid;
206    wire        h2f_lw_awready;
207    wire [11:0] h2f_lw_wid;
208    wire [31:0] h2f_lw_wdata;
```

```
209    wire [3:0]  h2f_lw_wstrb;
210    wire        h2f_lw_wlast;
211    wire        h2f_lw_wvalid;
212    wire        h2f_lw_wready;
213    wire [11:0] h2f_lw_bid;
214    wire [1:0]  h2f_lw_bresp;
215    wire        h2f_lw_bvalid;
216    wire        h2f_lw_bready;
217    wire [11:0] h2f_lw_arid;
218    wire [20:0] h2f_lw_araddr;
219    wire [3:0]  h2f_lw_arlen;
220    wire [2:0]  h2f_lw_arsize;
221    wire [1:0]  h2f_lw_arburst;
222    wire [1:0]  h2f_lw_arlock;
223    wire [3:0]  h2f_lw_arcache;
224    wire [2:0]  h2f_lw_arprot;
225    wire        h2f_lw_arvalid;
226    wire        h2f_lw_arready;
227    wire [11:0] h2f_lw_rid;
228    wire [31:0] h2f_lw_rdata;
229    wire [1:0]  h2f_lw_rresp;
230    wire        h2f_lw_rlast;
231    wire        h2f_lw_rvalid;
232    wire        h2f_lw_rready;
233
234    function automatic [6:0] hex_digit_to_7seg(input [3:0] value);
235       begin
236          case (value)
237             4'h0:    hex_digit_to_7seg = 7'b1000000;
238             4'h1:    hex_digit_to_7seg = 7'b1111001;
239             4'h2:    hex_digit_to_7seg = 7'b0100100;
240             4'h3:    hex_digit_to_7seg = 7'b0110000;
241             4'h4:    hex_digit_to_7seg = 7'b0011001;
242             4'h5:    hex_digit_to_7seg = 7'b0010010;
243             4'h6:    hex_digit_to_7seg = 7'b0000010;
244             4'h7:    hex_digit_to_7seg = 7'b1111000;
245             4'h8:    hex_digit_to_7seg = 7'b0000000;
246             4'h9:    hex_digit_to_7seg = 7'b0010000;
247             4'ha:    hex_digit_to_7seg = 7'b0001000;
248             4'hb:    hex_digit_to_7seg = 7'b0000011;
249             4'hc:    hex_digit_to_7seg = 7'b1000110;
250             4'hd:    hex_digit_to_7seg = 7'b0100001;
251             4'he:    hex_digit_to_7seg = 7'b0000110;
252             default: hex_digit_to_7seg = 7'b0001110;
253          endcase
254       end
255    endfunction
256
257    assign rst_n = KEY[0] & h2f_rst_n;
258
259    always @(posedge CLOCK_50 or negedge rst_n) begin
260       if (!rst_n) begin
261          done_latched              <= 1'b0;
262          accel_busy                <= 1'b0;
263          infer_cycle_count_latched <= 28'd0;
264       end else begin
265          if (start_pulse || clear_done_pulse) begin
266             done_latched <= 1'b0;
```

```
267         end else if (infer_done) begin
268            done_latched              <= 1'b1;
269            infer_cycle_count_latched <= infer_cycle_count;
270         end
271
272         if (start_pulse) begin
273            accel_busy                <= 1'b1;
274            infer_cycle_count_latched <= 28'd0;
275         end else if (infer_done) begin
276            accel_busy <= 1'b0;
277         end
278      end
279   end
280
281   // FPGA inference core.
282   lenet_int8_top dut (
283     .clk(CLOCK_50),
284     .rst_n(rst_n),
285     .start(start_pulse),
286     .pixel_wr_en(pixel_wr_en),
287     .pixel_wr_addr(pixel_wr_addr),
288     .pixel_wr_data(pixel_wr_data),
289     .pixel_dbg_rd_en(pixel_rd_en),
290     .pixel_dbg_rd_addr(pixel_rd_addr),
291     .pixel_dbg_rd_data(pixel_rd_data),
292     .busy(),
293     .done(infer_done),
294     .predicted_digit(predicted_digit),
295     .logits_flat(logits_flat),
296     .cycle_count(infer_cycle_count)
297   );
298
299   genvar logit_idx;
300   generate
301     for (logit_idx = 0; logit_idx < LOGIT_COUNT; logit_idx = logit_idx + 1) begin : g_logit_regs
302       assign logit_regs_flat[logit_idx * 32 +: 32] = logits_flat[logit_idx * 32 +: 32];
303     end
304   endgenerate
305
306   assign result_regs_valid = done_latched;
307
308   // Memory-mapped register window exposed to HPS at the lightweight bridge.
309   hps_lenet_regs #(
310     .REG_COUNT(LOGIT_COUNT),
311     .PIXELS(PIXELS)
312   ) u_regs (
313     .clk(CLOCK_50),
314     .rst_n(rst_n),
315     .regs_valid(result_regs_valid),
316     .accel_busy(accel_busy),
317     .accel_done(done_latched),
318     .predicted_digit(predicted_digit),
319     .cycle_count(infer_cycle_count_latched),
320     .regs_flat(logit_regs_flat),
321     .start_pulse(start_pulse),
322     .clear_done_pulse(clear_done_pulse),
323     .pixel_wr_en(pixel_wr_en),
324     .pixel_wr_addr(pixel_wr_addr),
```

```
325       .pixel_wr_data(pixel_wr_data),
326       .pixel_rd_en(pixel_rd_en),
327       .pixel_rd_addr(pixel_rd_addr),
328       .pixel_rd_data(pixel_rd_data),
329       .input_loaded(input_loaded),
330       .s_awid(h2f_lw_awid),
331       .s_awaddr(h2f_lw_awaddr),
332       .s_awvalid(h2f_lw_awvalid),
333       .s_awready(h2f_lw_awready),
334       .s_wdata(h2f_lw_wdata),
335       .s_wstrb(h2f_lw_wstrb),
336       .s_wlast(h2f_lw_wlast),
337       .s_wvalid(h2f_lw_wvalid),
338       .s_wready(h2f_lw_wready),
339       .s_bid(h2f_lw_bid),
340       .s_bresp(h2f_lw_bresp),
341       .s_bvalid(h2f_lw_bvalid),
342       .s_bready(h2f_lw_bready),
343       .s_arid(h2f_lw_arid),
344       .s_araddr(h2f_lw_araddr),
345       .s_arvalid(h2f_lw_arvalid),
346       .s_arready(h2f_lw_arready),
347       .s_rid(h2f_lw_rid),
348       .s_rdata(h2f_lw_rdata),
349       .s_rresp(h2f_lw_rresp),
350       .s_rlast(h2f_lw_rlast),
351       .s_rvalid(h2f_lw_rvalid),
352       .s_rready(h2f_lw_rready)
353     );
354
355     // HPS subsystem generated by Platform Designer.
356     soc_system_hps_0 #(
357       .F2S_Width(2),
358       .S2F_Width(2)
359     ) hps_0 (
360       .h2f_rst_n(h2f_rst_n),
361       .f2h_axi_clk(CLOCK_50),
362       .f2h_AWID(8'd0),
363       .f2h_AWADDR(32'd0),
364       .f2h_AWLEN(4'd0),
365       .f2h_AWSIZE(3'd0),
366       .f2h_AWBURST(2'd0),
367       .f2h_AWLOCK(2'd0),
368       .f2h_AWCACHE(4'd0),
369       .f2h_AWPROT(3'd0),
370       .f2h_AWVALID(1'b0),
371       .f2h_AWREADY(),
372       .f2h_AWUSER(5'd0),
373       .f2h_WID(8'd0),
374       .f2h_WDATA(64'd0),
375       .f2h_WSTRB(8'd0),
376       .f2h_WLAST(1'b0),
377       .f2h_WVALID(1'b0),
378       .f2h_WREADY(),
379       .f2h_BID(),
380       .f2h_BRESP(),
381       .f2h_BVALID(),
382       .f2h_BREADY(1'b0),
```

```
383         .f2h_ARID(8'd0),
384         .f2h_ARADDR(32'd0),
385         .f2h_ARLEN(4'd0),
386         .f2h_ARSIZE(3'd0),
387         .f2h_ARBURST(2'd0),
388         .f2h_ARLOCK(2'd0),
389         .f2h_ARCACHE(4'd0),
390         .f2h_ARPROT(3'd0),
391         .f2h_ARVALID(1'b0),
392         .f2h_ARREADY(),
393         .f2h_ARUSER(5'd0),
394         .f2h_RID(),
395         .f2h_RDATA(),
396         .f2h_RRESP(),
397         .f2h_RLAST(),
398         .f2h_RVALID(),
399         .f2h_RREADY(1'b0),
400         .h2f_lw_axi_clk(CLOCK_50),
401         .h2f_lw_AWID(h2f_lw_awid),
402         .h2f_lw_AWADDR(h2f_lw_awaddr),
403         .h2f_lw_AWLEN(h2f_lw_awlen),
404         .h2f_lw_AWSIZE(h2f_lw_awsize),
405         .h2f_lw_AWBURST(h2f_lw_awburst),
406         .h2f_lw_AWLOCK(h2f_lw_awlock),
407         .h2f_lw_AWCACHE(h2f_lw_awcache),
408         .h2f_lw_AWPROT(h2f_lw_awprot),
409         .h2f_lw_AWVALID(h2f_lw_awvalid),
410         .h2f_lw_AWREADY(h2f_lw_awready),
411         .h2f_lw_WID(h2f_lw_wid),
412         .h2f_lw_WDATA(h2f_lw_wdata),
413         .h2f_lw_WSTRB(h2f_lw_wstrb),
414         .h2f_lw_WLAST(h2f_lw_wlast),
415         .h2f_lw_WVALID(h2f_lw_wvalid),
416         .h2f_lw_WREADY(h2f_lw_wready),
417         .h2f_lw_BID(h2f_lw_bid),
418         .h2f_lw_BRESP(h2f_lw_bresp),
419         .h2f_lw_BVALID(h2f_lw_bvalid),
420         .h2f_lw_BREADY(h2f_lw_bready),
421         .h2f_lw_ARID(h2f_lw_arid),
422         .h2f_lw_ARADDR(h2f_lw_araddr),
423         .h2f_lw_ARLEN(h2f_lw_arlen),
424         .h2f_lw_ARSIZE(h2f_lw_arsize),
425         .h2f_lw_ARBURST(h2f_lw_arburst),
426         .h2f_lw_ARLOCK(h2f_lw_arlock),
427         .h2f_lw_ARCACHE(h2f_lw_arcache),
428         .h2f_lw_ARPROT(h2f_lw_arprot),
429         .h2f_lw_ARVALID(h2f_lw_arvalid),
430         .h2f_lw_ARREADY(h2f_lw_arready),
431         .h2f_lw_RID(h2f_lw_rid),
432         .h2f_lw_RDATA(h2f_lw_rdata),
433         .h2f_lw_RRESP(h2f_lw_rresp),
434         .h2f_lw_RLAST(h2f_lw_rlast),
435         .h2f_lw_RVALID(h2f_lw_rvalid),
436         .h2f_lw_RREADY(h2f_lw_rready),
437         .h2f_axi_clk(CLOCK_50),
438         .h2f_AWID(),
439         .h2f_AWADDR(),
440         .h2f_AWLEN(),
```

```
441         .h2f_AWSIZE(),
442         .h2f_AWBURST(),
443         .h2f_AWLOCK(),
444         .h2f_AWCACHE(),
445         .h2f_AWPROT(),
446         .h2f_AWVALID(),
447         .h2f_AWREADY(1'b0),
448         .h2f_WID(),
449         .h2f_WDATA(),
450         .h2f_WSTRB(),
451         .h2f_WLAST(),
452         .h2f_WVALID(),
453         .h2f_WREADY(1'b0),
454         .h2f_BID(12'd0),
455         .h2f_BRESP(2'd0),
456         .h2f_BVALID(1'b0),
457         .h2f_BREADY(),
458         .h2f_ARID(),
459         .h2f_ARADDR(),
460         .h2f_ARLEN(),
461         .h2f_ARSIZE(),
462         .h2f_ARBURST(),
463         .h2f_ARLOCK(),
464         .h2f_ARCACHE(),
465         .h2f_ARPROT(),
466         .h2f_ARVALID(),
467         .h2f_ARREADY(1'b0),
468         .h2f_RID(12'd0),
469         .h2f_RDATA(64'd0),
470         .h2f_RRESP(2'd0),
471         .h2f_RLAST(1'b0),
472         .h2f_RVALID(1'b0),
473         .h2f_RREADY(),
474         .mem_a(HPS_DDR3_ADDR),
475         .mem_ba(HPS_DDR3_BA),
476         .mem_ck(HPS_DDR3_CK_P),
477         .mem_ck_n(HPS_DDR3_CK_N),
478         .mem_cke(HPS_DDR3_CKE),
479         .mem_cs_n(HPS_DDR3_CS_N),
480         .mem_ras_n(HPS_DDR3_RAS_N),
481         .mem_cas_n(HPS_DDR3_CAS_N),
482         .mem_we_n(HPS_DDR3_WE_N),
483         .mem_reset_n(HPS_DDR3_RESET_N),
484         .mem_dq(HPS_DDR3_DQ),
485         .mem_dqs(HPS_DDR3_DQS_P),
486         .mem_dqs_n(HPS_DDR3_DQS_N),
487         .mem_odt(HPS_DDR3_ODT),
488         .mem_dm(HPS_DDR3_DM),
489         .oct_rzqin(HPS_DDR3_RZQ),
490         .hps_io_emac1_inst_TX_CLK(HPS_ENET_GTX_CLK),
491         .hps_io_emac1_inst_TXD0(HPS_ENET_TX_DATA[0]),
492         .hps_io_emac1_inst_TXD1(HPS_ENET_TX_DATA[1]),
493         .hps_io_emac1_inst_TXD2(HPS_ENET_TX_DATA[2]),
494         .hps_io_emac1_inst_TXD3(HPS_ENET_TX_DATA[3]),
495         .hps_io_emac1_inst_RXD0(HPS_ENET_RX_DATA[0]),
496         .hps_io_emac1_inst_MDIO(HPS_ENET_MDIO),
497         .hps_io_emac1_inst_MDC(HPS_ENET_MDC),
498         .hps_io_emac1_inst_RX_CTL(HPS_ENET_RX_DV),
```

```
499       .hps_io_emac1_inst_TX_CTL(HPS_ENET_TX_EN),
500       .hps_io_emac1_inst_RX_CLK(HPS_ENET_RX_CLK),
501       .hps_io_emac1_inst_RXD1(HPS_ENET_RX_DATA[1]),
502       .hps_io_emac1_inst_RXD2(HPS_ENET_RX_DATA[2]),
503       .hps_io_emac1_inst_RXD3(HPS_ENET_RX_DATA[3]),
504       .hps_io_sdio_inst_CMD(HPS_SD_CMD),
505       .hps_io_sdio_inst_D0(HPS_SD_DATA[0]),
506       .hps_io_sdio_inst_D1(HPS_SD_DATA[1]),
507       .hps_io_sdio_inst_CLK(HPS_SD_CLK),
508       .hps_io_sdio_inst_D2(HPS_SD_DATA[2]),
509       .hps_io_sdio_inst_D3(HPS_SD_DATA[3]),
510       .hps_io_usb1_inst_D0(HPS_USB_DATA[0]),
511       .hps_io_usb1_inst_D1(HPS_USB_DATA[1]),
512       .hps_io_usb1_inst_D2(HPS_USB_DATA[2]),
513       .hps_io_usb1_inst_D3(HPS_USB_DATA[3]),
514       .hps_io_usb1_inst_D4(HPS_USB_DATA[4]),
515       .hps_io_usb1_inst_D5(HPS_USB_DATA[5]),
516       .hps_io_usb1_inst_D6(HPS_USB_DATA[6]),
517       .hps_io_usb1_inst_D7(HPS_USB_DATA[7]),
518       .hps_io_usb1_inst_CLK(HPS_USB_CLKOUT),
519       .hps_io_usb1_inst_STP(HPS_USB_STP),
520       .hps_io_usb1_inst_DIR(HPS_USB_DIR),
521       .hps_io_usb1_inst_NXT(HPS_USB_NXT),
522       .hps_io_spim1_inst_CLK(HPS_SPIM_CLK),
523       .hps_io_spim1_inst_MOSI(HPS_SPIM_MOSI),
524       .hps_io_spim1_inst_MISO(HPS_SPIM_MISO),
525       .hps_io_spim1_inst_SS0(HPS_SPIM_SS),
526       .hps_io_uart0_inst_RX(HPS_UART_RX),
527       .hps_io_uart0_inst_TX(HPS_UART_TX),
528       .hps_io_i2c0_inst_SDA(HPS_I2C1_SDAT),
529       .hps_io_i2c0_inst_SCL(HPS_I2C1_SCLK),
530       .hps_io_i2c1_inst_SDA(HPS_I2C2_SDAT),
531       .hps_io_i2c1_inst_SCL(HPS_I2C2_SCLK),
532       .hps_io_gpio_inst_GPIO09(HPS_CONV_USB_N),
533       .hps_io_gpio_inst_GPIO35(HPS_ENET_INT_N),
534       .hps_io_gpio_inst_GPIO40(HPS_LTC_GPIO),
535       .hps_io_gpio_inst_GPIO48(HPS_I2C_CONTROL),
536       .hps_io_gpio_inst_GPIO53(HPS_LED),
537       .hps_io_gpio_inst_GPIO54(HPS_KEY),
538       .hps_io_gpio_inst_GPIO61(HPS_GSENSOR_INT)
539   );
540
541    // Tie unused board peripherals to stable values or high-Z so the DE1-SoC
542    // top-level pinout remains complete without changing the active design.
543    assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
544    assign ADC_DIN = SW[0];
545    assign ADC_SCLK = SW[0];
546
547    assign AUD_ADCLRCK = SW[1] ? SW[0] : 1'bZ;
548    assign AUD_BCLK = SW[1] ? SW[0] : 1'bZ;
549    assign AUD_DACDAT = SW[0];
550    assign AUD_DACLRCK = SW[1] ? SW[0] : 1'bZ;
551    assign AUD_XCK = SW[0];
552
553    assign DRAM_ADDR = { 13{ SW[0] } };
554    assign DRAM_BA = { 2{ SW[0] } };
555    assign DRAM_DQ = SW[1] ? { 16{ SW[0] } } : DRAM_DQ_HIZ;
556    assign {DRAM_CAS_N, DRAM_CKE, DRAM_CLK, DRAM_CS_N,
```

```
           DRAM_LDQM, DRAM_RAS_N, DRAM_UDQM, DRAM_WE_N} = { 8{SW[0]} };

  assign FAN_CTRL = SW[0];

  assign FPGA_I2C_SCLK = SW[0];
  assign FPGA_I2C_SDAT = SW[1] ? SW[0] : 1'bZ;

  assign GPIO_0 = GPIO_HIZ;
  assign GPIO_1 = GPIO_HIZ;

  assign HEX0 = hex_digit_to_7seg(predicted_digit);
  assign HEX1 = HEX_OFF;
  assign HEX2 = HEX_OFF;
  assign HEX3 = HEX_OFF;
  assign HEX4 = HEX_OFF;
  assign HEX5 = HEX_OFF;

  assign IRDA_TXD = SW[0];

  assign LEDR[0] = (predicted_digit == 4'd0);
  assign LEDR[4:1] = predicted_digit;
  assign LEDR[9:5] = 5'd0;

  assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
  assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
  assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
  assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

  assign TD_RESET_N = SW[0];

  assign {VGA_R, VGA_G, VGA_B} = { 24{ SW[0] } };
  assign {VGA_BLANK_N, VGA_CLK,
          VGA_HS, VGA_SYNC_N, VGA_VS} = { 5{ SW[0] } };
endmodule
```

Listing 11: Shared LeNet INT8 constants and dimensions.

```
package lenet_int8_pkg;
  parameter int PIX_W = 8;
  parameter int ACT_W = 16;
  parameter int WGT_W = 8;
  parameter int BIAS_W = 8;
  parameter int ACC_W = 32;
  parameter int MAC_LANES = 3;
  parameter int SHIFT_W = 5;
  parameter int ACT_MAX = (1 << ACT_W) - 1;

  parameter int IMG_W = 32;
  parameter int IMG_PIXELS = IMG_W * IMG_W;
  parameter int MAX_FM_COUNT = 4704;

  parameter int CONV1_IN_CH = 1;
  parameter int CONV1_OUT_CH = 6;
  parameter int CONV1_K = 5;
  parameter int CONV1_CONV_W = 28;
  parameter int CONV1_POOL_W = 14;
  parameter int CONV1_POOL_COUNT = CONV1_OUT_CH * CONV1_POOL_W * CONV1_POOL_W;
  parameter int CONV1_WEIGHT_COUNT = CONV1_OUT_CH * CONV1_IN_CH * CONV1_K * CONV1_K;
```

```
  parameter int CONV1_BIAS_COUNT = CONV1_OUT_CH;
  parameter int CONV1_PARAM_COUNT = CONV1_WEIGHT_COUNT + CONV1_BIAS_COUNT;

  parameter int CONV2_IN_CH = 6;
  parameter int CONV2_OUT_CH = 16;
  parameter int CONV2_K = 5;
  parameter int CONV2_CONV_W = 10;
  parameter int CONV2_POOL_W = 5;
  parameter int CONV2_POOL_COUNT = CONV2_OUT_CH * CONV2_POOL_W * CONV2_POOL_W;
  parameter int CONV2_WEIGHT_COUNT = CONV2_OUT_CH * CONV2_IN_CH * CONV2_K * CONV2_K;
  parameter int CONV2_BIAS_COUNT = CONV2_OUT_CH;
  parameter int CONV2_PARAM_COUNT = CONV2_WEIGHT_COUNT + CONV2_BIAS_COUNT;

  parameter int FC1_IN_COUNT = 400;
  parameter int FC1_OUT_COUNT = 120;
  parameter int FC1_WEIGHT_COUNT = FC1_IN_COUNT * FC1_OUT_COUNT;
  parameter int FC1_BIAS_COUNT = FC1_OUT_COUNT;
  parameter int FC1_PARAM_COUNT = FC1_WEIGHT_COUNT + FC1_BIAS_COUNT;

  parameter int FC2_IN_COUNT = 120;
  parameter int FC2_OUT_COUNT = 84;
  parameter int FC2_WEIGHT_COUNT = FC2_IN_COUNT * FC2_OUT_COUNT;
  parameter int FC2_BIAS_COUNT = FC2_OUT_COUNT;
  parameter int FC2_PARAM_COUNT = FC2_WEIGHT_COUNT + FC2_BIAS_COUNT;

  parameter int OUT_IN_COUNT = 84;
  parameter int OUT_COUNT = 10;
  parameter int OUT_WEIGHT_COUNT = OUT_IN_COUNT * OUT_COUNT;
  parameter int OUT_BIAS_COUNT = OUT_COUNT;
  parameter int OUT_PARAM_COUNT = OUT_WEIGHT_COUNT + OUT_BIAS_COUNT;

  parameter int CONV1_SHIFT = 6;
  parameter int CONV2_SHIFT = 8;
  parameter int FC1_SHIFT = 8;
  parameter int FC2_SHIFT = 7;
endpackage
```

Listing 12: Parameterized BRAM and ROM memory modules.

```
module BRAM #(
  parameter DATA_WIDTH = 8,
  parameter ADDR_WIDTH = 10,
  parameter INIT_FILE = ""

)(
input clk,
input w_en,
input [ADDR_WIDTH-1:0] w_addr,
input r_en,
input [ADDR_WIDTH-1:0] r_addr,
input [DATA_WIDTH-1:0] w_data,
output reg [DATA_WIDTH-1:0] r_data
);

(* ramstyle = "M10K" *) reg [DATA_WIDTH-1:0] mem [0:(1<<ADDR_WIDTH)-1];

initial begin
  if (INIT_FILE != "") begin
```

```
    $readmemh(INIT_FILE, mem);
  end
end

always@(posedge clk)begin
  if(w_en) mem[w_addr] <= w_data;
end

always@(posedge clk)begin
  if(r_en) r_data <= mem[r_addr];
end

endmodule

module PACKED_ROM #(
  parameter DATA_WIDTH = 8,
  parameter ADDR_WIDTH = 10,
  parameter LANES = 2,
  parameter WORD_COUNT = (1 << ADDR_WIDTH),
  parameter INIT_FILE = ""
)(
input clk,
input r_en,
input [ADDR_WIDTH-1:0] r_addr,
output reg [LANES*DATA_WIDTH-1:0] r_data
);

(* ramstyle = "M10K" *) reg [LANES*DATA_WIDTH-1:0] mem [0:WORD_COUNT-1];

initial begin
  if (INIT_FILE != "") begin
    $readmemh(INIT_FILE, mem);
  end
end

always@(posedge clk)begin
  if(r_en) r_data <= mem[r_addr];
end

endmodule
```

Listing 13: HPS-to-FPGA memory-mapped register interface.

```
module hps_lenet_regs #(
  parameter REG_COUNT = 10,
  parameter PIXELS = 1024,
  parameter PIX_BASE_WORD = 12
) (
  input  wire                     clk,
  input  wire                     rst_n,
  input  wire                     regs_valid,
  input  wire                     accel_busy,
  input  wire                     accel_done,
  input  wire [3:0]               predicted_digit,
  input  wire [27:0]              cycle_count,
  input  wire [REG_COUNT*32-1:0]  regs_flat,
  output reg                      start_pulse,
  output reg                      clear_done_pulse,
```

```
16    output reg                       pixel_wr_en,
17    output reg  [$clog2(PIXELS)-1:0] pixel_wr_addr,
18    output reg  [7:0]                pixel_wr_data,
19    output reg                       pixel_rd_en,
20    output reg  [$clog2(PIXELS)-1:0] pixel_rd_addr,
21    input  wire [7:0]                pixel_rd_data,
22    output reg                       input_loaded,
23
24    input  wire [11:0]               s_awid,
25    input  wire [20:0]               s_awaddr,
26    input  wire                      s_awvalid,
27    output wire                      s_awready,
28    input  wire [31:0]               s_wdata,
29    input  wire [3:0]                s_wstrb,
30    input  wire                      s_wlast,
31    input  wire                      s_wvalid,
32    output wire                      s_wready,
33    output reg  [11:0]               s_bid,
34    output reg  [1:0]                s_bresp,
35    output reg                       s_bvalid,
36    input  wire                      s_bready,
37
38    input  wire [11:0]               s_arid,
39    input  wire [20:0]               s_araddr,
40    input  wire                      s_arvalid,
41    output wire                      s_arready,
42    output reg  [11:0]               s_rid,
43    output reg  [31:0]               s_rdata,
44    output reg  [1:0]                s_rresp,
45    output reg                       s_rlast,
46    output reg                       s_rvalid,
47    input  wire                      s_rready
48  );
49
50    localparam [1:0] AXI_RESP_OKAY = 2'b00;
51    localparam integer PIX_REG_COUNT = PIXELS;
52    localparam [1:0] READ_IDLE = 2'd0;
53    localparam [1:0] READ_PIXEL_WAIT_BRAM = 2'd1;
54
55    reg [11:0] awid_latched;
56    reg [20:0] awaddr_latched;
57    reg [31:0] wdata_latched;
58    reg        aw_seen;
59    reg        w_seen;
60    reg [1:0]  read_state;
61    reg [11:0] pending_rid;
62
63    function automatic [31:0] read_register_word;//status reg
64      input [20:0] addr;
65      reg [11:0] word_index;
66      begin
67        word_index = addr[13:2];
68        if (word_index == 0) begin
69          read_register_word = {
70            28'd0,
71            input_loaded,
72            accel_done,
73            accel_busy,
```

```
74          regs_valid
75        };
76      end else if (word_index == 1) begin
77        read_register_word = {cycle_count, predicted_digit};
78      end else if ((word_index - 2) < REG_COUNT) begin
79        read_register_word = regs_flat[(word_index - 2) * 32 +: 32];
80      end else begin
81        read_register_word = 32'd0;
82      end
83    end
84  endfunction
85
86  function automatic is_pixel_window;
87    input [20:0] addr;
88    reg [11:0] word_index;
89    begin
90      word_index = addr[13:2];
91      is_pixel_window =
92        (word_index >= PIX_BASE_WORD) &&
93        ((word_index - PIX_BASE_WORD) < PIX_REG_COUNT);
94    end
95  endfunction
96
97  function automatic integer pixel_word_index;
98    input [20:0] addr;
99    reg [11:0] word_index;
100   begin
101     word_index = addr[13:2];
102     pixel_word_index = word_index - PIX_BASE_WORD;
103   end
104 endfunction
105
106 assign s_arready = !s_rvalid && (read_state == READ_IDLE);
107 assign s_awready = !s_bvalid && !aw_seen;
108 assign s_wready  = !s_bvalid && !w_seen;
109
110 always @(posedge clk or negedge rst_n) begin
111   if (!rst_n) begin
112     s_rid       <= 12'd0;
113     s_rdata     <= 32'd0;
114     s_rresp     <= AXI_RESP_OKAY;
115     s_rlast     <= 1'b0;
116     s_rvalid    <= 1'b0;
117     pixel_rd_en <= 1'b0;
118     pixel_rd_addr <= {$clog2(PIXELS){1'b0}};
119     pending_rid <= 12'd0;
120     read_state  <= READ_IDLE;
121   end else begin
122     pixel_rd_en <= 1'b0;
123
124     if (s_rvalid && s_rready) begin
125       s_rvalid <= 1'b0;
126       s_rlast  <= 1'b0;
127     end
128
129     case (read_state)
130       READ_IDLE: begin
131         if (!s_rvalid && s_arvalid) begin
```

```
132             if (is_pixel_window(s_araddr)) begin
133               pending_rid <= s_arid; // pixel  BRAM  BRAM  READ_PIXEL_WAIT_BRAM
134               pixel_rd_en <= 1'b1;
135               pixel_rd_addr <= pixel_word_index(s_araddr);
136               read_state <= READ_PIXEL_WAIT_BRAM;
137             end else begin
138               s_rid   <= s_arid;
139               s_rdata <= read_register_word(s_araddr);
140               s_rresp <= AXI_RESP_OKAY;
141               s_rlast <= 1'b1;
142               s_rvalid <= 1'b1;
143             end
144           end
145         end
146
147         READ_PIXEL_WAIT_BRAM: begin
148           if (!s_rvalid) begin
149             s_rid   <= pending_rid;
150             s_rdata <= {24'd0, pixel_rd_data};
151             s_rresp <= AXI_RESP_OKAY;
152             s_rlast <= 1'b1;
153             s_rvalid <= 1'b1;
154             read_state <= READ_IDLE;
155           end
156         end
157
158         default: begin
159           read_state <= READ_IDLE;
160         end
161       endcase
162     end
163   end
164
165   always @(posedge clk or negedge rst_n) begin
166     if (!rst_n) begin
167       start_pulse <= 1'b0;
168       clear_done_pulse <= 1'b0;
169       pixel_wr_en <= 1'b0;
170       pixel_wr_addr <= {$clog2(PIXELS){1'b0}};
171       pixel_wr_data <= 8'd0;
172       input_loaded <= 1'b0;
173       awid_latched <= 12'd0;
174       awaddr_latched <= 21'd0;
175       wdata_latched <= 32'd0;
176       aw_seen      <= 1'b0;
177       w_seen       <= 1'b0;
178       s_bid        <= 12'd0;
179       s_bresp      <= AXI_RESP_OKAY;
180       s_bvalid     <= 1'b0;
181     end else begin
182       if (s_bvalid && s_bready) begin
183         s_bvalid <= 1'b0;
184       end
185
186       start_pulse <= 1'b0;
187       clear_done_pulse <= 1'b0;
188       pixel_wr_en <= 1'b0;
189
```

```
      if (!aw_seen && s_awvalid) begin
        awid_latched <= s_awid;
        awaddr_latched <= s_awaddr;
        aw_seen <= 1'b1;
      end

      if (!w_seen && s_wvalid) begin
        wdata_latched <= s_wdata;
        w_seen <= 1'b1;
      end

      if (!s_bvalid && aw_seen && w_seen) begin
        if (is_pixel_window(awaddr_latched)) begin
          pixel_wr_en <= 1'b1;
          pixel_wr_addr <= pixel_word_index(awaddr_latched);
          pixel_wr_data <= wdata_latched[7:0];
          if (pixel_word_index(awaddr_latched) == (PIXELS - 1)) begin
            input_loaded <= 1'b1;
          end
        end else if (awaddr_latched[13:2] == 0) begin
          if (wdata_latched[0]) begin
            start_pulse <= 1'b1;
          end
          if (wdata_latched[1]) begin
            clear_done_pulse <= 1'b1;
          end
          if (wdata_latched[2]) begin
            input_loaded <= 1'b0;
          end
        end

        s_bid    <= awid_latched;
        s_bresp  <= AXI_RESP_OKAY;
        s_bvalid <= 1'b1;
        aw_seen  <= 1'b0;
        w_seen   <= 1'b0;
      end
    end
  end

endmodule
```

Listing 14: Convolution and pooling address-generation logic.

```
module lenet_addr_calc (
  input logic [3:0] out_ch,
  input logic [4:0] out_y,
  input logic [4:0] out_x,
  input logic [3:0] pool_ch,
  input logic [3:0] pool_y,
  input logic [3:0] pool_x,
  input logic [8:0] term_idx,
  input logic [1:0] pool_read_idx,
  output logic [$clog2(lenet_int8_pkg::IMG_PIXELS)-1:0] conv1_src_addr,
  output logic [$clog2(lenet_int8_pkg::MAX_FM_COUNT)-1:0] conv1_dst_addr,
  output logic [$clog2(lenet_int8_pkg::MAX_FM_COUNT)-1:0] pool1_src_addr,
  output logic [$clog2(lenet_int8_pkg::MAX_FM_COUNT)-1:0] pool1_dst_addr,
  output logic [$clog2(lenet_int8_pkg::MAX_FM_COUNT)-1:0] conv2_src_addr,
```

```
  output logic [$clog2(lenet_int8_pkg::MAX_FM_COUNT)-1:0] conv2_dst_addr,
  output logic [$clog2(lenet_int8_pkg::MAX_FM_COUNT)-1:0] pool2_src_addr,
  output logic [$clog2(lenet_int8_pkg::MAX_FM_COUNT)-1:0] pool2_dst_addr
);
  import lenet_int8_pkg::*;

  logic [2:0] conv1_ky;
  logic [2:0] conv1_kx;
  logic       pool1_dy;
  logic       pool1_dx;
  logic [2:0] conv2_ic;
  logic [4:0] conv2_tap;
  logic [2:0] conv2_ky;
  logic [2:0] conv2_kx;
  logic       pool2_dy;
  logic       pool2_dx;

  function automatic logic [2:0] tap5_y(input logic [8:0] idx);
    begin
      if (idx < 5) begin
        tap5_y = 3'd0;
      end else if (idx < 10) begin
        tap5_y = 3'd1;
      end else if (idx < 15) begin
        tap5_y = 3'd2;
      end else if (idx < 20) begin
        tap5_y = 3'd3;
      end else begin
        tap5_y = 3'd4;
      end
    end
  endfunction

  function automatic logic [2:0] tap5_x(input logic [8:0] idx);
    begin
      case (idx)
        0, 5, 10, 15, 20: tap5_x = 3'd0;
        1, 6, 11, 16, 21: tap5_x = 3'd1;
        2, 7, 12, 17, 22: tap5_x = 3'd2;
        3, 8, 13, 18, 23: tap5_x = 3'd3;
        default:          tap5_x = 3'd4;
      endcase
    end
  endfunction

  function automatic logic [2:0] conv2_in_ch(input logic [8:0] idx);
    begin
      if (idx < 25) begin
        conv2_in_ch = 3'd0;
      end else if (idx < 50) begin
        conv2_in_ch = 3'd1;
      end else if (idx < 75) begin
        conv2_in_ch = 3'd2;
      end else if (idx < 100) begin
        conv2_in_ch = 3'd3;
      end else if (idx < 125) begin
        conv2_in_ch = 3'd4;
      end else begin
```

```
73          conv2_in_ch = 3'd5;
74        end
75      end
76    endfunction
77
78    function automatic logic [4:0] conv2_tap_idx(input logic [8:0] idx);
79      begin
80        if (idx < 25) begin
81          conv2_tap_idx = idx[4:0];
82        end else if (idx < 50) begin
83          conv2_tap_idx = idx[4:0] - 5'd25;
84        end else if (idx < 75) begin
85          conv2_tap_idx = idx[4:0] - 5'd18;
86        end else if (idx < 100) begin
87          conv2_tap_idx = idx[4:0] - 5'd11;
88        end else if (idx < 125) begin
89          conv2_tap_idx = idx[4:0] - 5'd4;
90        end else begin
91          conv2_tap_idx = idx[4:0] - 5'd29;
92        end
93      end
94    endfunction
95
96    always_comb begin
97      conv1_ky = tap5_y(term_idx);
98      conv1_kx = tap5_x(term_idx);
99      conv1_src_addr = ((out_y + conv1_ky) * IMG_W) + (out_x + conv1_kx);
100     conv1_dst_addr = (out_ch * CONV1_CONV_W * CONV1_CONV_W) + (out_y * CONV1_CONV_W) + out_x;
101
102     pool1_dy = (pool_read_idx >= 2);
103     pool1_dx = pool_read_idx[0];
104     pool1_src_addr = (pool_ch * CONV1_CONV_W * CONV1_CONV_W) + ((pool_y * 2 + pool1_dy) *
        CONV1_CONV_W) + (pool_x * 2 + pool1_dx);
105     pool1_dst_addr = (pool_ch * CONV1_POOL_W * CONV1_POOL_W) + (pool_y * CONV1_POOL_W) + pool_x;
106
107     conv2_ic = conv2_in_ch(term_idx);
108     conv2_tap = conv2_tap_idx(term_idx);
109     conv2_ky = tap5_y(conv2_tap);
110     conv2_kx = tap5_x(conv2_tap);
111     conv2_src_addr = (conv2_ic * CONV1_POOL_W * CONV1_POOL_W) + ((out_y + conv2_ky) * CONV1_POOL_W)
        + (out_x + conv2_kx);
112     conv2_dst_addr = (out_ch * CONV2_CONV_W * CONV2_CONV_W) + (out_y * CONV2_CONV_W) + out_x;
113
114     pool2_dy = (pool_read_idx >= 2);
115     pool2_dx = pool_read_idx[0];
116     pool2_src_addr = (pool_ch * CONV2_CONV_W * CONV2_CONV_W) + ((pool_y * 2 + pool2_dy) *
        CONV2_CONV_W) + (pool_x * 2 + pool2_dx);
117     pool2_dst_addr = (pool_ch * CONV2_POOL_W * CONV2_POOL_W) + (pool_y * CONV2_POOL_W) + pool_x;
118   end
119 endmodule
```

Listing 15: Lane-parallel multiply-accumulate array.

```
module lenet_mac_array (
  input  logic [lenet_int8_pkg::MAC_LANES * lenet_int8_pkg::ACT_W-1:0] activations_flat,
  input  logic signed [lenet_int8_pkg::MAC_LANES * lenet_int8_pkg::WGT_W-1:0] weights_flat,
  input  logic [lenet_int8_pkg::MAC_LANES-1:0] valid_mask,
  output logic signed [lenet_int8_pkg::ACC_W-1:0] partial_sum
);
  import lenet_int8_pkg::*;

  integer lane;
  logic signed [ACT_W:0] act_value;
  logic signed [WGT_W-1:0] weight_value;

  always_comb begin
    partial_sum = '0;
    for (lane = 0; lane < MAC_LANES; lane = lane + 1) begin
      act_value = $signed({1'b0, activations_flat[lane * ACT_W +: ACT_W]});
      weight_value = weights_flat[lane * WGT_W +: WGT_W];
      if (valid_mask[lane]) begin
        partial_sum = partial_sum + (act_value * weight_value);
      end
    end
  end
endmodule
```

Listing 16: Shift-based quantization and ReLU clamp logic.

```
module lenet_quantize (
  input  logic signed [lenet_int8_pkg::ACC_W-1:0] value_in,
  input  logic [lenet_int8_pkg::SHIFT_W-1:0] shift_right,
  output logic [lenet_int8_pkg::ACT_W-1:0] value_out
);
  import lenet_int8_pkg::*;

  logic signed [ACC_W-1:0] rounded_value;
  logic signed [ACC_W-1:0] scaled_value;
  logic signed [ACC_W-1:0] round_offset;

  always_comb begin
    value_out = '0;
    rounded_value = value_in;
    scaled_value = value_in;
    round_offset = '0;

    if (value_in <= 0) begin
      value_out = '0;
    end else begin
      if (shift_right != 0) begin
        round_offset = $signed(32'd1 <<< (shift_right - 1'b1));
        rounded_value = value_in + round_offset;
        scaled_value = rounded_value >>> shift_right;
      end

      if (scaled_value > ACT_MAX) begin
        value_out = ACT_W'(ACT_MAX);
      end else begin
        value_out = scaled_value[ACT_W-1:0];
```

```
      end
    end
  end
endmodule
```

Listing 17: Sequential argmax module for final digit prediction.

```
module lenet_argmax (
  input logic signed [lenet_int8_pkg::OUT_COUNT * lenet_int8_pkg::ACC_W-1:0] logits_flat,
  output logic [3:0] digit
);
  import lenet_int8_pkg::*;

  integer idx;
  integer best_idx;
  logic signed [ACC_W-1:0] best_score;
  logic signed [ACC_W-1:0] score;

  always_comb begin
    best_score = $signed(logits_flat[0 +: ACC_W]);
    best_idx = 0;
    for (idx = 1; idx < OUT_COUNT; idx = idx + 1) begin
      score = $signed(logits_flat[idx * ACC_W +: ACC_W]);
      if (score > best_score) begin
        best_score = score;
        best_idx = idx;
      end
    end
    digit = best_idx[3:0];
  end
endmodule
```

Listing 18: Top-level INT8 LeNet accelerator core.

```
module lenet_int8_top #(
  parameter CONV1_PARAM_FILE = "D:/Coursefile/ee4840_embedded/embedded_lab/parameter/
    weights_bias_422/conv1_pack3.hex",
  parameter CONV2_PARAM_FILE = "D:/Coursefile/ee4840_embedded/embedded_lab/parameter/
    weights_bias_422/conv2_pack3.hex",
  parameter FC1_PARAM_FILE   = "D:/Coursefile/ee4840_embedded/embedded_lab/parameter/
    weights_bias_422/fc1_pack3.hex",
  parameter FC2_PARAM_FILE   = "D:/Coursefile/ee4840_embedded/embedded_lab/parameter/
    weights_bias_422/fc2_pack3.hex",
  parameter OUT_PARAM_FILE   = "D:/Coursefile/ee4840_embedded/embedded_lab/parameter/
    weights_bias_422/fc3_pack3.hex"
) (
  input  logic clk,
  input  logic rst_n,
  input  logic start,
  input  logic pixel_wr_en,
  input  logic [$clog2(lenet_int8_pkg::IMG_PIXELS)-1:0] pixel_wr_addr,
  input  logic [lenet_int8_pkg::PIX_W-1:0] pixel_wr_data,
  input  logic pixel_dbg_rd_en,
  input  logic [$clog2(lenet_int8_pkg::IMG_PIXELS)-1:0] pixel_dbg_rd_addr,
  output logic [lenet_int8_pkg::PIX_W-1:0] pixel_dbg_rd_data,
  output logic busy,
  output logic done,
  output logic [3:0] predicted_digit,
```

```
20    output logic signed [lenet_int8_pkg::OUT_COUNT * lenet_int8_pkg::ACC_W-1:0] logits_flat,
21    output logic [27:0] cycle_count
22  );
23    import lenet_int8_pkg::*;
24
25    localparam int IMG_ADDR_W = $clog2(IMG_PIXELS);
26    localparam int FM_ADDR_W = $clog2(MAX_FM_COUNT);
27    localparam int CONV1_PARAM_ADDR_W = $clog2(CONV1_PARAM_COUNT);
28    localparam int CONV2_PARAM_ADDR_W = $clog2(CONV2_PARAM_COUNT);
29    localparam int FC1_PARAM_ADDR_W = $clog2(FC1_PARAM_COUNT);
30    localparam int FC2_PARAM_ADDR_W = $clog2(FC2_PARAM_COUNT);
31    localparam int OUT_PARAM_ADDR_W = $clog2(OUT_PARAM_COUNT);
32    localparam int CONV1_WORDS_PER_ROW = ((CONV1_IN_CH * CONV1_K * CONV1_K) + MAC_LANES - 1) /
        MAC_LANES;
33    localparam int CONV2_WORDS_PER_ROW = ((CONV2_IN_CH * CONV2_K * CONV2_K) + MAC_LANES - 1) /
        MAC_LANES;
34    localparam int FC1_WORDS_PER_ROW = (FC1_IN_COUNT + MAC_LANES - 1) / MAC_LANES;
35    localparam int FC2_WORDS_PER_ROW = (FC2_IN_COUNT + MAC_LANES - 1) / MAC_LANES;
36    localparam int OUT_WORDS_PER_ROW = (OUT_IN_COUNT + MAC_LANES - 1) / MAC_LANES;
37    localparam int CONV1_WEIGHT_WORD_COUNT = CONV1_OUT_CH * CONV1_WORDS_PER_ROW;
38    localparam int CONV2_WEIGHT_WORD_COUNT = CONV2_OUT_CH * CONV2_WORDS_PER_ROW;
39    localparam int FC1_WEIGHT_WORD_COUNT = FC1_OUT_COUNT * FC1_WORDS_PER_ROW;
40    localparam int FC2_WEIGHT_WORD_COUNT = FC2_OUT_COUNT * FC2_WORDS_PER_ROW;
41    localparam int OUT_WEIGHT_WORD_COUNT = OUT_COUNT * OUT_WORDS_PER_ROW;
42    localparam int CONV1_PARAM_WORD_COUNT = CONV1_WEIGHT_WORD_COUNT + CONV1_BIAS_COUNT;
43    localparam int CONV2_PARAM_WORD_COUNT = CONV2_WEIGHT_WORD_COUNT + CONV2_BIAS_COUNT;
44    localparam int FC1_PARAM_WORD_COUNT = FC1_WEIGHT_WORD_COUNT + FC1_BIAS_COUNT;
45    localparam int FC2_PARAM_WORD_COUNT = FC2_WEIGHT_WORD_COUNT + FC2_BIAS_COUNT;
46    localparam int OUT_PARAM_WORD_COUNT = OUT_WEIGHT_WORD_COUNT + OUT_BIAS_COUNT;
47    localparam int CONV1_PARAM_WORD_ADDR_W = $clog2(CONV1_PARAM_WORD_COUNT);
48    localparam int CONV2_PARAM_WORD_ADDR_W = $clog2(CONV2_PARAM_WORD_COUNT);
49    localparam int FC1_PARAM_WORD_ADDR_W = $clog2(FC1_PARAM_WORD_COUNT);
50    localparam int FC2_PARAM_WORD_ADDR_W = $clog2(FC2_PARAM_WORD_COUNT);
51    localparam int OUT_PARAM_WORD_ADDR_W = $clog2(OUT_PARAM_WORD_COUNT);
52
53    typedef enum logic [5:0] {
54      ST_IDLE,
55      ST_CONV1_INIT,
56      ST_CONV1_READ,
57      ST_CONV1_ACCUM,
58      ST_CONV1_ADD,
59      ST_CONV1_WRITE,
60      ST_POOL1_INIT,
61      ST_POOL1_READ,
62      ST_POOL1_ACCUM,
63      ST_POOL1_WRITE,
64      ST_CONV2_INIT,
65      ST_CONV2_READ,
66      ST_CONV2_ACCUM,
67      ST_CONV2_ADD,
68      ST_CONV2_WRITE,
69      ST_POOL2_INIT,
70      ST_POOL2_READ,
71      ST_POOL2_ACCUM,
72      ST_POOL2_WRITE,
73      ST_FC1_INIT,
74      ST_FC1_READ,
75      ST_FC1_ACCUM,
```

```
76      ST_FC1_ADD,
77      ST_FC1_WRITE,
78      ST_FC2_INIT,
79      ST_FC2_READ,
80      ST_FC2_ACCUM,
81      ST_FC2_ADD,
82      ST_FC2_WRITE,
83      ST_FC3_INIT,
84      ST_FC3_READ,
85      ST_FC3_ACCUM,
86      ST_FC3_ADD,
87      ST_FC3_WRITE,
88      ST_ARGMAX_INIT,
89      ST_ARGMAX_SCAN,
90      ST_DONE
91    } state_t;
92
93    state_t state;
94
95    logic [MAC_LANES-1:0] img_r_en;
96    logic [IMG_ADDR_W-1:0] img_r_addr [0:MAC_LANES-1];
97    logic [PIX_W-1:0] img_r_data [0:MAC_LANES-1];
98
99    logic [MAC_LANES-1:0] fm_a_r_en;
100   logic [FM_ADDR_W-1:0] fm_a_r_addr [0:MAC_LANES-1];
101   logic [ACT_W-1:0] fm_a_r_data [0:MAC_LANES-1];
102   logic fm_a_w_en;
103   logic [FM_ADDR_W-1:0] fm_a_w_addr;
104   logic [ACT_W-1:0] fm_a_w_data;
105
106   logic [MAC_LANES-1:0] fm_b_r_en;
107   logic [FM_ADDR_W-1:0] fm_b_r_addr [0:MAC_LANES-1];
108   logic [ACT_W-1:0] fm_b_r_data [0:MAC_LANES-1];
109   logic fm_b_w_en;
110   logic [FM_ADDR_W-1:0] fm_b_w_addr;
111   logic [ACT_W-1:0] fm_b_w_data;
112
113   logic signed [ACC_W-1:0] logits_mem [0:OUT_COUNT-1];
114
115   logic [MAC_LANES-1:0] conv1_param_r_en;
116   logic [CONV1_PARAM_WORD_ADDR_W-1:0] conv1_param_r_addr [0:MAC_LANES-1];
117   logic [WGT_W-1:0] conv1_param_r_data [0:MAC_LANES-1];
118   logic [MAC_LANES-1:0] conv2_param_r_en;
119   logic [CONV2_PARAM_WORD_ADDR_W-1:0] conv2_param_r_addr [0:MAC_LANES-1];
120   logic [WGT_W-1:0] conv2_param_r_data [0:MAC_LANES-1];
121   logic [MAC_LANES-1:0] fc1_param_r_en;
122   logic [FC1_PARAM_WORD_ADDR_W-1:0] fc1_param_r_addr [0:MAC_LANES-1];
123   logic [WGT_W-1:0] fc1_param_r_data [0:MAC_LANES-1];
124   logic [MAC_LANES-1:0] fc2_param_r_en;
125   logic [FC2_PARAM_WORD_ADDR_W-1:0] fc2_param_r_addr [0:MAC_LANES-1];
126   logic [WGT_W-1:0] fc2_param_r_data [0:MAC_LANES-1];
127   logic [MAC_LANES-1:0] out_param_r_en;
128   logic [OUT_PARAM_WORD_ADDR_W-1:0] out_param_r_addr [0:MAC_LANES-1];
129   logic [WGT_W-1:0] out_param_r_data [0:MAC_LANES-1];
130   logic [MAC_LANES * WGT_W-1:0] conv1_param_r_data_flat;
131   logic [MAC_LANES * WGT_W-1:0] conv2_param_r_data_flat;
132   logic [MAC_LANES * WGT_W-1:0] fc1_param_r_data_flat;
133   logic [MAC_LANES * WGT_W-1:0] fc2_param_r_data_flat;
```

```
134   logic [MAC_LANES * WGT_W-1:0] out_param_r_data_flat;
135
136   logic [MAC_LANES * ACT_W-1:0] mac_activation;
137   logic signed [MAC_LANES * WGT_W-1:0] mac_weight;
138   logic [MAC_LANES-1:0] mac_valid;
139   logic signed [ACC_W-1:0] mac_partial_sum;
140   logic signed [ACC_W-1:0] mac_partial_sum_r;
141   logic signed [ACC_W-1:0] acc_reg;
142   logic signed [BIAS_W-1:0] current_bias;
143   logic [SHIFT_W-1:0] current_shift;
144   logic signed [ACC_W-1:0] acc_with_bias;
145   logic [ACT_W-1:0] quantized_value;
146   logic [ACT_W-1:0] pool_max_reg;
147   logic [18:0] cycle_counter;
148
149   integer out_ch;
150   integer out_y;
151   integer out_x;
152   integer pool_ch;
153   integer pool_y;
154   integer pool_x;
155   integer neuron_idx;
156   logic [8:0] term_idx;
157   logic [8:0] term_word_idx;
158   integer lane_idx;
159   integer pool_read_idx;
160   integer flat_idx;
161   integer reset_idx;
162   logic [3:0] argmax_index;
163   logic [3:0] argmax_best_digit;
164   logic signed [ACC_W-1:0] argmax_best_score;
165   logic [8:0] term_idx_lane [0:MAC_LANES-1];
166   logic [MAC_LANES-1:0] term_valid;
167   logic [IMG_ADDR_W-1:0] addr_conv1_src [0:MAC_LANES-1];
168   logic [FM_ADDR_W-1:0] addr_conv1_dst [0:MAC_LANES-1];
169   logic [FM_ADDR_W-1:0] addr_pool1_src [0:MAC_LANES-1];
170   logic [FM_ADDR_W-1:0] addr_pool1_dst [0:MAC_LANES-1];
171   logic [FM_ADDR_W-1:0] addr_conv2_src [0:MAC_LANES-1];
172   logic [FM_ADDR_W-1:0] addr_conv2_dst [0:MAC_LANES-1];
173   logic [FM_ADDR_W-1:0] addr_pool2_src [0:MAC_LANES-1];
174   logic [FM_ADDR_W-1:0] addr_pool2_dst [0:MAC_LANES-1];
175
176   genvar mem_lane;
177   generate
178     for (mem_lane = 0; mem_lane < MAC_LANES; mem_lane = mem_lane + 1) begin : g_lane_mem
179       BRAM #(
180         .DATA_WIDTH(PIX_W),
181         .ADDR_WIDTH(IMG_ADDR_W)
182       ) u_image_mem (
183         .clk(clk),
184         .w_en(pixel_wr_en),
185         .w_addr(pixel_wr_addr),
186         .r_en((mem_lane == 0) ? (pixel_dbg_rd_en | img_r_en[mem_lane]) : img_r_en[mem_lane]),
187         .r_addr((mem_lane == 0 && pixel_dbg_rd_en) ? pixel_dbg_rd_addr : img_r_addr[mem_lane]),
188         .w_data(pixel_wr_data),
189         .r_data(img_r_data[mem_lane])
190       );
191
```

```
192       BRAM #(
193         .DATA_WIDTH(ACT_W),
194         .ADDR_WIDTH(FM_ADDR_W)
195       ) u_fm_a (
196         .clk(clk),
197         .w_en(fm_a_w_en),
198         .w_addr(fm_a_w_addr),
199         .r_en(fm_a_r_en[mem_lane]),
200         .r_addr(fm_a_r_addr[mem_lane]),
201         .w_data(fm_a_w_data),
202         .r_data(fm_a_r_data[mem_lane])
203       );
204
205       BRAM #(
206         .DATA_WIDTH(ACT_W),
207         .ADDR_WIDTH(FM_ADDR_W)
208       ) u_fm_b (
209         .clk(clk),
210         .w_en(fm_b_w_en),
211         .w_addr(fm_b_w_addr),
212         .r_en(fm_b_r_en[mem_lane]),
213         .r_addr(fm_b_r_addr[mem_lane]),
214         .w_data(fm_b_w_data),
215         .r_data(fm_b_r_data[mem_lane])
216       );
217     end
218   endgenerate
219
220   PACKED_ROM #(
221     .DATA_WIDTH(WGT_W),
222     .ADDR_WIDTH(CONV1_PARAM_WORD_ADDR_W),
223     .LANES(MAC_LANES),
224     .WORD_COUNT(CONV1_PARAM_WORD_COUNT),
225     .INIT_FILE(CONV1_PARAM_FILE)
226   ) u_conv1_params (
227     .clk(clk),
228     .r_en(|conv1_param_r_en),
229     .r_addr(conv1_param_r_addr[0]),
230     .r_data(conv1_param_r_data_flat)
231   );
232
233   PACKED_ROM #(
234     .DATA_WIDTH(WGT_W),
235     .ADDR_WIDTH(CONV2_PARAM_WORD_ADDR_W),
236     .LANES(MAC_LANES),
237     .WORD_COUNT(CONV2_PARAM_WORD_COUNT),
238     .INIT_FILE(CONV2_PARAM_FILE)
239   ) u_conv2_params (
240     .clk(clk),
241     .r_en(|conv2_param_r_en),
242     .r_addr(conv2_param_r_addr[0]),
243     .r_data(conv2_param_r_data_flat)
244   );
245
246   PACKED_ROM #(
247     .DATA_WIDTH(WGT_W),
248     .ADDR_WIDTH(FC1_PARAM_WORD_ADDR_W),
249     .LANES(MAC_LANES),
```

```
    .WORD_COUNT(FC1_PARAM_WORD_COUNT),
    .INIT_FILE(FC1_PARAM_FILE)
  ) u_fc1_params (
    .clk(clk),
    .r_en(|fc1_param_r_en),
    .r_addr(fc1_param_r_addr[0]),
    .r_data(fc1_param_r_data_flat)
  );

  PACKED_ROM #(
    .DATA_WIDTH(WGT_W),
    .ADDR_WIDTH(FC2_PARAM_WORD_ADDR_W),
    .LANES(MAC_LANES),
    .WORD_COUNT(FC2_PARAM_WORD_COUNT),
    .INIT_FILE(FC2_PARAM_FILE)
  ) u_fc2_params (
    .clk(clk),
    .r_en(|fc2_param_r_en),
    .r_addr(fc2_param_r_addr[0]),
    .r_data(fc2_param_r_data_flat)
  );

  PACKED_ROM #(
    .DATA_WIDTH(WGT_W),
    .ADDR_WIDTH(OUT_PARAM_WORD_ADDR_W),
    .LANES(MAC_LANES),
    .WORD_COUNT(OUT_PARAM_WORD_COUNT),
    .INIT_FILE(OUT_PARAM_FILE)
  ) u_out_params (
    .clk(clk),
    .r_en(|out_param_r_en),
    .r_addr(out_param_r_addr[0]),
    .r_data(out_param_r_data_flat)
  );

  genvar param_lane;
  generate
    for (param_lane = 0; param_lane < MAC_LANES; param_lane = param_lane + 1) begin : g_param_unpack
      assign conv1_param_r_data[param_lane] = conv1_param_r_data_flat[param_lane * WGT_W +: WGT_W];
      assign conv2_param_r_data[param_lane] = conv2_param_r_data_flat[param_lane * WGT_W +: WGT_W];
      assign fc1_param_r_data[param_lane] = fc1_param_r_data_flat[param_lane * WGT_W +: WGT_W];
      assign fc2_param_r_data[param_lane] = fc2_param_r_data_flat[param_lane * WGT_W +: WGT_W];
      assign out_param_r_data[param_lane] = out_param_r_data_flat[param_lane * WGT_W +: WGT_W];
    end
  endgenerate

  lenet_mac_array u_mac (
    .activations_flat(mac_activation),
    .weights_flat(mac_weight),
    .valid_mask(mac_valid),
    .partial_sum(mac_partial_sum)
  );

  lenet_quantize u_quantize (
    .value_in(acc_with_bias),
    .shift_right(current_shift),
    .value_out(quantized_value)
  );
```

```
308
309 generate
310   for (mem_lane = 0; mem_lane < MAC_LANES; mem_lane = mem_lane + 1) begin : g_addr_calc
311     lenet_addr_calc u_addr_calc (
312       .out_ch(out_ch[3:0]),
313       .out_y(out_y[4:0]),
314       .out_x(out_x[4:0]),
315       .pool_ch(pool_ch[3:0]),
316       .pool_y(pool_y[3:0]),
317       .pool_x(pool_x[3:0]),
318       .term_idx(term_idx_lane[mem_lane]),
319       .pool_read_idx(pool_read_idx[1:0]),
320       .conv1_src_addr(addr_conv1_src[mem_lane]),
321       .conv1_dst_addr(addr_conv1_dst[mem_lane]),
322       .pool1_src_addr(addr_pool1_src[mem_lane]),
323       .pool1_dst_addr(addr_pool1_dst[mem_lane]),
324       .conv2_src_addr(addr_conv2_src[mem_lane]),
325       .conv2_dst_addr(addr_conv2_dst[mem_lane]),
326       .pool2_src_addr(addr_pool2_src[mem_lane]),
327       .pool2_dst_addr(addr_pool2_dst[mem_lane])
328     );
329   end
330 endgenerate
331
332 always_comb begin
333   pixel_dbg_rd_data = img_r_data[0];
334
335   img_r_en = '0;
336   fm_a_r_en = '0;
337   fm_a_w_en = 1'b0;
338   fm_a_w_addr = '0;
339   fm_a_w_data = '0;
340   fm_b_r_en = '0;
341   fm_b_w_en = 1'b0;
342   fm_b_w_addr = '0;
343   fm_b_w_data = '0;
344   conv1_param_r_en = '0;
345   conv2_param_r_en = '0;
346   fc1_param_r_en = '0;
347   fc2_param_r_en = '0;
348   out_param_r_en = '0;
349   mac_activation = '0;
350   mac_weight = '0;
351   mac_valid = '0;
352   current_bias = '0;
353   current_shift = '0;
354   term_valid = '0;
355
356   for (lane_idx = 0; lane_idx < MAC_LANES; lane_idx = lane_idx + 1) begin
357     img_r_addr[lane_idx] = '0;
358     fm_a_r_addr[lane_idx] = '0;
359     fm_b_r_addr[lane_idx] = '0;
360     conv1_param_r_addr[lane_idx] = '0;
361     conv2_param_r_addr[lane_idx] = '0;
362     fc1_param_r_addr[lane_idx] = '0;
363     fc2_param_r_addr[lane_idx] = '0;
364     out_param_r_addr[lane_idx] = '0;
365     term_idx_lane[lane_idx] = term_idx + lane_idx;
```

```
    end

    logits_flat = '0;
    for (flat_idx = 0; flat_idx < OUT_COUNT; flat_idx = flat_idx + 1) begin
      logits_flat[flat_idx * ACC_W +: ACC_W] = logits_mem[flat_idx];
    end

    case (state)
      ST_CONV1_READ: begin
        for (lane_idx = 0; lane_idx < MAC_LANES; lane_idx = lane_idx + 1) begin
          term_valid[lane_idx] = ((term_idx + lane_idx) < (CONV1_K * CONV1_K));
          img_r_en[lane_idx] = term_valid[lane_idx];
          img_r_addr[lane_idx] = addr_conv1_src[lane_idx];
          conv1_param_r_en[lane_idx] = term_valid[lane_idx];
        end
        conv1_param_r_addr[0] = (out_ch * CONV1_WORDS_PER_ROW) + term_word_idx;
      end

      ST_CONV1_ACCUM: begin
        for (lane_idx = 0; lane_idx < MAC_LANES; lane_idx = lane_idx + 1) begin
          mac_activation[lane_idx * ACT_W +: ACT_W] = img_r_data[lane_idx];
          mac_weight[lane_idx * WGT_W +: WGT_W] = conv1_param_r_data[lane_idx];
          mac_valid[lane_idx] = ((term_idx + lane_idx) < (CONV1_K * CONV1_K));
        end
        current_shift = CONV1_SHIFT;
        if ((term_idx + MAC_LANES) >= (CONV1_K * CONV1_K)) begin
          conv1_param_r_en[0] = 1'b1;
          conv1_param_r_addr[0] = CONV1_WEIGHT_WORD_COUNT + out_ch;
        end
      end

      ST_CONV1_WRITE: begin
        current_bias = conv1_param_r_data[0];
        current_shift = CONV1_SHIFT;
        fm_a_w_en = 1'b1;
        fm_a_w_addr = addr_conv1_dst[0];
        fm_a_w_data = quantized_value;
      end

      ST_POOL1_READ: begin
        fm_a_r_en[0] = 1'b1;
        fm_a_r_addr[0] = addr_pool1_src[0];
      end

      ST_POOL1_WRITE: begin
        fm_b_w_en = 1'b1;
        fm_b_w_addr = addr_pool1_dst[0];
        fm_b_w_data = pool_max_reg;
      end

      ST_CONV2_READ: begin
        for (lane_idx = 0; lane_idx < MAC_LANES; lane_idx = lane_idx + 1) begin
          term_valid[lane_idx] = ((term_idx + lane_idx) < (CONV2_IN_CH * CONV2_K * CONV2_K));
          fm_b_r_en[lane_idx] = term_valid[lane_idx];
          fm_b_r_addr[lane_idx] = addr_conv2_src[lane_idx];
          conv2_param_r_en[lane_idx] = term_valid[lane_idx];
        end
        conv2_param_r_addr[0] = (out_ch * CONV2_WORDS_PER_ROW) + term_word_idx;
```

```
424         end
425
426         ST_CONV2_ACCUM: begin
427           for (lane_idx = 0; lane_idx < MAC_LANES; lane_idx = lane_idx + 1) begin
428             mac_activation[lane_idx * ACT_W +: ACT_W] = fm_b_r_data[lane_idx];
429             mac_weight[lane_idx * WGT_W +: WGT_W] = conv2_param_r_data[lane_idx];
430             mac_valid[lane_idx] = ((term_idx + lane_idx) < (CONV2_IN_CH * CONV2_K * CONV2_K));
431           end
432           current_shift = CONV2_SHIFT;
433           if ((term_idx + MAC_LANES) >= (CONV2_IN_CH * CONV2_K * CONV2_K)) begin
434             conv2_param_r_en[0] = 1'b1;
435             conv2_param_r_addr[0] = CONV2_WEIGHT_WORD_COUNT + out_ch;
436           end
437         end
438
439         ST_CONV2_WRITE: begin
440           current_bias = conv2_param_r_data[0];
441           current_shift = CONV2_SHIFT;
442           fm_a_w_en = 1'b1;
443           fm_a_w_addr = addr_conv2_dst[0];
444           fm_a_w_data = quantized_value;
445         end
446
447         ST_POOL2_READ: begin
448           fm_a_r_en[0] = 1'b1;
449           fm_a_r_addr[0] = addr_pool2_src[0];
450         end
451
452         ST_POOL2_WRITE: begin
453           fm_b_w_en = 1'b1;
454           fm_b_w_addr = addr_pool2_dst[0];
455           fm_b_w_data = pool_max_reg;
456         end
457
458         ST_FC1_READ: begin
459           for (lane_idx = 0; lane_idx < MAC_LANES; lane_idx = lane_idx + 1) begin
460             term_valid[lane_idx] = ((term_idx + lane_idx) < FC1_IN_COUNT);
461             fm_b_r_en[lane_idx] = term_valid[lane_idx];
462             fm_b_r_addr[lane_idx] = term_idx + lane_idx;
463             fc1_param_r_en[lane_idx] = term_valid[lane_idx];
464           end
465           fc1_param_r_addr[0] = (neuron_idx * FC1_WORDS_PER_ROW) + term_word_idx;
466         end
467
468         ST_FC1_ACCUM: begin
469           for (lane_idx = 0; lane_idx < MAC_LANES; lane_idx = lane_idx + 1) begin
470             mac_activation[lane_idx * ACT_W +: ACT_W] = fm_b_r_data[lane_idx];
471             mac_weight[lane_idx * WGT_W +: WGT_W] = fc1_param_r_data[lane_idx];
472             mac_valid[lane_idx] = ((term_idx + lane_idx) < FC1_IN_COUNT);
473           end
474           current_shift = FC1_SHIFT;
475           if ((term_idx + MAC_LANES) >= FC1_IN_COUNT) begin
476             fc1_param_r_en[0] = 1'b1;
477             fc1_param_r_addr[0] = FC1_WEIGHT_WORD_COUNT + neuron_idx;
478           end
479         end
480
481         ST_FC1_WRITE: begin
```

```
482         current_bias = fc1_param_r_data[0];
483         current_shift = FC1_SHIFT;
484         fm_a_w_en = 1'b1;
485         fm_a_w_addr = neuron_idx;
486         fm_a_w_data = quantized_value;
487       end
488
489       ST_FC2_READ: begin
490         for (lane_idx = 0; lane_idx < MAC_LANES; lane_idx = lane_idx + 1) begin
491           term_valid[lane_idx] = ((term_idx + lane_idx) < FC2_IN_COUNT);
492           fm_a_r_en[lane_idx] = term_valid[lane_idx];
493           fm_a_r_addr[lane_idx] = term_idx + lane_idx;
494           fc2_param_r_en[lane_idx] = term_valid[lane_idx];
495         end
496         fc2_param_r_addr[0] = (neuron_idx * FC2_WORDS_PER_ROW) + term_word_idx;
497       end
498
499       ST_FC2_ACCUM: begin
500         for (lane_idx = 0; lane_idx < MAC_LANES; lane_idx = lane_idx + 1) begin
501           mac_activation[lane_idx * ACT_W +: ACT_W] = fm_a_r_data[lane_idx];
502           mac_weight[lane_idx * WGT_W +: WGT_W] = fc2_param_r_data[lane_idx];
503           mac_valid[lane_idx] = ((term_idx + lane_idx) < FC2_IN_COUNT);
504         end
505         current_shift = FC2_SHIFT;
506         if ((term_idx + MAC_LANES) >= FC2_IN_COUNT) begin
507           fc2_param_r_en[0] = 1'b1;
508           fc2_param_r_addr[0] = FC2_WEIGHT_WORD_COUNT + neuron_idx;
509         end
510       end
511
512       ST_FC2_WRITE: begin
513         current_bias = fc2_param_r_data[0];
514         current_shift = FC2_SHIFT;
515         fm_b_w_en = 1'b1;
516         fm_b_w_addr = neuron_idx;
517         fm_b_w_data = quantized_value;
518       end
519
520       ST_FC3_READ: begin
521         for (lane_idx = 0; lane_idx < MAC_LANES; lane_idx = lane_idx + 1) begin
522           term_valid[lane_idx] = ((term_idx + lane_idx) < OUT_IN_COUNT);
523           fm_b_r_en[lane_idx] = term_valid[lane_idx];
524           fm_b_r_addr[lane_idx] = term_idx + lane_idx;
525           out_param_r_en[lane_idx] = term_valid[lane_idx];
526         end
527         out_param_r_addr[0] = (neuron_idx * OUT_WORDS_PER_ROW) + term_word_idx;
528       end
529
530       ST_FC3_ACCUM: begin
531         for (lane_idx = 0; lane_idx < MAC_LANES; lane_idx = lane_idx + 1) begin
532           mac_activation[lane_idx * ACT_W +: ACT_W] = fm_b_r_data[lane_idx];
533           mac_weight[lane_idx * WGT_W +: WGT_W] = out_param_r_data[lane_idx];
534           mac_valid[lane_idx] = ((term_idx + lane_idx) < OUT_IN_COUNT);
535         end
536         if ((term_idx + MAC_LANES) >= OUT_IN_COUNT) begin
537           out_param_r_en[0] = 1'b1;
538           out_param_r_addr[0] = OUT_WEIGHT_WORD_COUNT + neuron_idx;
539         end
```

```
      end

      ST_FC3_WRITE: begin
        current_bias = out_param_r_data[0];
      end

      default: begin
      end
    endcase
  end

  assign acc_with_bias = acc_reg + {{(ACC_W-BIAS_W){current_bias[BIAS_W-1]}}, current_bias};

  always_ff @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      state <= ST_IDLE;
      acc_reg <= '0;
      predicted_digit <= '0;
      argmax_index <= '0;
      argmax_best_digit <= '0;
      argmax_best_score <= '0;
      out_ch <= 0;
      out_y <= 0;
      out_x <= 0;
      pool_ch <= 0;
      pool_y <= 0;
      pool_x <= 0;
      neuron_idx <= 0;
      term_idx <= 0;
      term_word_idx <= 0;
      pool_read_idx <= 0;
      pool_max_reg <= '0;
      mac_partial_sum_r <= '0;
      for (reset_idx = 0; reset_idx < OUT_COUNT; reset_idx = reset_idx + 1) begin
        logits_mem[reset_idx] <= '0;
      end
      cycle_counter <= 19'd0;
    end else begin
      if ((state == ST_IDLE) && start) begin
        cycle_counter <= 19'd1;
      end else if ((state != ST_IDLE) && (state != ST_DONE)) begin
        cycle_counter <= cycle_counter + 19'd1;
      end

      case (state)
        ST_IDLE: begin
          acc_reg <= '0;
          if (start) begin
            predicted_digit <= '0;
            argmax_index <= '0;
            argmax_best_digit <= '0;
            argmax_best_score <= '0;
            out_ch <= 0;
            out_y <= 0;
            out_x <= 0;
            term_idx <= 0;
            term_word_idx <= 0;
            for (reset_idx = 0; reset_idx < OUT_COUNT; reset_idx = reset_idx + 1) begin
```

```
598             logits_mem[reset_idx] <= '0;
599           end
600           state <= ST_CONV1_INIT;
601         end
602       end
603
604       ST_CONV1_INIT: begin
605         acc_reg <= '0;
606         term_idx <= 0;
607         term_word_idx <= 0;
608         state <= ST_CONV1_READ;
609       end
610       ST_CONV1_READ: state <= ST_CONV1_ACCUM;
611       ST_CONV1_ACCUM: begin
612         mac_partial_sum_r <= mac_partial_sum;
613         state <= ST_CONV1_ADD;
614       end
615       ST_CONV1_ADD: begin
616         acc_reg <= acc_reg + mac_partial_sum_r;
617         if ((term_idx + MAC_LANES) >= (CONV1_K * CONV1_K)) begin
618           state <= ST_CONV1_WRITE;
619         end else begin
620           term_idx <= term_idx + MAC_LANES;
621           term_word_idx <= term_word_idx + 1;
622           state <= ST_CONV1_READ;
623         end
624       end
625       ST_CONV1_WRITE: begin
626         if (out_x == (CONV1_CONV_W - 1)) begin
627           out_x <= 0;
628           if (out_y == (CONV1_CONV_W - 1)) begin
629             out_y <= 0;
630             if (out_ch == (CONV1_OUT_CH - 1)) begin
631               out_ch <= 0;
632               pool_ch <= 0;
633               pool_y <= 0;
634               pool_x <= 0;
635               state <= ST_POOL1_INIT;
636             end else begin
637               out_ch <= out_ch + 1;
638               state <= ST_CONV1_INIT;
639             end
640           end else begin
641             out_y <= out_y + 1;
642             state <= ST_CONV1_INIT;
643           end
644         end else begin
645           out_x <= out_x + 1;
646           state <= ST_CONV1_INIT;
647         end
648       end
649
650       ST_POOL1_INIT: begin
651         pool_read_idx <= 0;
652         pool_max_reg <= '0;
653         state <= ST_POOL1_READ;
654       end
655       ST_POOL1_READ: state <= ST_POOL1_ACCUM;
```

```
656         ST_POOL1_ACCUM: begin
657           if ((pool_read_idx == 0) || (fm_a_r_data[0] > pool_max_reg)) begin
658             pool_max_reg <= fm_a_r_data[0];
659           end
660           if (pool_read_idx == 3) begin
661             state <= ST_POOL1_WRITE;
662           end else begin
663             pool_read_idx <= pool_read_idx + 1;
664             state <= ST_POOL1_READ;
665           end
666         end
667         ST_POOL1_WRITE: begin
668           if (pool_x == (CONV1_POOL_W - 1)) begin
669             pool_x <= 0;
670             if (pool_y == (CONV1_POOL_W - 1)) begin
671               pool_y <= 0;
672               if (pool_ch == (CONV1_OUT_CH - 1)) begin
673                 pool_ch <= 0;
674                 out_ch <= 0;
675                 out_y <= 0;
676                 out_x <= 0;
677                 state <= ST_CONV2_INIT;
678               end else begin
679                 pool_ch <= pool_ch + 1;
680                 state <= ST_POOL1_INIT;
681               end
682             end else begin
683               pool_y <= pool_y + 1;
684               state <= ST_POOL1_INIT;
685             end
686           end else begin
687             pool_x <= pool_x + 1;
688             state <= ST_POOL1_INIT;
689           end
690         end
691
692         ST_CONV2_INIT: begin
693           acc_reg <= '0;
694           term_idx <= 0;
695           term_word_idx <= 0;
696           state <= ST_CONV2_READ;
697         end
698         ST_CONV2_READ: state <= ST_CONV2_ACCUM;
699         ST_CONV2_ACCUM: begin
700           mac_partial_sum_r <= mac_partial_sum;
701           state <= ST_CONV2_ADD;
702         end
703         ST_CONV2_ADD: begin
704           acc_reg <= acc_reg + mac_partial_sum_r;
705           if ((term_idx + MAC_LANES) >= (CONV2_IN_CH * CONV2_K * CONV2_K)) begin
706             state <= ST_CONV2_WRITE;
707           end else begin
708             term_idx <= term_idx + MAC_LANES;
709             term_word_idx <= term_word_idx + 1;
710             state <= ST_CONV2_READ;
711           end
712         end
713         ST_CONV2_WRITE: begin
```

```
714             if (out_x == (CONV2_CONV_W - 1)) begin
715               out_x <= 0;
716               if (out_y == (CONV2_CONV_W - 1)) begin
717                 out_y <= 0;
718                 if (out_ch == (CONV2_OUT_CH - 1)) begin
719                   out_ch <= 0;
720                   pool_ch <= 0;
721                   pool_y <= 0;
722                   pool_x <= 0;
723                   state <= ST_POOL2_INIT;
724                 end else begin
725                   out_ch <= out_ch + 1;
726                   state <= ST_CONV2_INIT;
727                 end
728               end else begin
729                 out_y <= out_y + 1;
730                 state <= ST_CONV2_INIT;
731               end
732             end else begin
733               out_x <= out_x + 1;
734               state <= ST_CONV2_INIT;
735             end
736           end
737
738           ST_POOL2_INIT: begin
739             pool_read_idx <= 0;
740             pool_max_reg <= '0;
741             state <= ST_POOL2_READ;
742           end
743           ST_POOL2_READ: state <= ST_POOL2_ACCUM;
744           ST_POOL2_ACCUM: begin
745             if ((pool_read_idx == 0) || (fm_a_r_data[0] > pool_max_reg)) begin
746               pool_max_reg <= fm_a_r_data[0];
747             end
748             if (pool_read_idx == 3) begin
749               state <= ST_POOL2_WRITE;
750             end else begin
751               pool_read_idx <= pool_read_idx + 1;
752               state <= ST_POOL2_READ;
753             end
754           end
755           ST_POOL2_WRITE: begin
756             if (pool_x == (CONV2_POOL_W - 1)) begin
757               pool_x <= 0;
758               if (pool_y == (CONV2_POOL_W - 1)) begin
759                 pool_y <= 0;
760                 if (pool_ch == (CONV2_OUT_CH - 1)) begin
761                   pool_ch <= 0;
762                   neuron_idx <= 0;
763                   state <= ST_FC1_INIT;
764                 end else begin
765                   pool_ch <= pool_ch + 1;
766                   state <= ST_POOL2_INIT;
767                 end
768               end else begin
769                 pool_y <= pool_y + 1;
770                 state <= ST_POOL2_INIT;
771               end
```

```
772           end else begin
773             pool_x <= pool_x + 1;
774             state <= ST_POOL2_INIT;
775           end
776         end
777
778         ST_FC1_INIT: begin
779           acc_reg <= '0;
780           term_idx <= 0;
781           term_word_idx <= 0;
782           state <= ST_FC1_READ;
783         end
784         ST_FC1_READ: state <= ST_FC1_ACCUM;
785         ST_FC1_ACCUM: begin
786           mac_partial_sum_r <= mac_partial_sum;
787           state <= ST_FC1_ADD;
788         end
789         ST_FC1_ADD: begin
790           acc_reg <= acc_reg + mac_partial_sum_r;
791           if ((term_idx + MAC_LANES) >= FC1_IN_COUNT) begin
792             state <= ST_FC1_WRITE;
793           end else begin
794             term_idx <= term_idx + MAC_LANES;
795             term_word_idx <= term_word_idx + 1;
796             state <= ST_FC1_READ;
797           end
798         end
799         ST_FC1_WRITE: begin
800           if (neuron_idx == (FC1_OUT_COUNT - 1)) begin
801             neuron_idx <= 0;
802             state <= ST_FC2_INIT;
803           end else begin
804             neuron_idx <= neuron_idx + 1;
805             state <= ST_FC1_INIT;
806           end
807         end
808
809         ST_FC2_INIT: begin
810           acc_reg <= '0;
811           term_idx <= 0;
812           term_word_idx <= 0;
813           state <= ST_FC2_READ;
814         end
815         ST_FC2_READ: state <= ST_FC2_ACCUM;
816         ST_FC2_ACCUM: begin
817           mac_partial_sum_r <= mac_partial_sum;
818           state <= ST_FC2_ADD;
819         end
820         ST_FC2_ADD: begin
821           acc_reg <= acc_reg + mac_partial_sum_r;
822           if ((term_idx + MAC_LANES) >= FC2_IN_COUNT) begin
823             state <= ST_FC2_WRITE;
824           end else begin
825             term_idx <= term_idx + MAC_LANES;
826             term_word_idx <= term_word_idx + 1;
827             state <= ST_FC2_READ;
828           end
829         end
```

```
830         ST_FC2_WRITE: begin
831           if (neuron_idx == (FC2_OUT_COUNT - 1)) begin
832             neuron_idx <= 0;
833             state <= ST_FC3_INIT;
834           end else begin
835             neuron_idx <= neuron_idx + 1;
836             state <= ST_FC2_INIT;
837           end
838         end
839 
840         ST_FC3_INIT: begin
841           acc_reg <= '0;
842           term_idx <= 0;
843           term_word_idx <= 0;
844           state <= ST_FC3_READ;
845         end
846         ST_FC3_READ: state <= ST_FC3_ACCUM;
847         ST_FC3_ACCUM: begin
848           mac_partial_sum_r <= mac_partial_sum;
849           state <= ST_FC3_ADD;
850         end
851         ST_FC3_ADD: begin
852           acc_reg <= acc_reg + mac_partial_sum_r;
853           if ((term_idx + MAC_LANES) >= OUT_IN_COUNT) begin
854             state <= ST_FC3_WRITE;
855           end else begin
856             term_idx <= term_idx + MAC_LANES;
857             term_word_idx <= term_word_idx + 1;
858             state <= ST_FC3_READ;
859           end
860         end
861         ST_FC3_WRITE: begin
862           logits_mem[neuron_idx] <= acc_with_bias;
863           if (neuron_idx == (OUT_COUNT - 1)) begin
864             state <= ST_ARGMAX_INIT;
865           end else begin
866             neuron_idx <= neuron_idx + 1;
867             state <= ST_FC3_INIT;
868           end
869         end
870 
871         ST_ARGMAX_INIT: begin
872           argmax_index <= 4'd1;
873           argmax_best_digit <= 4'd0;
874           argmax_best_score <= logits_mem[0];
875           state <= ST_ARGMAX_SCAN;
876         end
877 
878         ST_ARGMAX_SCAN: begin
879           if (logits_mem[argmax_index] > argmax_best_score) begin
880             argmax_best_score <= logits_mem[argmax_index];
881             argmax_best_digit <= argmax_index;
882           end
883 
884           if (argmax_index == (OUT_COUNT - 1)) begin
885             predicted_digit <= (logits_mem[argmax_index] > argmax_best_score) ? argmax_index :
    argmax_best_digit;
886             state <= ST_DONE;
```

```
          end else begin
            argmax_index <= argmax_index + 4'd1;
          end
        end

        ST_DONE: begin
          state <= ST_IDLE;
        end

        default: begin
          state <= ST_IDLE;
        end
      endcase
    end
  end

  assign busy = (state != ST_IDLE) && (state != ST_DONE);
  assign done = (state == ST_DONE);
  assign cycle_count = {9'd0, cycle_counter};
endmodule
```

A.4 Testbench Code

Listing 19: ModelSim testbench for the INT8 LeNet accelerator.

```
`timescale 1ns/1ps

module tb_lenet_int8_top;
  import lenet_int8_pkg::*;

  logic clk;
  logic rst_n;
  logic start;

  logic pixel_wr_en;
  logic [$clog2(IMG_PIXELS)-1:0] pixel_wr_addr;
  logic [PIX_W-1:0] pixel_wr_data;

  logic pixel_dbg_rd_en;
  logic [$clog2(IMG_PIXELS)-1:0] pixel_dbg_rd_addr;
  logic [PIX_W-1:0] pixel_dbg_rd_data;

  logic busy;
  logic done;
  logic [3:0] predicted_digit;
  logic signed [OUT_COUNT * ACC_W-1:0] logits_flat;
  logic [27:0] cycle_count;

  integer i;
  integer wait_cycles;
  integer logit_index;
  logic signed [ACC_W-1:0] one_logit;

  lenet_int8_top dut (
    .clk(clk),
    .rst_n(rst_n),
    .start(start),
```

```
    .pixel_wr_en(pixel_wr_en),
    .pixel_wr_addr(pixel_wr_addr),
    .pixel_wr_data(pixel_wr_data),
    .pixel_dbg_rd_en(pixel_dbg_rd_en),
    .pixel_dbg_rd_addr(pixel_dbg_rd_addr),
    .pixel_dbg_rd_data(pixel_dbg_rd_data),
    .busy(busy),
    .done(done),
    .predicted_digit(predicted_digit),
    .logits_flat(logits_flat),
    .cycle_count(cycle_count)
  );

  initial begin
    clk = 1'b0;
    forever #10 clk = ~clk;
  end

  initial begin
    rst_n = 1'b0;
    start = 1'b0;
    pixel_wr_en = 1'b0;
    pixel_wr_addr = '0;
    pixel_wr_data = '0;
    pixel_dbg_rd_en = 1'b0;
    pixel_dbg_rd_addr = '0;

    repeat (5) @(posedge clk);
    rst_n = 1'b1;
    repeat (2) @(posedge clk);

    // Write a simple 32x32 test image. Every pixel is 1.
    for (i = 0; i < IMG_PIXELS; i = i + 1) begin
      pixel_wr_en = 1'b1;
      pixel_wr_addr = i;
      pixel_wr_data = 8'd1;
      @(posedge clk);
    end

    pixel_wr_en = 1'b0;
    pixel_wr_addr = '0;
    pixel_wr_data = '0;
    repeat (2) @(posedge clk);

    // Start one inference.
    start = 1'b1;
    @(posedge clk);
    start = 1'b0;

    wait_cycles = 0;
    while ((done == 1'b0) && (wait_cycles < 2000000)) begin
      @(posedge clk);
      wait_cycles = wait_cycles + 1;
    end

    if (done == 1'b1) begin
      $display("Inference␣finished.");
      $display("predicted_digit␣=␣%0d", predicted_digit);
```

```
      $display("cycle_count␣␣␣␣␣=␣%0d", cycle_count);

      for (logit_index = 0; logit_index < OUT_COUNT; logit_index = logit_index + 1) begin
        one_logit = logits_flat[logit_index * ACC_W +: ACC_W];
        $display("logit[%0d]␣=␣%0d", logit_index, one_logit);
      end
    end else begin
      $display("ERROR:␣timeout␣waiting␣for␣done.");
    end

    repeat (10) @(posedge clk);
    $finish;
  end
endmodule
```

Listing 20: ModelSim run script for compiling and simulating the accelerator testbench.

```
transcript on

if {[file exists work]} {
  vdel -lib work -all
}
vlib work
vmap work work

vlog -sv ../RTL/lenet_int8_pkg.sv
vlog -sv ../RTL/BRAM.sv
vlog -sv ../RTL/lenet_addr_calc.sv
vlog -sv ../RTL/lenet_mac_array.sv
vlog -sv ../RTL/lenet_quantize.sv
vlog -sv ../RTL/lenet_argmax.sv
vlog -sv ../RTL/lenet_int8_top.sv
vlog -sv tb_lenet_int8_top.sv

vsim -voptargs=+acc work.tb_lenet_int8_top

add wave -divider "Testbench Control"
add wave sim:/tb_lenet_int8_top/clk
add wave sim:/tb_lenet_int8_top/rst_n
add wave sim:/tb_lenet_int8_top/start
add wave sim:/tb_lenet_int8_top/busy
add wave sim:/tb_lenet_int8_top/done

add wave -divider "Pixel Write Interface"
add wave sim:/tb_lenet_int8_top/pixel_wr_en
add wave -radix unsigned sim:/tb_lenet_int8_top/pixel_wr_addr
add wave -radix unsigned sim:/tb_lenet_int8_top/pixel_wr_data

add wave -divider "CNN Output"
add wave -radix unsigned sim:/tb_lenet_int8_top/predicted_digit
add wave -radix unsigned sim:/tb_lenet_int8_top/cycle_count
add wave -radix signed sim:/tb_lenet_int8_top/logits_flat

add wave -divider "Internal FSM"
add wave sim:/tb_lenet_int8_top/dut/state
add wave -radix unsigned sim:/tb_lenet_int8_top/dut/out_ch
add wave -radix unsigned sim:/tb_lenet_int8_top/dut/out_y
add wave -radix unsigned sim:/tb_lenet_int8_top/dut/out_x
```

```
add wave -radix unsigned sim:/tb_lenet_int8_top/dut/term_idx
add wave -radix signed sim:/tb_lenet_int8_top/dut/acc_reg

run -all
wave zoom full
```

B Final FPGA Parameter Files

The following files are the final lane-packed INT8 parameter memories used by the FPGA accelerator through `$readmemh`. Unpacked software-reference parameter files are summarized in Table 3; the packed `pack3` files are listed here because they are the files directly consumed by the final RTL.

Listing 21: Packed Conv1 weights and biases for three-lane FPGA ROM loading.

```
bedff0
24eecd
e9cbc7
f73408
09ebdd
062d3f
4bff42
4c584c
000026
33200f
ab0731
c0e7c1
8fe8e7
dfe081
f10310
32d7de
232a29
00002e
262814
001816
3ded2e
fed406
3e1d32
294605
27f730
3437fe
000004
ea24fa
3ccffd
23274c
34023e
1c48f7
ed06d9
c50e21
29fa09
0000ec
f5ec1a
f6b9e9
d00933
2215ed
03f319
311747
```

```
35420b
16de22
00000b
130000
cd2e1c
0b38e1
fb102b
292150
350a2f
e242f8
f419d5
000010
0000fb
00007f
000004
000001
000027
000007
```

Listing 22: Packed Conv2 weights and biases for three-lane FPGA ROM loading.

```
2228ee
eee800
2312ea
f6bcdf
f50b20
1615f7
e12913
05f8e3
fcea11
f3f1dc
fbceb1
def2e9
0cefd1
cfe9e6
10ead8
e00ee6
22cef2
d0eeee
281ded
ecd80d
f22602
1dfce6
1b210a
07f211
eb1c25
e41410
f6e403
dce200
fe1d00
efe71b
3714e3
f0e508
1a221a
1ee905
ee10fa
24f2ee
e5e3e6
```

```
ff150d
0fdf04
111f11
08eb0a
1e0900
fafefa
fc1f1d
11eacc
e81cfc
00e5ed
f80f12
180af6
111dfe
3934fc
bbd0fe
345917
c0ce13
074729
fdccd9
ea2439
0ff6f0
12e219
e70c2f
3ce3ca
e50429
13c79d
c8e52a
e3f2c4
e7f0e4
15d1e1
e5dd14
0808e1
e5e918
001b07
e1d6e5
151d18
f5d6f7
e7e920
00f617
2712da
fb231a
1202e8
09122a
3ff9dd
ccfd26
f2ffdd
1614f0
edff0e
0b29e3
c2e713
163806
efebfd
fe0825
03dfe3
110d14
e0f2e1
221fed
f2dfdf
```

```
fe261a
dae402
ff231d
ede4de
e5e70c
1103fc
c1222f
ede4cc
ddc608
d5d8db
040305
fee8ea
0e2d1f
321322
0d1415
0c1a0b
38f6fc
354141
360928
492c34
2c1bf8
000e1f
201c16
e60627
0c050b
ecd5d3
d41e10
d1d6e0
f2e106
f50a0e
fcfcfc
241414
0f280a
dbf10a
f800ea
eba5bd
0ed4e1
eaf1eb
242ad5
1d081d
f5ee16
d8fffc
ddd1da
f212d7
d5e8d6
f8fdf6
0d04fd
19f0fa
11190d
280023
ddd2dd
ef01ef
e1d5b3
ed13da
01e6f5
e8fb1b
17fefb
f2d72a
```

```
1d0afe
e6ebd2
c4c2f7
d1e5f0
16edca
fa17dc
fafa16
000400
f1df21
e9f700
0dfda9
99d715
270ff9
1e9df4
f6da11
f93901
25e4cd
c4f026
05441a
0dc9f2
e7f43c
293be1
ddfb14
1506ec
f7e71c
e92c1c
2a11cd
d1e310
fe140a
03d9dc
19f70b
1df6ef
e1362a
3ffb00
f5e617
0d2300
f9ffd9
d9e80a
ff3a00
e8f7e3
ea060b
1904dd
c7fd35
001f09
3adecc
bc0e3b
322713
01fafa
f2fbed
2402e0
22efff
f5f514
000d24
24f7cd
cde81a
f1f301
e0f513
190e00
```

```
01ddd5
c7e92a
040cb1
d7c1b5
bddaf2
1d18ed
ffd1dc
d10320
010f0e
0ed9e4
f7032f
201f05
1e04eb
e010f0
f4df09
111911
f80fe2
ddd621
fc0613
fe0001
ef0e22
0b10ff
010efd
03fd0b
e4f415
dd1e1b
0b0fea
f60a23
161402
1908f4
201417
02f809
00f412
212616
01dcf0
d9ed25
230615
2602d3
e10b1e
4e4419
1f321e
394215
cbe8ec
bcb4a6
f80000
fb1e0b
454025
341b20
031637
0dfafb
02d8db
1a041a
54052f
565e3c
1114f0
07e406
0e1bf4
10f630
```

```
26192b
ef0d0b
00ff1c
dec7d1
00e3dd
f0f5d6
fefb03
09ebf6
281e1d
2f0b15
022e18
b7f111
efdac5
dfeded
13fbe2
25131c
431f2a
f81d1e
bdd0d9
ef01c8
fedcef
f812f5
f20318
01ffdf
0eff28
1c0818
1dff02
08ecf9
e1c9c9
dee9e4
f3e1ce
b8e4d4
1faad7
e5ed18
211ed6
2e2904
090ff7
0b283b
0b1a14
150af3
e8ec13
173b34
1b4b36
2f1e0e
ee0960
06ff1e
10ffee
ea1d01
e9d1c0
d0cf0b
cdb2e5
e0f4e2
17e3e0
1500f9
020c28
130f12
efcbe1
c6f5e4
```

```
afd8b6
fcedd0
dcd8f3
1a060e
fe2b20
001422
e1f728
cfd9d8
efd9f6
1bcce1
200204
130a1f
2d2400
100518
df0016
f5e8d4
b8ba1b
fab4d7
e5f3ef
0be1ea
f5f11a
f30915
202d01
1921f6
12dcf4
eedb0b
fad01d
131a1a
4c0be9
49d42e
e6383b
e809af
15f2cf
fc0303
da2909
3610ef
2e0bf7
aad91d
b6f617
efd9b4
e40a20
eeffd7
effbd7
cfd7e1
0730f9
3721f6
231920
d82709
030afb
fc1311
f40bea
fe0ce4
f508f4
000ee1
1ef904
213002
240fe9
df11fc
```

```
fd16e1
e1faf3
08dee1
09f800
0ff920
1a331e
e4f0f7
16031e
0a0d12
00eeec
d0d6dc
d618f5
260be3
130018
d91f3b
f4f5fe
0a161a
a9afc0
2f38cb
051227
441af9
ec1126
2d240d
dfe5eb
eac7ee
152527
3f0b2e
58645d
fdf557
26162f
e7e800
0018ff
3c2310
d0de26
11ecce
c4f00d
f4d3db
ffd4f0
1e05e2
f5241b
3414fc
db2426
19e7d1
db0115
d9c69a
ed0434
1de4f0
1d1428
e6fafd
2e0227
beb9d8
07e5b4
dbf1eb
2624d1
040c10
0f1bef
f4080e
2c242a
```

```
e5e43f
00f7ff
c0addd
f200d6
eae4e9
fc15dd
000424
09e6db
f20104
fff010
192b12
1e1b11
f1da14
e8482e
f924dc
f6f4e5
ead5dd
eedb0c
070ff7
fbeded
f1e41b
2304f8
2ceef3
e8e11b
0b00eb
0ad1df
faff12
090f0f
17f006
f10ff2
0eeb18
13210d
f603e5
010eea
0b00e4
f40000
e0160d
16060a
0bf6fa
0c05f3
cef208
f0180c
15e4d8
1006fd
1b1af4
160810
10101e
01e6db
e1f81d
f6f207
f8f2f0
04eef7
061bf7
eb01ea
fbf61c
13fd00
f7ffe7
12f50d
```

```
1a220c
504c14
f2e740
0c131b
cbe0e9
f3d5d8
27fd19
ead50f
1a090b
0cf6d2
e6293c
d0e4e9
2900d4
5c452d
3039f7
082242
ffe814
0916e3
0d2af7
0016e9
ff25f6
f1f1ee
cfda1f
f4f5c5
ee0ee8
f4ecd7
f9f5fa
2d0508
2a2638
08141b
f1fbfa
d4dbca
fce3e7
f0fddd
21fc12
1b1317
1a330a
d3f70e
e5cbca
eeeb00
eee7f0
e2e50f
0c07f5
f820e7
251c10
051a35
e9fa09
d1d5f5
e6ead9
1e190a
e0e4fd
1b1bea
faeef6
02251a
e0f7df
1c340c
e6cc13
090d0a
```

```
fd121f
ebc3fb
c6172a
ddcfb1
fb0dfe
17e6d8
ecef00
ed0bd4
ff03f4
f702eb
d702f5
0f0c0b
ead0e9
1219fd
0a08f8
0111fc
06f91a
27feee
ed0626
f2f204
fbf023
1efae4
edfcf7
1bee12
1f16e9
fb0709
1e1cfa
04f1fe
1c1912
06f418
fdfefd
f308f4
fb0c1d
fa12fa
211500
d9d0eb
11ffe8
0a03d9
131d12
0cd9da
e80116
c9fd01
22bae7
342817
462706
464744
e90ece
b52007
b2b5dc
e3eece
c3e4c4
2a2be8
f12e0f
1b2935
ded43a
1f04ea
0e0606
f92213
```

```
f5eed9
f300e5
f6d303
241f1f
1a0405
1bfc1b
d40808
0be6f1
f5cd09
050517
d9e9ef
0e0ac3
e51913
222028
f81a32
f3f5f8
fd0406
cfe5cb
f2f11d
3af706
33261a
0d0334
11f8ef
e6d3d0
e1e5c7
0beb00
e8eb04
f5c0ef
041102
f9141e
16150d
d1fd16
f6f8f9
000502
190d03
dfdcfb
1fe60e
fced00
4725c3
e0f4fd
1c3e2f
d0e2f3
f201c7
efb997
810c04
0b17cd
ee9702
000645
28c9c1
102631
eeed2a
f529d9
01ebd9
d20c19
2cf4e1
ec0008
20070c
df0111
```

```
fc1628
27d50b
dc0bef
fc0a07
0e0bf4
0c1514
29f303
e50121
fbf2e8
f3091c
020e13
1debf1
d70706
1df4ed
e3f6fe
1306ea
190403
d9e819
160ffb
02d7d5
e4de0d
30e80e
edfaec
0930fc
0c0213
1a3216
fa0ae9
270ef5
dae616
e40911
fb0d34
2fef05
eeb509
f3013a
e0120e
08dff4
10e409
cad301
ed0fdf
ddd6ac
cde30f
0114e1
e4ebff
07f2e2
f71216
fdeed9
dd0d16
08fe18
1d02ea
d6ef0e
022401
ff1802
1b0207
ebf50a
031114
0817d3
e8e9f2
0d03f4
```

```
15f2f5
0c1e19
e0eeff
fd0012
1a17f4
ff05fc
0b1ffb
00e8e7
f1ff13
ef0504
ee0708
00dfed
1b0afb
d60bf8
f21d04
2b0aea
cfe011
26fef1
e803fa
2a15f9
e2dd3e
0fd1d5
c803e0
cdd9c9
fbe2cd
f2fc2e
0222f7
12cfee
dc493c
1014c7
c5bff4
29270a
01dacc
e71c00
ec0909
090ae5
10f619
ff0f1c
032115
09fef1
df26f4
fee4b2
0010e6
0606e3
0f1202
f8efdc
3d42fe
d2fa1d
060905
dcb503
16e9e0
e1071e
021cef
ed271a
f506fd
120e0c
07b9db
e01418
```

```
de0dcd
e3ef01
0d21e0
fb140e
131227
1f27fd
10eed3
d7ff10
eae0e4
0000a3
000081
00001d
00005c
0000bf
0000fd
000043
0000ce
00009b
00008f
000091
0000dd
000096
00003f
000001
0000de
```

Listing 23: Packed FC1 weights and biases for three-lane FPGA ROM loading.

```
020a0d
f00cfc
f6faf3
000804
ed02f4
f7f800
0debf5
0512ee
07f8f7
04fcee
f4f100
0206fa
f80bee
04f7ed
f2fa07
000b05
010306
fe05fb
f200fb
ec0300
f305f0
0aebf5
edf805
f60010
fcf8f9
08fb0b
0a01f8
11020f
0e00ff
ef00fa
```

```
f3f307
f105f1
fbf308
f10c00
f105fd
faf108
01fa0f
05ecf3
06f70d
050cf9
fa080a
ee0e05
eef40c
060af8
0d07ef
f31006
080201
edf203
f2f5f8
ff000a
000af4
fa08f2
01fdf6
0e010b
07f500
fe1109
f9fc06
0b0af4
effbfe
f9ffee
ecfeeb
f8f103
fafdf6
0df5f9
f8140b
070cef
09f3f0
0a0e08
07120d
0e0e01
08ee07
05ea10
f40df5
050a07
fd10f8
efec04
fb0804
f1f002
0efbed
f500f2
070105
00f5ef
000102
f2f2f2
0ef0ee
fafeed
f707f0
02faec
```

```
f9f60c
0c02ed
f600f9
0205fb
f5f60c
eb0af4
f90cee
020aec
09f7f0
06f1ec
effef0
fd0006
0a0e0d
ef07f4
ee0afc
0b040d
0d0f04
0707f1
01eded
0cfa05
f8f903
050c01
f5fc03
f0020b
04f7ef
f7f70a
fff4f9
f4f304
09f200
0209f9
f7f5f5
f0ee08
fd070b
fdfcf2
f8eef7
f100f4
f3f7ed
f0fd09
f10309
f700f9
080a08
f2fcf8
090b01
0efd00
ec080b
0000f5
13fe17
e90a1c
f703e9
10e513
06eae8
00f4ee
100603
02fb09
e8f5f5
362e12
04f1f7
eb1c0f
```

```
f5fe05
df01e9
00fc00
15ff05
340716
0d3036
fb16c6
c5fe0b
f31118
e8dbfe
0dfe0f
08f113
2c1403
daebe4
f0d6f4
32dbdf
faf432
1905f6
ee0705
1df5f1
03fbfc
f1f100
f8fef2
1e0402
ec1f27
24160e
03fa25
f80df3
ff0f12
4407e8
e51630
131bdf
e4e602
f411fa
e50c25
152603
fdf42e
08f503
eefe25
0408f7
ecdf0c
1708f4
f3db13
00e701
ffeefa
10d3e4
fd2719
dcedf3
11fc02
0afa20
1c1e2e
000a03
05ee06
03090e
3f07ec
f20634
262216
d3f62a
```

```
f2090b
f3e100
f802ff
08df12
29f1e9
0fff05
ee0601
eff40a
0cfa09
1904fc
f90fff
2608fd
0ae304
070cf3
f00f2c
f00b30
f509f7
f804f5
fbf517
1dfff4
fef70a
f40cf7
2417e9
e9fb16
26ff00
fdf402
0df6f5
030c0b
1112f5
f407f8
1923ff
27190d
dcfe09
1fff01
12f616
f409ea
fbf30a
07edf3
ce06f4
0112ee
fbe8cc
d30bf8
0af4ec
ec0e0d
0014f6
0d131e
04fdf8
230700
040b31
1e0c01
1b03eb
10f603
0205ff
fb0108
13fef7
130f06
2ce218
f0f714
```

```
14f1f8
1c0fff
fcfdfc
f11215
12f200
000002
0eee08
ca0712
17e8e3
fadf0b
070bed
fcfe07
0d0c09
062118
e60102
192919
09e3e0
d00e10
02e2e1
00e802
f8f6ed
03f6fd
1500fa
e90c34
e701b2
d9eaed
f9dbee
0417e7
00f8f9
ea06e2
fedff1
f1f5eb
f7ea00
ecf702
f92128
33f305
ffec11
200515
f1c511
00fa00
eb06fb
100c0a
e7fd00
03061b
0501ef
12f80a
fe1e03
3329ee
fc053d
152ed3
d3ff1b
0103ed
fd18f3
fbfe08
ed05fe
13fcfb
e61f13
1908fa
```

```
0cd513
f2e712
10fae0
05ee00
0c0316
0a1215
091605
efe4ee
0412de
f4e71b
070d05
f7f9fc
f116fd
2e180f
0d0929
0cf808
17ea0c
e50603
cdeff0
14fcd8
f90c06
10fefa
dd0df9
01080b
ef0319
f1fdf6
feee03
efee04
06faec
23f4ff
131e04
2c1b0d
fd2135
f1f518
0202f3
02faf6
f8e9f5
ffece3
0c0bfa
e2faf8
1c16fb
daf800
191c10
05f9f1
f50af1
f805f6
0112f8
0ff514
221d09
081820
01f3fb
e6d7f1
01e606
ee02ef
07f1f5
2003fd
c2ec07
0e09f2
```

```
ebe1d9
f1211e
17f800
e6111c
12ff1a
01ff24
120609
000508
f60913
ff171b
f4f1ed
09ff04
0d0afe
05f115
0a0600
283309
01ea10
04f0d9
f9f412
09f9eb
d5f6ff
fafdf8
08f7f8
000005
f9faf7
e51c04
00f2f0
020911
e6fa0a
04faff
eafc19
0201ef
d70014
260011
1a01fd
190703
190031
1b13ff
0d1007
100d07
f303ef
05f3ff
0eed20
e7f4f6
01f316
fc04e5
090104
f41200
ede5ed
b8ec11
0dd9d4
ebe008
1b0515
0807e6
e60812
09212d
2ffc17
0d1300
```

```
e5f916
15f4f3
f10f06
00150d
0ffc0e
e7fffc
042004
0e00fe
221930
0f080e
e40217
0fe7f0
f52c2c
fc1bf6
0ef101
e7e9f6
0c122c
e92507
f9d5d5
f3dd07
ece8ea
ced2f2
e2f9d5
d2ebd7
1b08fd
160bfd
16ded6
eb1d34
160e16
1004d1
fb03f6
07f801
000b13
16f1fa
05cc35
042634
e11b0a
ecf6f4
f4ebdf
fdedee
d4eaeb
0305fb
140720
f70905
fd0317
02fff1
0b00ed
11e9fd
0809f7
27c604
08333e
0119fc
f41511
08f405
fefc06
d6ff0b
fd0004
00d0ee
```

```
140a01
effd11
120d16
171402
00f50a
010b05
03f9f7
f11316
0f11f8
161323
00ee03
11f721
ff09fa
eaf803
e2fbd5
fbfc06
dbf5ea
0d00fc
fafffe
09f000
09f117
0aeafb
010002
1c0300
fd1003
f30213
f70907
1b0e24
1e0003
f8fa01
00fe1a
0c17f4
09060a
1126cf
0af22c
fbdb09
f416fa
0a0ff1
dfeb1a
09f004
de0009
0000dc
160000
ebf00c
170ae3
e5ef13
090cec
f3f700
fb0a0d
08f6f9
0a01e0
ed0312
f3f108
d1ec18
f2dfd3
edf4ed
e704ec
14e8e1
```

```
361e09
ef092a
f521e9
dafe07
05fcf2
feddfc
1e0e01
0d022a
2b2b18
4f14e9
fe102b
2c5507
f9111c
112730
05230b
15f304
e24923
ec00d6
eb0604
0ff3e9
e5df12
0d000e
06f2ea
ffec05
ed07f0
1dccee
eadaff
0127e9
e5e9ff
e60c10
fceef6
08fdfa
16ed27
290f11
f61608
0febfd
182a28
14031c
03181d
f60c00
150712
03e00f
f1e533
f5eaf6
0d1009
f8f4f7
f8010a
18f910
0f0201
db1636
12f6e2
fd1d19
1d21fa
f4eaff
e7eaf8
e6e1e9
00f4f0
f01806
```

```
292bed
f3120d
fbe2fa
11f504
0bf3ff
0d1116
1607e2
fa1c28
02f804
0b0d00
e0070a
03001e
10f0f4
e8e1e8
02f813
11170d
10f8ff
100421
fb0b08
fdece9
f614f3
01faf9
fbfc0b
f3f5fc
fd03ed
f9f0fd
110dfd
1104f5
f4f605
fe01f8
120103
f60006
051415
1bfae9
000d28
060900
ecd7f1
f81b18
14e2da
fe0a0d
0c0903
1cf114
f9ed11
021508
eceff8
e10d03
ffeae9
f1dff4
eb140a
f709df
04ea0a
160029
030902
0004fa
e0fbf9
160bfa
0ffce1
eb0b14
```

```
1a1def
000011
f400eb
fdf902
e9e4f3
0afbf2
00fbfc
f5ec0e
00f407
060ef8
ece7f4
f000f5
040a00
07eff6
0b0d06
0e0d05
f00309
f8f400
02ec01
ff08ff
080a06
f5f70f
feee01
fafc09
edeff3
02fb09
040006
0ef40b
ed0f03
edf8fd
0604f3
05fbeb
f7f303
eff2f6
05eced
fa0fe6
f60cee
f4f7ee
f80af3
f402f9
0cf6ee
010a03
f50aff
0fed02
07faf9
f2fd00
f40d07
e9f7fc
f400f4
f0fe02
f1f5f2
ebf8f3
eefff1
eaeeec
f70905
080304
fc0ffc
f90a0b
```

```
07ebfc
f5f7ed
f505f6
05f2fc
0407fb
01f9fd
f4f310
09f3eb
0908f8
02040c
06f5f3
04edf8
e4ee07
eefde3
f0fbeb
04ebfc
ebffed
f40a0e
0500f8
fa08f8
060200
0dfaea
f7f500
f10703
000706
080d0b
08ffef
06fa06
0001fa
00f7f9
fcf3f7
f5ebf0
f8ffed
05fd00
0808f4
01eff2
09fcfb
09f201
fe00ec
ff02fe
fe0e0c
effcf0
f7f50a
000cee
f0f7ea
f8fc05
03f203
0afc0b
0ef301
00f603
faf3f6
f50f03
0001ed
ec00f8
0aea0c
0507f8
fbf700
edfeff
```

```
ebf608
0cf90e
f4f5f5
03f904
00fdee
ebee06
ef0cfe
040b00
0e0af5
fe0000
090904
f60300
e5f900
f3f6f2
f60810
fb00ed
f2f5f3
0ffef3
fd08ea
0000f4
ea1104
0bfa0b
090f02
1e11fe
11f005
ed0bef
e8f2e4
eaf5dc
2a1c00
00eff6
000714
19fc02
ec0001
feff09
f2fc0b
fdecdb
0bf609
f1e3f5
f8ecfa
060110
07ddf3
0ef21e
0bf807
f216fa
05fef6
4c2220
273c2d
21451a
180bf6
09f104
1128f7
0b03fc
171a40
14f30a
241405
f102fa
18020d
0c0c00
```

```
fa071b
f3081b
0912f4
03dae8
eff2e2
d8ec15
00f80a
f4fef9
0ef4fd
19f0f4
f2fe10
f80e12
08090c
f30618
e0f2de
fce203
e90107
24e5e5
e7f0f8
d406cd
06fbf0
1f13f1
0d0403
f70f00
230be9
02f725
e7dbef
beee06
eee9c5
ecedfa
f208ff
fff3e0
2702d7
dbdffe
dbe4ef
d715e4
e8f9ee
01fe09
f8fdf9
0803f3
28131b
101317
0603f3
cd081b
f500ea
fa0cf1
f8e8e3
e9edd4
0c0e16
08f816
0a1a06
0e1915
fc05e4
1bfe00
f50811
1613ee
1409ef
000d06
```

```
1a0b03
0a02f6
f8fb07
020ff7
f00003
ea06e7
f613e2
09f6fd
fe000f
15240c
cf0015
d3e2ea
1012df
15e7f2
122122
0af9f0
051110
3307fb
f7080d
1ce603
05091c
160e08
1309fb
f4f1e2
04050d
fd1107
01f8ff
1001fd
e4110e
f31d1f
e832e3
f12509
14eefe
f3faec
042110
eaf521
160800
0000f1
15fa02
df04f5
ededf7
090be9
f1fef8
04f2e5
f8000d
0cf508
f5ef08
1908fe
12f103
110912
0efd02
0e08eb
fd0611
ebee0e
0df8f6
f7effb
1010e9
fff90f
```

```
060007
fd37ff
ee0009
0e10d4
ddd1e9
a7eef9
00cfc5
c5c4fb
2a0b34
2201f3
02eb17
0d0e25
1aec08
f8ff08
f70311
0eef00
f0ed00
e5090d
18f0f1
ebed00
0a0a16
f4000f
0ff5fc
0ffbe3
1efef8
1bf5e7
f41211
d9f91d
00ebcd
e7e2ec
fd0f04
022416
d9f0f5
ee10e7
f4edf5
becefc
030ed8
eb0fe6
1208ed
0008f6
1104e0
f1e607
f8eb2c
000de8
dcfed7
0d1c00
0beefc
fb1703
f0e808
f3170c
f33009
fa0cf8
efec00
0dfbfd
c3c909
0c07fd
e8111c
ee14f8
```

```
e21301
00ed08
18efee
08f1f4
08edf7
27def7
071b1b
0cfe0f
e71602
00effa
f70beb
e4fbf0
00ebe6
05d4d5
02f0fe
f4eb01
090c16
110aff
f1041a
050e08
f4ff03
f605f6
2a2419
e51831
06df00
f4f201
e7eae6
fff50d
0300ff
e9f5ef
d7e7ec
f0f7e9
f80bf0
03f40a
26e807
04e615
0108fc
0e10fa
ff0b09
ed08f9
fd0010
f80018
0b09e8
fef9f3
02f700
02f9eb
17fc0f
f909f8
efebea
12e404
f9fc04
e502fd
fc0719
e6e00c
02f5eb
0000f9
13190a
e2f603
```

```
e8ffda
02ee00
f0f0fe
ebfacb
f41822
0a2a1d
fef608
092125
1e0502
de0321
04e3d6
dbd2e4
0dfdfd
1bfde8
331220
120c1c
f80eea
e40010
ddf612
dc02f7
f0eae6
f7f102
f6e8e7
0ffe0e
f3f212
123fdf
fb0a0a
00112e
1c1e10
17fdec
f61030
090800
fd0bf9
0df4ec
0cec00
f92100
1b0af2
f4f707
0c1f01
24fbf2
001410
2d0bf8
f1051f
dbf713
effcec
f5f6fe
d9e3fa
0fe8df
080608
0f09ea
00e800
2632fb
00f303
fbe820
f20afc
12e6c4
0e081a
eaecfd
```

```
edfa04
0be602
0b2134
f31ffe
04fbf5
071506
0b040f
edf70e
1a1aff
eaf10e
f6020b
fde6e0
0ecfc7
ddcc0e
0efafb
19140d
ebfeff
f0eded
f6eef8
000c04
18e7e2
21071b
11031f
2e0d0b
f6072a
161724
01070f
eedbeb
ebf7f2
00fbed
edff06
04f814
0c1d0e
f4f5ff
160201
f105fc
dc0201
fd110d
0ff6fb
0cfcfe
1c2d0a
0df31f
04e2f1
e9f3f3
eddcca
eee7f2
dc19f3
24fdf2
de1a2a
f7080b
0402f1
e01310
15efef
f5e80f
070516
212b0f
0bf205
0c08f9
```

```
06f906
121304
130ff6
f8e3ef
0307f3
0410eb
0d030e
290701
ff0a19
010ced
c20513
ff05f7
00e0f0
d407fa
0912f0
000019
f4ee09
08ef0b
1213f4
001e03
1af913
ec1616
f3fcfe
fed1da
ff030b
00ff0c
efed0f
100405
f3f310
0e1200
ed0bfb
f406fd
f711f9
eee0de
fbded0
e2051b
1018fe
25cdd8
f51c2f
0b1600
f70127
fbfd0a
130a18
1507f7
fa070c
361cfb
1ef3fa
09293a
526304
0900e9
e3ef04
0cf909
fbfe10
0900e6
fcf7f9
e3f70c
fcece8
fb00ef
```

```
efecf7
05fdf2
f3d8ff
0009ea
0c03ee
1b2b18
06fef8
eff61d
1109fd
1705fb
d0e3d7
ddc1f1
f2f724
18f3f7
32fae6
d8043a
0f142f
0401f9
27f815
e00911
0004e8
13f6ec
dadbf7
ea1b16
101be7
fdef11
d8daf5
0fecf8
f608e0
fcdfcc
d52329
2936c0
f0f603
fffeed
f50701
f8ff1a
08f505
f9ff10
e5ea05
f40a05
f5e8d5
e9f5e8
f7f909
ebebda
e9e7e0
f82812
210c06
f70d20
f5e4f4
fae6f6
09fcf8
0c0103
110cfb
0d1006
fcf20f
04f5e3
00f80d
f10bed
```

```
0efcf2
ea0afc
e2dde6
060400
13ff1a
062813
f7e1e5
d5fae7
e6e3fa
0601fd
01e511
2708fc
fe02fb
18220d
1907dc
f2c808
0103f7
0f0007
041700
05180f
12f7ff
fe0002
00f5f9
fef2eb
02f3f2
140deb
00f4ee
edd3f5
070d01
04ebdd
f01c29
0509fd
111013
000000
01fe10
f10c00
04fff1
f70ff4
01f4ed
0401ef
07fbeb
eee5ff
fcfe04
fdf9f2
0708f5
00f6eb
04f1ec
10fe03
00ff0b
efec01
f7fa04
f50bf5
0f01e9
0014f3
f6ee01
e4f40c
f601ed
0009ff
```

```
fd0908
01f9ef
f7fcf9
08f5f3
11e70c
0bf500
01feeb
fbf105
e7ecfd
0509fe
fbedea
e90709
f9f1f5
efe700
000307
08f1f2
f0f5f5
e409f3
f004e5
f3f4f5
00f402
fcef08
e60700
06f701
00f4eb
f9ffee
fc0dff
f8fc08
f1f902
030ae8
e9040f
fbede5
fd0305
08f1fe
00050a
f1f808
05fbeb
eff6f5
f412fb
fe000b
0401ee
f8fffd
fcf3fd
f70301
f707f1
0a0a05
f7f5f1
fd0d14
0deb02
e9f4ff
01ece9
edfb00
f20a04
ffeb02
f3f205
fdfe0e
f2e705
f0f0fe
```

```
eafaec
f6e902
f3100e
0ef504
0bfaf5
01f400
fdf1fc
0803fd
ecf8f0
f4f8f0
fdfded
04f8f9
08f9fa
ec04f1
ebeafc
f4fdf3
e6f3fd
ebf9f7
e3e5ed
02ede6
08f9fd
f6fdfe
fdfbee
f8eee7
f7eff8
06f700
ff0dfc
e7f6fb
02f4f1
0efce5
f104f7
f503fe
f0f7fe
e809f4
0ce6fa
ff04e7
02f0f8
faf0f0
eaea0c
02f7ee
000a00
050800
040206
07f4eb
fb0013
fbfd01
0df1f6
f2eb0a
fa0704
f205f7
ec020b
00000d
f8ef05
12fa0e
10f1f4
f21101
f90d0f
fb0e00
```

```
0d0411
0d0bf6
0f0afd
0f0fed
fcfb04
05f5f6
01000a
10f702
fa0af1
f9fced
f90fed
0d0c11
f10fee
f20000
f708f0
03ed09
f501ee
f700ef
f30d00
02f900
10f512
f606ef
f8fc00
ff09f9
01f309
f5faf9
1102ff
01fc00
01f0f1
f4f4fd
f50e05
10fd02
fdfdf9
0500f9
fe0d01
ed0505
0706f4
fef210
04f4fe
ee040e
1000fd
10f0f6
fb0e0b
f3f6ec
eef4fd
050ff2
02f2f2
f1f707
0310fa
02f210
f909fc
ee030b
010607
05eef7
fe05f3
10eff1
f7f4f2
00f1ee
```

```
fbf307
0c10ee
f3f70f
fd0206
fa02f2
05faf7
0b030d
0600f2
f2f10e
ff0af0
f7ecfb
fd00fb
f90a05
f70702
f10bf0
eef609
fdf700
f60ff2
f00410
0e0603
060df4
07f5fd
f20e0b
080ff6
ed0dfb
eeeff8
f9ff0d
ecedfa
f9f7ed
00f8fa
07ef0c
fb06fe
edf6ee
ecfcf0
07eff5
030900
0c0801
fdf8fc
f5feed
0eeff9
f607f3
f0f506
04f0f7
0becfd
f600f1
08ed07
01fbfe
05f0f7
03f702
f10d0f
03f6ef
f90df2
fef110
04f506
f90f08
10f209
f40c0d
ee0800
```

```
00edf6
07f00d
0807f4
ff1105
f600f5
04fe0d
ff10f7
0d01ef
12f00d
fff5f8
000c00
0000ff
eef002
0712fa
fe090b
2a1dfd
0ef105
f90d0a
ddf001
0c0700
0eff0a
00def3
021d28
29f6e0
0a2830
120509
01f716
0efb05
f4ee16
eedebf
f9d31a
23fbe6
f10407
0307e8
fafff5
f006f4
fedaeb
fd1129
250f22
f60404
23f9eb
03c60c
fd2d0f
142305
18070c
1b0c17
0524fe
0e1712
050e04
fd0dfc
faee17
ebe8fe
15f9f3
0d07f9
0b0f05
efe42a
230ee9
0002de
```

```
13321e
2727f1
f4000d
f70407
060804
eb0200
e5e8eb
e6cf0d
f400f8
f3dff0
f4edd2
f400d3
ef08fa
281c03
0afc13
df2f12
1100fb
0403f6
f800f2
e8feec
f80afb
f9f2d2
d4e218
1efdff
01fade
05f4f3
000014
fd06fd
e4f008
fdf600
311709
f7fb0c
250302
0efcf2
1209f8
dde5f8
fdf7ee
fad7f4
0704e7
03e7c9
0d11ec
041e25
1e0729
081605
0cfee6
21fafd
f5e50d
1c36f5
01f1fc
011116
16190a
fa10f6
03fb08
f61618
0aedf2
f90af5
080800
280b16
```

```
10210e
0717f7
dceef6
fdf5eb
160305
0ff7f9
10261f
280608
fcfc31
2b07f5
060f0e
23f8d8
170817
f4ef15
f614fa
01fdf5
1e0d20
091718
051718
faf4ff
fe0c0f
fa1adb
c824f9
edf00f
253114
11fff9
f40006
0cefe2
ec02fb
0000e5
ff1503
1bf4f3
f31c31
1b0000
120c10
010fe2
ed03ea
1a0afb
211a07
f3e20b
0c2a30
2af900
fe2621
15f9fe
0811ff
f60b04
f6f3fa
0dd5d2
00f8e4
dfecdb
040200
0c00f7
1b0202
fd0c23
fceddf
161b23
0e252e
321a11
```

```
28edf2
dfe50d
ee4710
08f4f1
120b13
1e1211
2125f5
fcf917
23350d
fa1a05
f7fafc
d9e601
170b0d
0cf2f1
dd02e2
dcdf11
06e4d8
f70af2
0c2d13
291300
e2e138
f808f9
05fe01
f90802
e50deb
ee13eb
dbdd02
e4d4f8
b7ecc7
d90ecd
e80406
0c231c
01f11a
051007
070f16
fa0112
08ebea
e2fde9
0c04ed
e6fa01
10fdf6
d5e611
070500
e4edf2
04dcf0
f7d9fe
f30af7
010905
3108fc
070826
0f0016
1115fd
0dee00
100c10
fd0000
e1eaeb
f5fcee
0b00f0
```

```
2dfdfb
190814
130430
0c1c0d
f3050d
0ff608
041109
21351b
0cf619
040f1c
fa1a17
11fe0e
120400
fc1107
03f906
fbfefc
fa2010
230710
0d1812
13f0eb
fb1d11
070301
372df3
f5f600
272206
fe040e
0f0803
1cf90d
181d08
192901
16f00b
09010c
110522
0e1003
191105
1c1018
f40815
f20dfd
f2ef17
dff005
f31fe6
ec06fe
34351e
022601
000204
02e706
040204
0000e6
0a0007
f3f617
02fdf2
f6ff07
07ea11
fafef7
fb0ef3
03f6ea
f00813
09180d
```

```
f1e0d5
fa0614
fbfbe4
fdfcf5
0af8f6
19fd04
1a0301
0d3b45
0709b5
b4f606
11ff09
09f208
f92506
1c1e1d
13130d
00e7dc
f3f8eb
00f5ec
d6e4f9
ff0119
1de1e3
0f0002
fa0dee
fbf9d9
f0dfed
ed08f9
03f50c
130600
ebfef4
160a12
f40104
2b1502
f10114
fd2dc2
d9cf13
ec1520
17fefc
f90b1c
160af6
060909
ede7ec
fdeee3
faf507
100bf1
191a31
2d3214
17191b
2ded1d
eef91b
d5d4df
080912
0ff3f6
081bed
fd1200
0ffd16
f711fd
181e12
e21106
```

```
061edd
e3f41d
31201e
28f0f1
0e1828
1814fb
2d1013
fceff4
dbe004
0607f0
15f7f1
0c0006
e10d10
0e18f5
00ea0d
291f18
e6f214
07150f
00efea
eb0309
0e0bfc
0ef204
f01a1b
0126fc
080d0c
07e1fe
0512f6
0705eb
fe000b
03f7fc
0a0308
fa0805
ff10fa
fa0204
d4f0ee
1a04cd
0c082a
120703
fff608
fa1903
f2f01b
ff00ed
f7dce4
da1509
dbfbe9
f3dffa
0906f6
ede507
13fb01
0b000e
f2eb00
eafb00
e2faf4
10fd10
140a11
140417
1402e5
03050b
```

```
0fc0d6
faf5e3
0e01ea
0c0cfd
1624fb
042102
f4f106
000028
f91a06
041f13
eff1fd
f5ea0b
f3f90d
1cf9ea
2801f2
000e28
fefaf6
1dff15
2418fd
100908
fb080e
130b08
e6ec18
ff1516
fad9ef
0afefc
01f1da
f807ed
01eee9
e50a00
04dbe9
eef500
14fcfb
e4170a
0dd9f9
1ec9f0
ed0b0f
150916
12d7f8
eb1df4
d1c2e8
0f19fc
0f06f2
011a04
0816de
fbf118
fa281c
1219f6
00ff18
e70811
fcebfd
f0f8e4
f30808
110905
e90d0b
09f906
f3e9fb
100be0
```

```
0a0700
111818
fff520
d401d8
d8fd08
06f0d1
a3f712
ed2701
130b04
f2f1f9
df0106
1afdd0
1afef7
ed0df2
1409fe
3407dc
022523
0514f6
17fb18
e70000
f7eaed
d8dffd
efe5e4
fde00d
0e1124
160007
1b270e
efff0a
0affea
06d601
0600ff
1003f1
24191f
03f6f1
2013fe
ff07f0
f90702
f8e7ff
ebeb10
f8ecff
ef0521
0af5f6
060c02
22130d
100622
f81111
f6f40c
080700
1f2af6
fbfe14
230a00
040e2c
0a190b
f5faf9
e1cfea
0006f9
feede7
fe111d
```

```
fe030c
16f515
fb0d05
ed0b05
0e161a
3205ff
f6f615
122e1b
fdfc0e
0203fa
14141d
26f103
000d0f
0f10f8
feebf7
ef0f06
e60aff
18fede
fbfc09
111708
f5d902
12ea07
e5f5ec
ee08ef
010107
000001
07ede6
f1f700
ff020c
f50aeb
e2ff1d
07f400
d3020f
fbf1e8
0601ee
edeceb
04000c
fef1f7
0d151c
fbf4ee
181801
08f3eb
fc0414
1c02eb
021754
3f090a
ebf303
f11d02
14ef00
fafe03
090603
f4fbfc
ee1af1
db0802
1a04f2
deffff
f72210
08e400
```

```
f2f4e5
f61000
f300fe
fcf9fa
e5f500
f10bfd
1d0fff
0700f0
ee0ff6
e4f405
271c0f
03072a
281725
e6f714
d9ef08
02eaf7
250cff
000d18
fc00f5
df0ae5
271efd
3e1712
3e2608
e4fb24
d62bfc
12e4e8
f600ed
f1fceb
0f04de
fd0220
f50110
020d08
081bfb
06e4d2
cdf7f6
0401c7
1ffe12
2e3838
121128
083935
f5e5dc
e9e228
f411e8
00f6f8
04e7f3
f6f205
e91609
f81706
22fee7
00eaea
020ff9
06dd0d
15010c
fffc15
0e1100
f700ea
e3ddea
01f0f1
```

```
000a05
0df311
ecefed
0a0df9
e10705
fd0a0d
f1f813
181611
fc0309
050415
e60ee2
03e4fa
15f809
2220fa
101f09
00e718
f60813
1a0af7
09f210
f50b15
020fee
fce2f9
0e0cee
e3f7f5
071e06
0d06f7
0af006
08faf5
e5f5f6
f40004
00f5f7
010701
01f910
040b01
1500f2
fdf4df
ff0a02
17ed03
f2160f
eb1701
f8f5fb
10ec07
0e0001
000009
0004eb
edef05
f202f6
090b05
07eaee
e60513
01f7ef
fbfdf6
ef0205
09070c
e9fce2
0ce900
f903ec
e2fb07
```

```
e8fdf3
0008f5
05e401
f70302
f4f9e9
f4f9ff
ee00ff
ef0c08
030f05
ff0909
00edec
f6fc02
e8e8e4
ea06fa
0ce2f2
f6fe0c
fbe3f9
f5e50c
f7f6ef
000cf4
e1fdf2
f6f7f6
07f800
f3f702
f6faef
0be3e4
e6edfb
f003fe
080ff9
0105ec
eff2f6
010004
ed0a0e
07f5fe
06fbef
f1f7fd
e1f4ec
dbeae6
0416ef
06ee0b
feeefb
fd0b01
0001f3
1b0e11
f10cf7
f5f5f5
f8eef1
effdfd
fdf803
fb0700
10ebe4
f8e9eb
f308ee
f2eae9
16ea05
0b04f9
04ebf3
f409ed
```

```
10f8f4
08ecf1
020fff
f400ef
eae604
05eb09
020002
fd04f6
e70001
00e9e5
ff02ee
14040d
f8f001
e804ea
f8eb04
f8fcf3
010405
f9e6ef
f5fcfb
ec04ff
ebf108
f5f8fb
e9e5f4
e6f4f1
f105ef
eaf5e9
ef08ed
f2e0e9
07e903
0300fb
f90107
02edfd
fee801
fa00ea
fcf2f1
06f1f9
ebf80a
f10bfe
0207e6
fcece9
f4e1e5
0a06e0
e8f3fa
f60510
f9e5dc
06ebf3
00f7e8
e7e5f0
e5edff
0006e7
f2e9eb
edf8f5
04e2fe
140f12
02fbfe
f6fdfb
fd00f5
11fdf8
```

```
fbf7f0
00dfec
0e10fa
0000f5
f0110d
0100ff
04f904
f90000
eb07fe
fe122e
0deff5
00f600
1011f2
eafae4
f20511
2005e3
05f2eb
152d07
fd0809
f10206
e7f1ee
fffcf0
ede129
372711
ffe7ee
2a1625
fefe0b
2019cf
f60706
3f19fb
171719
ed2e24
1019fe
101e00
0900fb
f8382d
113631
fbe826
17090a
0ff807
02f100
ecf601
f30307
1a080f
0c02ee
05010b
12fd0e
0ee81b
2e210d
eff7e6
04e400
ece6e5
0e0bd9
f11603
0a12f2
f7fe06
17281a
d9ed0a
```

```
0114e8
1828eb
52eb10
f72042
fdf0f4
090113
ee2409
f4e3e3
e4e9ed
14ec0c
db1119
02f410
e6d8f8
eff1e4
e0e206
2419e8
0ee6d6
161512
0a081d
0c53d5
aee8fe
04fcf3
fb0000
0a02fb
00040c
ecfa17
000821
f60305
000900
f214fe
01fa0e
0304fc
04031a
f1edec
1413f6
03f300
02f7ee
1ee1fd
f5f8fc
f50e01
100efc
0df220
2218f8
fff7f4
0ce900
01f9f3
fa0802
de01fc
0e0a02
fbe50d
00f6f3
00152c
0d08fe
0206ff
0f0fd1
12eff5
fd0414
58f706
```

```
101025
1440f6
07e9fc
0de90f
0300f7
0412f5
1113f3
e9ff01
e8f8f3
e303fc
f60f0a
0af0fb
e7091b
fe1c16
e50720
16f814
eff000
f0f203
0d0408
19f412
00ff08
0000fd
0908e4
f6e900
080100
00f9e8
ef01ff
f6f107
fdf80b
25f900
f711fe
fd0c00
f6eaf4
fdf500
fe01d3
e70204
fa0b00
13ff06
151021
05071e
08251f
0edb0a
18111f
06fdfe
133026
201bfc
f6fd0b
2ffdf3
ef0a44
f41d0c
140a02
001af0
140bed
f4fc17
f52c06
f6e709
f91715
0ffde2
```

```
d4f016
e71710
02e8d8
07f40f
f2fb06
01140a
20faf9
f71200
0e0ce3
15f000
1f001a
f1fd29
0d24e0
01081d
feeee5
0df5fd
002217
18210d
2200f5
f8252b
4c0b21
200740
f5d12d
f81202
211de7
fb0209
f0daf4
211806
1818f7
fb0b0e
feebfe
eaf300
080fdf
fcf2ff
fe0829
3003f0
081333
1c2328
270b18
ffeee0
e2eefc
0afef2
f9f6ed
fe1811
fdfe0e
fcf914
f22103
ee13ee
fffaf9
04ec1c
e50200
0c19fc
0300f4
d9f9fb
06f90b
f1f81f
fc07f1
00de0e
```

```
ebfaf9
05f3f7
f7ef19
1b16ff
fdf107
f906fc
ebf3e5
efee02
0eee0f
0c1df4
fa0af0
e20327
3a04f9
0cf407
f4f622
0cff07
0ae31e
140f20
1df8d3
da1309
1a23f9
01f00c
e5f704
fc09e7
e8dd0c
ed080b
10faf1
f9de02
000df3
07fafe
0ffe02
c7f301
190ee5
17d1f0
f5f71c
021712
12f105
10ecf3
0018f6
000009
f811fe
f20bf1
f900fd
f6feeb
04ed11
1306f6
0d0315
0f0015
fff408
1c1415
1401eb
f3fa01
fd0909
0f1710
0dfd04
f60315
00f201
ff3018
```

```
0bfff3
2afdff
fee902
f51f00
f200ff
f4f705
f7fde8
b2de08
fcd88e
d4d6e2
d21214
10de06
f0b802
102108
16b0d3
0d0efe
e400f9
f005ff
14e1e3
11ea16
f9fe00
ff15ef
fffe07
f730ff
13051f
1110f9
131722
15eb23
fe2313
e3ed1c
f3fdf3
05fe0d
0f1715
050ff3
e4f3fc
ff250a
0a0e01
faf709
0e1a24
250909
f6f404
effcfc
06eaf9
11ece0
eafe17
0e0806
16ec24
4c25f9
e61832
150efa
1210f9
0300fc
251a2d
2a211d
201604
3cfc18
0a0718
030d01
```

```
060017
ebeff7
ff0ce2
f7f510
f9fc10
0ff5ec
15fb14
0af71f
24130b
030801
080402
e6eeef
fb04e9
00f7eb
f1fa01
e70cf0
0909f5
fef00b
f406fd
0a0e07
f5eff4
030ffd
1304f7
07fdf6
1208f1
fe1a17
fe0600
0cec06
f0f505
090dfd
0ceaf3
f90806
d9ed0d
02fa17
f913ec
e3fb01
f80ef7
f1db00
fc0d11
f10509
1a0e06
0a0416
fef315
10f6fc
f5040b
1204f5
07f50f
080bf9
1cf504
1007f8
e5e303
061ff9
f6ffff
0ef9ef
04f0f1
0dfeee
06e908
00000f
```

```
fe1b03
0c1af5
13fe0b
f3f50f
130f0e
10240b
21faf4
01ed17
fe0202
0beff5
0d0719
1b2607
00fb06
051106
030a03
001917
eee4f9
15e3ca
d4ce20
2c1dea
0207dc
0cfe13
efd4f5
ff1bee
021217
272718
37fcf0
1f381e
ec0fec
15223f
1b02e5
ff25f2
e9f920
0dfd21
2b0e15
ea0803
29fce1
07ef15
fa0425
041d04
15fafa
b8f204
0f15da
dad529
161b01
080be6
19eff7
e8c7c2
f5f909
faf7f7
0605f3
1a120e
2a321e
ebe522
f80e00
10faf2
eedb23
eb061a
```

```
0ff9ea
2519f0
f6fd0a
0cf2c3
f612db
13f301
f50c22
09fbf9
e7ef16
110af4
f2f206
2a01e3
07cdce
f7ee23
e810f8
062bdf
ea0314
05f6f6
190bef
03fef6
1c010c
ebf313
f20f05
f1070f
01fafd
d3cd00
140ad7
07e5d5
fefb19
e4fa04
fe08ec
09fcea
f70af0
0eee0a
f210f8
1413f4
0bfd0d
ff0c18
fd17f4
f00106
fa0df8
fb0afe
fbe3ff
fc0518
212af0
d7d821
ebddf2
06f909
01f406
f8fd02
2419e2
11f9fd
021505
23050f
ed1913
251300
10f716
f10af7
```

```
ee090b
fe0d01
0c01f1
fe000c
f20703
fb1e11
080203
0400f1
ee1cfd
1febd6
bd0913
152ef8
e7def8
fff1ea
efe4e2
071100
f8e512
0000e9
0af512
f100f5
f7fbf2
f51eef
0907ed
fa060b
f8f701
f201f0
07fdff
04f2ef
070dfe
14ff11
110207
151afb
f3f4e7
f70909
f7f8e3
0e15f3
06fe28
38fc00
1e0406
321620
120c23
1104f5
cfedf5
eb03f7
0110c9
eff81d
0d19fa
072104
23f4ed
f63848
2e3108
fa091e
fcfc17
10ebec
f5f1f6
eef40f
f1e8e5
df0202
```

```
1b0de8
f50103
1afd02
09020e
1a0206
09fa03
0be913
000d14
07f7ed
f1fd07
111808
fc0014
f2f000
f8e5e6
0015ed
ff1122
4c09f6
0a1a3a
04e6f0
041815
020cd8
e501f8
f7e908
0cfff2
e4fff6
002619
08db01
fafd01
04ecfc
131e0e
f50afb
1ff2e1
022e25
2431c8
bcc7f7
1b00fa
0c171f
fe13f1
07f6f7
08fff6
08f719
dee802
fbf8f6
0a0a0c
fe1222
19fff5
070e0c
f91705
08fffa
f6e6f5
fce4df
04e7e1
f70801
ed04ff
faf813
13f50d
050318
fbf2fa
```

```
01daf2
0dfb06
0d1c16
fd0309
0af1f6
05f014
00e8e0
02151c
0003f6
f4f9f3
00f7d7
0a11ee
fb091c
23f716
1f08ff
fd110b
03faed
02ea05
f10819
0004ee
00f502
f7fd23
04f2fb
e4fd01
f70703
fa07ed
e1fd1f
061801
e3060b
14f302
efff17
ebecf8
f0fc17
0ff1f3
f7f803
0000fa
f104ff
fe1f01
ec05eb
0f0b19
fde114
faff0e
33f7f9
ebf308
e4e30b
2a08f7
f7fdf3
0c12fa
f6190a
001412
f2f20a
07160e
030df7
170306
f0f4eb
e616f6
cfddda
01f0dd
```

```
05faf2
f30702
1c05fb
daee1c
10d1ef
26cde5
f71501
16f6fb
f2f2f8
f0f80e
17dee1
12000d
da1203
e80f1b
262906
26130c
f5281f
1202f4
00ee00
ed08db
2322fd
f6f0f9
dded06
fbf6ea
051003
281d29
0f1d2a
faff1c
f1f513
1a06e4
dfe807
f8f5f7
e00521
f2d3de
d8f1f3
eceffd
1b15fd
fce705
130b23
1b0d1c
101119
06fdff
0ee103
f41500
12fcd7
fe0229
e8ea0f
fa0712
0c1100
cef706
02eefb
1701ee
02ea2b
01e8f5
051502
020024
0f140d
1a0af5
```

```
060312
25f5ef
eee901
eed9fa
0420fb
ecf5e6
e9f3f1
0b040d
0ffd1f
08fef8
010f1a
ea17f5
0df2ec
f5fe18
000106
040611
1afdea
0cfa0d
0d2108
eef30d
13efe8
0c0ffc
e008fd
1400ff
fdfc11
0208fe
13f1fc
08190b
00f714
10eafa
fdebd8
faf1fb
09fa0c
2e1c0d
fefe11
071414
f409ea
0a0015
fe121f
320e08
f90721
06ff15
130c0d
0518f7
0fe3e8
1fd8da
000409
f2f4fd
ffd9eb
15f8ee
021907
fffff2
fe0b15
000002
e7e000
0e08fb
110815
200d01
```

```
f80518
03f81f
fe1909
f6f0e3
f1f510
fc0c00
ee06f0
0b01fc
03021e
212502
0f170e
ecf4fe
fcf8ea
c1e806
0bf1d5
f4d9de
02fd0d
160cdb
ed142a
ed0dd6
cbe1df
e5efeb
f51707
dbc61b
1903f5
f1d3b0
f6e8f7
fb1f27
39efe2
0ff025
00fafc
000e03
e51505
09f00a
f80efe
edfc06
0a0011
f83126
ddf110
f6f9fe
09fbf4
0b02fb
180e21
fc0f0f
fb1adf
e4020d
09e4fb
eaff0a
d4e7e2
efd6e3
f2e4e9
03fb0e
2b0ad5
e80908
f400f0
0c0b12
16eeee
d50c0f
```

```
fcec09
05f4f2
e80f05
1e0aff
fbf01f
f5f601
f8dd03
fd0014
0b2809
26101a
043b25
3848f5
c2da15
fa0102
f714ff
f61318
122103
00fa15
f20305
fb0af2
faea05
0a040f
fa0a0a
00ede4
f51408
181007
0e1404
ef0522
ef05f4
f5f7d8
0be8f9
f30200
100ef4
1d0e1c
11fe19
f90efa
fefa1c
17120a
f5fa10
070e0a
f2e5fe
15edf8
f2152a
010b1f
170a06
d1ed11
f5d7ec
04fbfb
f0fe19
33150e
170013
d4f316
0d04ef
fec7e4
0a0806
09f105
0afdfd
00f614
```

```
fe12ee
ecff14
1df402
0ef522
10000b
f4f409
06fcde
10d7e9
0e060b
e6fee3
091121
1a04d7
f9edf4
0000fe
f3f2fa
0becf1
0f0bfa
000b0a
fcfd10
f30609
f60dfd
edf0fd
0b0c03
fb09f9
0eec06
0aff00
f5f902
0df4ed
f60000
000606
f503f7
f31203
f1ecfa
fdf103
0200ed
0cfcf7
ec0502
f4f1f2
0df4f1
080702
03fdf6
f0fd0c
f5ef0e
0e0cfb
08fe0f
01edf6
00090e
08ecfb
f70af5
eceffd
f9edf4
10f000
0001fe
05faec
090aed
00f4f9
fdf3ef
fafcf1
```

```
0103f9
f6edf8
0efaf9
040bf8
f509f5
faecf7
0004fd
0ff2ff
f20bf3
f0f900
ee01ed
0cf7ff
ee090c
eeedeb
f8f0f7
ed03f2
080603
0efdff
efeeec
0eefed
ef080e
030cf5
080000
0cf3e5
fb030e
faf10a
fef900
f10d00
f5f7e9
0d0dfb
06f5ed
ef0c07
eef9f2
0af30d
ecf5f2
0c0600
f7040b
ec0a0b
fbecfa
0000fb
01eeea
000901
fff5ec
09fdf4
0bea0f
fb040b
eef9f4
0506ef
fa06f2
00f30a
fffcf5
eeecf1
0c0e03
07f8fe
f0f2f4
fd0ced
0a0c0e
050f08
```

```
040605
f8f603
ec0603
f501f2
0dfa06
08f700
0efd10
ff0d06
0b0def
fbf3f4
09f00e
06f4ed
f91003
f801ec
eff903
f0edf5
fbf80a
0807f7
fb10fa
0cffee
f5f10d
f7f5f4
f7f509
ef060f
f80008
fbfb02
03ecf8
f202f9
03eeea
070b07
ef09f1
000001
0608f3
030107
00f0fc
0cfbed
0f0e08
0808fe
0f08f9
000cf5
f8f0f2
0d06f7
faf5f3
fc03ec
fd0800
0204f8
030807
02ee0c
02fefd
0ff500
f2fffe
f204f8
f20fef
ff0003
f1f10d
f802ee
f710fa
f6f20d
```

```
080ef8
11fc10
0005f8
01f701
f010ed
ff0af4
0f0109
f9090c
fb06fa
f9feef
0703ef
00f3f6
faf4ed
f200f0
06fe03
01fbed
030ef0
03f4ed
0a05f3
ecfa08
05eeef
f10601
0cf3ed
f2f108
fbf7f7
0006fe
050504
00fa00
f1f800
0ef10e
06080a
faf9fc
fafef8
0dfffc
06f200
0007f3
f0f9fe
f208f1
080fed
0bf5f3
060005
f60d07
f20cec
fcfeed
02ecf3
fe00f1
0ff60b
ed0703
f10bff
0e0305
ecf4fe
f20dfc
ebedf8
fc0c00
f3ee0c
fe0a0a
f7ef0d
07fb03
```

```
fdeff1
f805fa
fb00fc
fafffe
0af200
f9ec0a
ff01ff
f3fcf2
f90906
07f1fb
07f500
02efee
f8f7f0
0007ff
0e0af2
fd0af5
f3f80e
00fbf2
00ed0f
02ef04
fcf4f9
f3fd0f
efedf7
f600f6
f2faff
00f2ef
0d05f2
03fefa
fbfd0c
ef00ff
0300fe
f2ed04
f80007
fa0cf2
04efee
f10dfe
eff80a
f0fe08
eb0600
0000f9
f307ff
010af3
00faf6
0ef8ef
00f1f3
f90300
f70dff
edfef9
080d06
0000f0
040fec
0a00f8
08fd0a
f80cf3
0cf8fe
0d02ef
03f708
0901fa
```

```
0df6fb
f7f6f9
fd01f6
fbef08
ec0cec
eff0ed
050e03
f9f5f4
f8f9f3
11f9f8
0800f7
ff0ff8
ee070d
edf407
f4f70f
fcf8f3
fefafd
0ef8f5
f10af6
f0f0ff
0dee07
09f407
eefcf7
04f9f7
0000fa
f1f603
09010a
f20701
f8fe0e
07fbf5
ed0ff8
05fd04
fd0100
0207f0
0211f6
04faff
eff1f3
00f50d
fc01f4
fe0800
fb05f1
fe100b
02edf4
fefaf0
04effd
fff0ff
f008f8
f70cf5
0709fd
f607fd
03eefc
ec06f0
f707f6
03f110
f00af7
ef0904
09f4f5
f1f30a
```

```
ef0b00
05f407
0cf8f7
f4fe00
04f1f9
0800f8
040eec
fa0604
fefcf2
07060e
0e0907
eef9f6
f5f307
0d00fd
010ef3
fbfd04
ee02f9
0b0409
f5ffee
f1f9fb
ef010d
f4f9f9
fbfbf9
f7eff8
f800ec
f1ec08
f0edf5
f707f1
f106ed
fd030d
fa0df2
f6fb10
edf803
00f807
f3030b
fffc00
01ed0d
070af8
edeef4
0ffcfd
00ee0c
ec0bf0
08f101
f90000
0a0001
0fff09
10f9eb
0eef0e
0310f8
f3faee
03030f
0d0cf7
f3fd00
01f60e
0001fe
faeced
f500f2
10fafd
```

```
07040f
05ff03
faedf0
f9f703
050907
f3fd09
080df9
ff00ef
f6f909
0000f6
0fe6e9
e1ff18
0304ee
fa0a0a
0202f8
faed00
fc0402
0c06fe
d8e20f
232716
fbe7ca
f500ff
0e1604
07fd09
f50cfb
1209f7
210905
ec1910
0c14ef
e400f6
dce7fd
deeee2
1304f0
110a24
0f1bfe
f6e701
04f208
04d2dd
fefd26
24fded
dddf41
0bf207
fbd0ce
0be8f1
e3fc05
0ced12
0f161f
fe2713
270e1d
050701
d9090b
0bfef8
10151a
0f20e7
d1f600
f100f2
ef06ff
2a1a16
```

```
f1001d
07fc13
db0206
1800ec
ffec01
0c161d
0d0225
fae50a
d22011
0ddee4
f72012
ecf8ef
2ef203
021420
0b062e
f40fe6
110deb
ece204
20100f
fe0b0d
1ffd05
eb011f
06231e
d3fdf6
fdf6e7
0dc6fa
471016
06e3de
f6eb02
e90412
fc12f8
1dfdfc
0ef900
200cf3
e90711
fdfa21
fc0d12
f708f2
d802fd
14020e
0cf510
000a04
f91600
e31d0f
00fbf7
e1ff15
091100
1df8fd
f60907
fe1114
1d030d
12f904
1704f5
11f706
e2dc0c
1616f3
1a2a3a
e7e5f2
```

```
e70600
071011
c5fc15
000df0
04d1ed
ef0705
07fff0
06030a
f91c10
121f05
0c09fb
1ff607
fde70f
0b1afd
1f0604
04ebef
0e091b
1619fb
0ae5e8
0b14d0
ffef18
f2e2f7
21ecf9
1016f7
04fbf2
06fdfe
f0010a
0000f2
f8f208
e7ef07
02f902
fd0cfe
0bfc01
fef00a
fa0aff
0dece5
db0115
2215fd
01f2e1
f413f4
edf7e7
f20efb
090601
07f6f4
25131a
d50d11
0c04c4
d8f1f8
010e1a
f6f7f6
e00d16
f3fcf3
e9f60e
e2d6e0
e9040d
00f1f2
08000e
1b06e2
```

```
e9f8f0
011422
143713
f1f007
e60ff9
0dfd01
e51928
170ff0
f6eaf8
0f08fc
091109
2af6ee
eafe19
1117e5
e5ddf9
14fa15
fd1417
00181d
0f00f0
f3ee0e
f9f20d
0e0ff9
efe0f3
0bfa08
fcd9fc
f2f319
27feef
f0fb09
0e0c20
fbeaf8
390bff
e51329
ef0d1b
f40b03
0bf1fe
08110e
2608f6
0fff21
13ef02
e0ec16
ed1120
1cfef1
0b0126
1219ff
faf7f7
f60c02
fafb03
080700
feee0d
f900ff
f2000e
fd01f2
0605e7
160613
f8001b
010d1b
f406f2
17ff14
```

```
10f20d
effc04
e9f4de
e8ffed
1d16e1
dbe520
1afaf6
f3f7fc
f9f40c
fd15fd
fcf508
ff04f2
0904f8
12f802
fffb00
10010c
f10412
fffdff
f21103
e8eaf4
d5e209
1513fd
f7ecef
fc01fd
02eadf
dbfa17
1012f6
06d3ff
f7ff09
0d15f3
e70b03
021711
0ef3fd
e4f708
0e0c10
02f8ee
0ff5fe
13171e
15f600
e8d5fa
021610
02ecea
ec0104
0a0a09
f4120f
000011
f71301
0b09f5
0df0ef
040cf2
0209f2
1006f9
f7fde7
f90602
fe02f8
01f7f1
041009
fbf606
```

```
0603fd
fff100
f90b0a
00f304
f006ea
0c0afe
0c09f7
00f20a
f6fe0d
f509fb
0e06f7
0c0801
02f80d
0dff0d
ef000f
f20007
070efe
0bf502
f0f6f0
f406fd
0bf3f9
06ecee
0cf9ed
edf5ed
00fd02
04efeb
03f900
0c0207
fc04fe
03f304
040ef0
f002f5
f2f50b
040700
03edf2
f90107
050002
09ebef
000601
07f909
ee0af4
fb08f7
ef0a05
0ff9ee
f106ee
0af2f3
f80c11
f6eff4
fcf90f
f0ff0f
eff008
0807f0
0610ec
04fc10
fb00fb
f9f2f0
0009f3
f9faf6
```

```
eafafa
070bfc
f203f5
02ed08
030c02
09fffb
1102f2
ee0d0d
0af0f9
0bebf8
f003f0
fd07f2
f2f5f3
03020a
fd0106
f20009
f609fe
08fef7
11ec00
fbed0c
060300
ee040d
ee0cf0
ff0409
05f706
070eeb
eefa08
01fef8
fa0105
0102ed
04f8ec
00fdfe
09f1f8
0600f6
fd0500
060004
fe0e02
f3ec00
12fa03
1003ec
020cee
07030f
f301f0
f5080b
0f07f3
faf0fc
ee0af3
f90afc
fd08f4
eb01f5
06f3f0
ee0ef9
fbf5ec
0af4ed
f50200
06f405
f6fb0e
f70cfc
```

```
f904f2
0c010b
f3f905
ed05fd
f70c04
000003
09f410
f804fc
fdecf6
0af3e5
0201f7
d4ffe6
c9dffb
fe19cc
f4f2f4
0ffc00
fceb1c
f3fdf6
0bf9e0
ea03db
d305eb
0cfccc
25f5f2
362b1b
000022
111a20
04fefc
09fe17
fbfafe
fcf203
06faf4
f91bf3
021dff
2a2304
f6fe05
1d4f20
223011
fe1d32
194f33
f1e3ee
eff41b
f708ee
0bfc09
fc1dfb
ff080a
e2d817
02081a
00d2e0
2e0bf2
05f719
f91228
e7fc04
fc04ed
10f6f5
dae508
f2f4de
0115fb
2a0c0c
```

```
181e1b
311e0e
f31013
f1ebf5
e8f6ec
00d0b1
f6fb17
f4fa15
f40afa
f4f117
091b14
10f8ec
cdcee2
aaea04
18f0bf
0cf70b
090eef
053012
fcfdfb
fae3dc
c2b2d6
b3e6ea
01f7b4
000707
f90008
0efb08
06f205
e812f1
faddee
dbef19
ecf9e4
ffdedf
2d2a0b
050007
1003fe
0ffc00
f5e9f4
0f010d
d3cac6
e5dbfb
110800
f31007
080415
f40102
03fffb
04eaf6
04f3de
ecfb0d
1b11eb
000918
08fd00
eb0c06
00fbee
f60c02
caf907
eec2ad
e618d4
f809ee
```

```
edf516
08ed1c
00fb00
f401fe
ec08ff
190afd
f1fd1f
001bf5
f20709
160606
1806f6
07fd0d
dc00f8
0b19de
e6f703
2a07ec
ed1142
1620ef
ddfb0e
fe2502
f1f9f8
bef325
eeebde
0000e2
070016
00fe04
f0fe00
f6fff6
ecf1fc
e8f9ff
04f7ed
edfdfe
f407f3
edffe6
f2f711
f304ea
060802
feeffb
ecf4eb
fa08fc
020205
0503e9
f30efa
080cf6
10f909
f3e501
f1f405
01f700
00f8f0
f6e404
0906ef
03f8f9
fff300
f90bfd
090707
f7f3de
def204
f9f607
```

```
fffa06
e60c0c
0b00e7
f0f1e9
f4f7f9
01edf1
f60fe3
04e5ec
0706f7
f00e07
011006
f809fc
0bf9f0
0ef1f0
e4fe01
e9eee8
02f001
f50b0d
0f0002
ffeff0
ef00f6
eff510
f5f9f8
0502f4
07030a
f3ff06
f0f1f4
edfcfa
ef0bff
e8f7ea
e50cf7
0ffefc
16f010
f50402
0011fa
f1f0f2
fff6fb
06f0fe
0408fd
0f05ec
ec0d02
0b12fb
11fbfa
030102
00eff0
f107e6
eee2e9
0903f4
0b0800
f9030c
f00cf1
0f1109
110d13
06eafe
060000
1002e6
e9eb04
0d0801
```

```
f2e9fc
fe0207
0aeeef
f7f404
fc0503
f7eaf6
f40cf6
09ece5
ff0a09
ff0509
f4f617
e10307
fef8f4
dce3e3
0bf3eb
000001
0304f0
e4f90b
fb0800
ef03f9
04feee
f3ebf2
edeffb
de0afb
0403fa
eefced
fe10f8
09f7f6
f1f6eb
ecfff7
f0f8f7
f1fb0d
f4fef1
f7eff2
120904
f20a1f
f405fe
fd0009
eb06fa
fff100
fe0be2
000006
16040d
2611fe
fad2eb
e0f900
09d9e5
fa15ff
09d7ed
c90c1a
f0f8d3
260cf9
e60af9
fe22fc
eef0f5
ec0afa
ebd2d1
110808
```

```
18ebd1
3b0809
ded2f2
fa472a
11efe2
0c0528
f4e0ed
e50612
2124f7
291e00
261300
1e3e05
182f4a
3c6d41
25191f
1f3329
0c4c28
03e7e7
15070c
0305fc
1003f1
00041a
01ff09
1b1706
f60a13
07e7f8
fc040a
e8d600
ed2919
fbf701
01f1d2
f4e7dc
05ede8
fff70b
2121fc
1c0408
f42150
ff00e0
d2f316
1de4f3
fbcd25
0e0b02
12f9e3
090f16
ee2304
0acaba
030ee9
1401e3
f8f702
dcf42f
06c8f7
0a0e0f
fb090f
d7f20c
d2dcc5
d3d1e1
dfdcdf
d600ba
```

```
d2f5e3
f90ef9
162a14
f6f804
fc0d16
0bedf0
f5fff9
ecf801
f9fdfb
e5d701
1c1111
08e3dc
0f1604
dcf012
e106f0
fc0ae2
ed0100
fcefff
0612f6
10181f
0c13f8
e3f10f
0c0de9
dbe203
020ff5
f7dcec
2509db
ef2821
f905ee
ef0b0d
ead0d2
dee7cc
ebd8d8
dbf2e9
ea0fb4
0bf4d0
cc08f3
040ef0
eaf21a
30170c
01e21a
323222
e6f107
fc09ee
061505
031afb
00fd25
0b0710
04fa13
1e1e05
f1f6ed
20fe00
ca1a21
0f2e10
f5e509
2af101
01d1e5
d40f1c
```

```
fc0d08
0000ef
f7edf0
f00100
fbedf0
ff0201
ebfdf8
0ff9f0
09e4ed
e9ef18
04e9f6
f7f505
e9f2ea
f4060e
0000fc
0df9f7
eefafc
0005f2
03fff7
140c0f
050521
141404
c5e6df
e3ef10
fae6dd
0f000d
372400
3021e7
0cf3ea
e81f25
09220d
18fe0e
f10634
0f10e4
e8d8e9
000cf3
f7090d
f9f2f8
01e602
0feef6
17fe15
0c11f7
fefcfa
1cecea
f90b0a
161dfc
ee081b
d9d0e6
eedadc
1a05ec
16fc14
050b1a
f9ed02
120df5
250831
0a1322
200c06
fefbc4
```

```
c8031e
fbf2eb
0400ed
15f402
fb0400
110ff5
04fcf5
e8ee1e
010c09
01e6da
f00a04
2a00fa
040c2a
331100
00e3e6
b50415
f4f9cf
ebfe13
121709
f6e9e8
fe06fc
fb0706
fcf700
f5150f
021ff7
051104
f8f9f6
0bfdf1
03f5ea
efefed
c90504
dff2e8
fef801
00f4f7
11160c
f60018
01f6ed
fefbf7
0ef2ec
0504f5
0506ee
00f114
0a060e
fdfcfe
e8e1f7
04eaf7
18f4e6
0bfd14
1afc02
f7f4f8
cc0611
03fbee
160308
0c0008
e4e607
f80307
0106ff
05f7f9
```

```
f3f106
fc12f4
08f0fe
ec0b0d
f7f5e9
e9f7f9
04fbfa
f80004
13031c
0a00f5
f6f7fe
0d0503
dfee1d
0202f9
0dcaf4
ff030a
f6f1ff
eb1b0c
1403f5
0000f9
f5f200
1df9ef
010102
2109fb
eede16
f20e13
f51208
14efda
e3ed0d
07fced
f8020b
1910f1
e90534
010414
121000
0014ef
f20705
fcdecc
e3bc30
090cfd
e3f902
22fbe8
1c1505
0a05f4
ec0f04
070804
0a110a
1c14fa
f809e8
1a0010
2515f7
0cfe29
2f3e18
1708ee
010711
020016
191c04
180ddd
```

```
e8f710
fdfffb
0d0b00
0d02e5
1a1412
f9d722
fef100
0000fb
1610fa
231600
060e09
fafd08
00f90f
fb0300
ff07ee
e5bf08
f70a1d
0bfcee
0be2f2
ecfb0b
fc1b0e
250005
21f90a
ec2b22
0002fa
03f2f0
e300f7
dbfb12
fc09e2
f5ecf0
f9ccfa
230c1f
1af6c3
0ae302
f50d2b
fa10f4
f4f100
e20205
180207
09f620
191dfc
16070e
ff08f9
d90405
10f0dc
d7c6ee
fefd0e
feded0
02f4f0
050517
25090e
05f81e
f9fe08
f3f60f
fef7fd
031af4
00faf0
050918
```

```
0209ee
f50306
fbf3ed
13fcef
f1ebf4
eff9e4
ecf4f4
1afc13
130332
1025f4
f2e7f8
01ed0a
02010a
000cfd
01160c
1005ec
fa0b2c
1ff6fd
0017f0
f6d8e9
030c16
e9f50c
ff13ec
0cf600
f2ec15
fb090d
100cfe
00fa00
09fff6
121fbb
acf71b
f9daee
211301
2104fb
040e10
1b02f5
0fff0e
0000ff
0de9ec
02fdfd
f1fcf6
effe0f
f5eefa
04f1f1
f407ec
f80109
eb00f4
faf1ed
faf706
f6f10a
feeb02
fbec03
0104f3
1101ec
f6f2e6
f302f1
0805f4
0e00e9
```

```
0ef2fd
08f415
eff7ef
edf1f4
06ef0d
05faf0
fb0d02
f5eefc
f509f3
fff5fc
0d0500
0cf0fe
f8f6f8
e50303
05fbf3
f4fe07
0cf8f0
0cf9fe
0808f5
04e7ed
f0fbf2
ebeeec
09e70b
f50805
02f20b
fdeef7
f3ebf4
06fefa
edfdef
f1f3ef
fbeb09
e70007
fa0200
f5f70b
fc0105
00fb04
fbefed
ee0504
fd0a0d
e6fdee
f9fe09
fdf5ec
ecea00
fa00f3
f400ef
fff5fe
eaf000
fff2f2
0c0bec
ee0708
eb0bec
f801f3
150ce8
f307f1
07f314
ecebf0
f00009
effff2
```

```
edf20f
eefeed
07ff02
f209ee
05edfa
f704ef
f5e8fa
0d0800
fce601
f809f5
08feee
eaf408
f9f20d
06f7e8
e9ecea
fdecff
f50006
fa0bfe
f000fc
0502fe
01f00a
fff3f3
07f5fe
f208fa
06f502
fe0eed
f305f3
fc09f6
fdf607
0e0e00
0508fd
0300f0
fafc06
ee00f2
f5edf2
09f30d
02f80d
f20b00
fa0000
00faf7
01080a
02eaf9
0aed06
f00fef
000af0
010d07
e9e705
f9fff6
18e9e8
fe0e07
fdf400
f701f4
efee00
ee0ae7
fbf110
000001
1008ff
141200
```

```
0c191f
0c06da
db1504
0bfae5
ede917
0d3e06
ed06e9
fefd13
231a16
02f7fd
051913
0000ef
010b1d
0802e5
ec070e
16f2e1
fa1326
1edf04
fa101d
ed17de
1ef2e8
e3e826
03fdef
152c21
050c2f
040213
25111e
03ffe8
e4412c
28fcf9
fbec12
0410ec
1ef80d
0ffe02
0900fb
eb2910
1d1df4
f7ef05
00060d
ed02e6
0901fa
03eb2e
1ff4fa
e4e814
c5260e
190dee
f4ec1b
fd00e4
f70c0f
dc0e0b
18faf6
252b09
f50d0d
d8f91d
c00b01
01e5b2
08fe19
fd0a13
```

```
f700e9
f5002d
15032b
fc1deb
1808d2
f8d0f1
f6190c
d6dcd1
0000f3
ff1c14
e50d0b
d9fff5
dab8c5
bc9a21
1e25e5
f9efed
000c0f
0f080c
e316e7
eaf903
1afef1
eff0fb
e9220b
e9eeed
0df8ff
e6eff8
03faee
1afb01
eee707
f51310
13fd02
f5000a
090c0a
101611
09040b
1d160d
fef4fb
1f0708
0012fa
01ff0f
e103f3
070c12
02ef03
150503
ee12f7
0b01ec
dbf308
1018dd
1b181a
e80818
0f171a
00f618
0a1601
f2da05
071528
2b22eb
f9fb19
00fd14
```

```
f3eb15
091015
2217e7
00ecfe
f01e20
1200e3
00e814
efe4c6
fa1529
f2e2ff
180afd
f50dfe
f9f1dd
05da10
1008ea
0000f9
000019
f0110c
f0f0d2
09e308
0500f7
090df3
f30cfd
f00c1a
0e0e0a
161d04
ff08fc
eb0405
0ffaf7
f6ea0e
f5eeee
01f109
1beeef
25402b
fd12ec
12340a
fcf00e
fd0b0c
ecf9ec
faf1fa
09fdef
d5dde7
e9e594
eff9e1
e4151d
fa1b34
25f5fe
060fe2
eaf002
f70ff3
f4e9f5
fdecfe
f00de5
18fa11
06ff18
190b01
0bfb09
28fb18
```

```
114330
140fde
e81637
000a24
e8dce2
04e706
dfe9fb
09f3f0
04ff0a
0ffd1c
16fcf8
0d1d11
1d1315
fcf206
e00e19
0412ed
050b0d
f70103
000400
0109f8
201305
01f125
02f823
0a0d02
13fe03
132e17
3f2322
20361b
0eec16
081222
f9f9dd
fafff6
0e1902
171e1a
f81d0b
0cef07
0804f6
0cf5fb
fa0200
f50a06
0eee0d
4825f8
043036
041617
0cf91c
e0e200
01f3e8
0f11e6
fdf2eb
fb1000
101900
0cf816
00f6ff
f0f0ff
e0fcfb
0702f6
05f8fc
0e0e00
```

```
392c10
101a21
110400
e70615
01ecfa
11f900
f90e01
e904ee
ea17ff
e9f304
02f5e1
d0f1f3
0b0005
e8f3f5
09e405
f5f31a
100500
eef7fc
fd1b0b
fb1604
ebfc16
ea0908
f6f00f
fefa11
f9fcfe
3c450e
22d20e
072808
f60a06
03dc03
eafdfa
d61122
fdfef4
000013
060c0d
f2f00d
f3fff7
f9fd06
081409
db0302
e0000a
12ebde
151aff
06f814
0a0a15
dc0b03
07ebdb
dff4f2
f00400
05f7f8
150214
da1925
0516c9
d8ed00
080310
f4dbee
fe021a
e5ef0e
```

```
11fcff
25f7ef
ee0125
1e23e1
fafedd
d619fe
fc1703
01dde9
da3116
05effe
0a050d
16e7e3
f80005
fd0df0
f7fee7
01fc13
1101f0
0bdcee
e2f4fb
0cfde0
00f1f0
f8051c
1d0a0a
15130f
fdeb06
fe04ff
0eec06
04f3f8
14191a
1635fe
0207f1
faf2fb
080efb
06f4e4
03e6f8
e5f205
0d0005
0e0b1c
091104
09ea14
e5fafb
eb1705
1fd6f5
ed0824
1033e8
e2ec23
fbfc00
0600e8
00e5f6
fc000e
1004d8
f5f614
e40bf4
01faf6
fbe911
fe1af8
00f6f4
f10b15
```

```
f5effb
060e00
e6f804
f40b0f
0d0f0e
0706f1
f0fdff
211b12
f9d8df
09f8ea
06f709
f2fbfe
0af4fc
03f3e7
e900ed
edfae6
f4ed07
edfc0d
04f510
1ff0f7
fc011d
f303fe
07170f
f20100
dc0f00
f8dcf5
1433f8
f2f9fb
10dee6
db0416
ffe5f9
e306f6
ec03e7
0cfef4
f4f61b
1b03fc
f0f814
f718ff
01eae9
edf7fa
f2fdef
00fafd
f00cf6
f9041c
060002
e100f6
e5f104
fa0ffb
10020b
f00b0d
17eedb
0000f8
f814f4
fdfb03
fdf9fd
1517ec
04f61a
0ed7f7
```

```
021411
101216
16e219
ebf714
1727fe
1af207
23122b
f8eb08
111312
f00e09
fffcf0
2c1f1d
1d152b
1807db
fbf50b
f21501
02f1f1
f9fd1c
f5e6eb
a7ff1c
05e286
a6a4f3
0b0bf5
fce7f3
e9d606
1a2100
0394c6
0c1d0c
e1f2dd
e9ff03
090a03
f30315
160fec
f20cfd
e3ea12
0e1f04
0b2808
090018
191a0e
09061f
ea0b1e
eff70d
0b01fb
f3e4fd
ff0603
ff17f8
02f001
0c40ff
1afb1b
efda09
c201db
0106ef
e8f7fd
040bf4
0ef6f8
210d13
0ffe0a
0b11fb
```

```
0ae40a
2408d0
f52604
1a14ef
2f1416
222f3e
0f1738
002b31
18efe5
0be229
141a0e
f6070c
20fceb
fcf322
f61b08
fefef8
0117ff
21f5eb
faf622
0a0408
101504
160606
12200c
001100
05ef06
02f203
fe1e09
f906e9
f6fc08
0bfef9
e80c12
001903
f611fa
1bfd06
f6101a
080e01
1d1007
f714fd
020507
2d220e
160129
ebe5f6
fb0bef
e912fc
00e8f8
eff20c
1024fa
c9eefd
0ff5e4
fddaed
070219
040bf0
1ff8f2
f3e003
0c0afe
070302
fb0e00
150009
```

```
0bfa11
0bfc25
1f01fa
1106ed
12dff0
f91af2
0b1200
0a0712
eeeaef
210007
edde07
000011
000bf3
eff704
f50cf0
10f10b
edfe07
10f20b
00f9f2
030dfe
f80702
f8f0ef
070cf9
f4f1f3
f0fa07
fe01ef
f1f701
01f2fb
0510fe
0efffd
f7edfd
f4ff04
09f50e
fb0003
f0f8fb
05ef06
0c11f5
f9f8f2
ee0dea
f6e907
fdf904
fbf4f5
030b0c
f30def
021009
0d09f5
0003ee
0b0701
f50900
08eef7
ec0af0
0703f8
f6070f
000600
0d02fc
f70d07
faf404
0302f4
```

```
02fa06
fdeffd
f40ef4
0cf60d
fc07f7
fef406
fdf5f7
01ea07
06f00e
0d0c09
00efef
faf6fd
fff3f7
fb05f3
07f1fd
08fae9
fbefef
02f2f4
0bf0f7
f4f909
0af2f4
070502
f3f0f8
01f0ea
05fb09
f010f9
0f01f3
fc0403
fb090b
f2f2f4
080502
100bef
05ffec
f4fe07
100100
fefbee
0c0400
ecfe0a
0a0002
fdf2f6
f3f3f6
09eff2
f7f109
f70f0e
00f3fb
02ef0f
0bf9fa
01f20f
eeedee
01eff5
0507ee
0efcf7
f2fa00
010ffc
fdecf4
07ed07
fafd06
040cf1
```

```
05f30f
0eff07
f200fc
ff03f2
f60b00
0cf3ff
08f105
06fffa
f000f6
0aef06
0ffe06
f5fdef
01f0ed
080505
07f404
ea0c07
fefc0b
0e0bef
ee0df7
ecfbfb
05f00b
f8ffed
fdedfd
08f703
f50ff0
fbfafe
f4f0f3
f9f400
000c01
000006
17faf4
01fbf5
fa0c12
f513ff
020d03
f101ee
090720
1a2c0f
24090a
0d040e
172200
090b06
1a1212
fdff00
0026ff
071800
f312f4
eccfce
fc1cd3
ecbbc4
200804
e51403
1a11f1
ddf3f6
ffe8d2
160329
0ee924
e7f30a
```

```
33fd1f
03e3db
f00e06
0ef818
fafcf0
f8fe18
131bf7
1510e6
0c031f
e30f06
0917ea
fef2f1
08ff0d
ef120c
d9eafd
cb21fb
18fea8
1907f6
eb321e
fd1704
ece8de
01ebd6
efe0fa
d80001
cb07db
1029d8
f5d7d6
faef05
d912f1
e1f2dd
dcdffe
111916
0e03ef
13d21e
d3d503
070003
0f0cdf
240ce4
f6ea1d
ddf4ec
0a09eb
e2de0d
c9143b
e91bdf
13f505
0ae018
e1f712
0a0012
0301fa
181a1d
e4e405
fc08f1
1dfcf8
20faf9
fb0415
001400
ebdef5
08f112
```

```
ef10ff
1d0ffb
e8e9fe
daf007
faeef7
1003ff
0e161e
24060e
0d1012
18ecef
081107
2113f6
0a2001
04f308
deff05
000804
fd131c
05f9ef
fd05fb
d1f1ee
fe02fd
12faf5
29faec
f5f526
12150d
f11503
19e9e0
130709
050822
052c0c
fbf809
0bf9fa
fefa08
f40517
0c1d0e
13ec0b
effaf5
0700f9
0904fd
b8e01a
3913d8
05d407
280a21
fa090b
1107f7
02e504
fe1008
000007
000511
ef000a
fafdea
00ef01
02faed
d5f009
fcf502
080102
cdf312
1d130b
```

```
00dff3
fe0b0b
06f0e8
e8020b
ecf4e0
fd0300
24f3ef
173822
f001e5
d8e70b
f0e710
f1f8f1
e5f1f3
f6e9de
07e4f2
cff5dc
00e7ca
f3f70a
001d14
190bf7
0a0a1d
111a09
0b2728
00e70c
f4fbf8
1e02fb
ff0c1b
031608
1400e5
fefef1
141ffb
1bdcff
1e2828
050f04
f8041d
01fd00
e30c15
0603f3
f6fdf4
0af904
140b0a
0c0c06
f2f3f3
030802
e1ee08
fbea0b
f4fdda
e8f6da
0d02f4
f90f14
220af5
feee1e
02121c
f8efe3
caf9ea
e91403
1afeee
0e071e
```

```
14ea16
f21d12
ce0d13
eee8e0
f600ee
f303cb
c0e9e8
18fd0c
e10e08
f4f5ed
e200fd
f505f7
0efb02
0af3f4
ea02fc
1cf400
f43847
f51313
e00002
fdf6f1
fef0ee
f3fdfe
e1dee9
fcebfd
fb11fb
d9fa0a
0efe17
0af702
f0f206
02f2f4
0c08f3
11fc0c
252513
140a22
f402ee
fbef12
eaf1df
0df5fd
ea0ce6
e4e0fb
e908d7
0c0ad6
efe0e7
e703f6
00eef8
f70111
081bed
fe040c
0f1119
05f8f0
f4f705
1e1310
1ceaf2
fceff8
fc04f2
0e0d01
fa0017
3a1606
```

```
0de70a
f4f0f8
f7e612
faedf0
ed02ff
daf5f0
effdf9
0000fb
e40109
eb030e
f0fcfb
00fd05
010311
04e40c
f60000
03e9f4
1c0f04
031812
fc16fe
e7f202
0f20ec
020a12
2a1b23
09e907
2715fc
d71131
0324c4
bfcee8
fffe11
00fdf9
170d28
f900fa
00efdb
f9e4de
e01a23
dec5f9
1002f4
c6c6a2
c6f9de
0c1004
15d8e0
01f310
0bfaf7
18fce7
e91227
f90e19
0f0aee
f408f6
f810f3
2d260d
d90a21
e91df4
15fce5
08fb00
052026
0b2830
f6f516
13f7f9
```

```
ecf2f1
e3f6f9
ebd8df
f80ce3
00e7e6
040719
0bf5e8
f4faef
e0f7d0
00e8fa
170102
191436
f80112
f8f10e
ee0701
0d1af2
0ad1f6
e6ff2e
1a18f1
d8fa1c
27262b
2a2b33
27271a
3222fa
c4d300
14fe03
e3edf3
120aef
f8fd03
1e150f
eff0f3
f4f500
ebf000
1c3cfc
e902fd
e71222
010d04
2a2c1c
212610
1d2122
e3eaff
1a14f9
fef9f5
fde70f
11f909
1602ef
f407fd
1f0dee
f5fc21
1a0b0a
f1ff0a
fd16fb
e70313
2416fa
292d20
f50c23
08180c
f4fc0d
```

```
f303d9
0808fb
1ce0f4
fe1410
0003dc
f9fafd
f204e8
e6d904
0c0d05
27edec
f2f605
f516fd
1af9ef
faf40d
110bf5
11060b
061e00
d31842
3f02c0
f7f9eb
192d05
dd12f7
fb231e
191cff
0adfff
000007
f409fc
0e0c08
fbf206
0e04ef
fd18f4
ea030e
f912f9
00e3f8
0e0e05
01edff
04120f
2bf40c
041a2d
091110
19f9f6
03f6fd
edf6ee
1bfa0f
01011c
27f0ef
11102c
31100f
03081b
fe0ecd
d6dd01
aee6f4
00db89
abb908
fb0809
0ff5f3
0fc7dd
f92a46
```

```
3c2303
f20d10
fb0300
06000d
eafdfc
0205f2
e2ebd7
d8f7ef
03f6fc
f92727
192217
01eb0f
2e07fe
350e04
210b0e
dd0210
000dca
e4dbf0
180c13
eff90e
ebeef0
f3f309
0afcd0
08102c
34f4f3
043029
0e0401
100eff
11deeb
e0f608
ecffec
fafbd9
e2ff0c
332607
e5120a
1b3101
efe711
2f2619
1b3619
33090e
fd302d
3541dc
bbd106
1906fe
fc0401
051a1a
f4fffb
07e8fd
f4f218
e4f8e8
0af00a
25060d
17181d
190d1b
182a02
160c15
0d21f0
e7e1ea
```

```
f7ffea
fdf2d7
f5ef03
001104
020d05
15160b
0f170b
110202
f6f90f
0d0c07
1d080d
fc1a10
091dfb
08ee12
fee6f3
0d2412
1afce0
cbed0c
f3d6ea
0c0500
ed1f0e
0bfa10
19fff3
dff506
0b11fa
f0ccff
090cfc
ebed02
f7f9e5
00150d
f70dfa
ebfc03
0e07e5
0aec16
061316
0b1efe
02dbfd
021512
122f01
f8ff04
f20b25
20e9fc
f0e408
0000f1
0f14ff
f0e719
fb0000
e2e500
041204
d009f2
000fe0
f4e2da
f10306
0118f3
0efef9
fd0810
edf1d5
e7f5ed
```

```
ff06e3
edf901
3110f3
194641
ff07e7
e5231b
f303f0
1ff300
ef1c00
14f712
2e3521
fe03e9
ff15fb
041017
f11d19
2c1f30
3d00ef
fc2825
28672c
fafce6
efea11
0cf4e5
f3eff4
18f80a
f6ee0a
070004
14f9e8
25d6f5
04fb0b
f802e6
f5ff02
f31a0d
19f3fa
1a05fa
09f51f
1bfd1e
0af6f3
22fd15
071431
03021b
0f121e
1cfaf5
38e6f8
ffd920
07f305
e2f3f3
001716
17f5eb
ecffe7
15e615
e7cafb
bf2235
20e9b7
112210
1c2bf5
02040a
f7daed
f7ebf0
```

```
d4f9fa
ff3adf
f4e1f6
110d07
eceb0a
1001fc
0af8f7
071605
e8ee05
e0001f
08e6dc
2221e2
fa1d0a
00ff0f
13f80f
00ee15
111e08
2e0f07
f4f5dd
f707ec
fe0002
f6f80a
051103
fbfe05
f502f7
eceff2
f9df04
f006ee
171502
f9f310
0a0d04
eff9ed
090505
0016ed
de1804
e4dde9
f400f3
0a00f1
f6e1f4
f20afa
fdfcfe
0612fc
0700e1
ede021
ebf517
1007f5
0808f6
081300
00fc13
00fe06
f900ed
13f8f0
f6e9f3
11061d
010009
fa09ea
08e2f5
16030d
```

```
14180c
e70c33
230de3
000008
f30001
f7edec
f0ebf8
f3f5fa
ebf7f5
f7f0f1
f3ee00
f0f901
f2f8f2
04f8f7
eef006
f9f0f2
e7f8ed
090906
f7ff0e
0004f5
fbfffd
080afb
03e605
fef8f1
e6eaf2
f9f4f7
f608f9
0aef01
f9f0f4
fc1109
fde9ff
fc02fa
e8f3f4
f90df5
f8ff00
00ecf7
02f9f4
f504fb
0beff7
fef7e9
000406
e7f6ec
f502fe
f8f5ec
ef0402
0516fd
eafff5
f8e903
fafde8
faf9e7
f603fa
ec00f2
f8f9f6
03faff
ededea
ebec08
04e809
0102fc
```

```
15fbf5
f6f709
eb0104
000bf7
f3f3ee
02eef0
f6e9ff
f001f5
0609e7
e6f1ee
eaf1f6
f20908
090b01
f809ef
00f703
fde6e6
fd07fe
070902
06efef
00e9ed
f41ae9
f1f4f5
00f5f9
f0ffe8
f5ff04
fbeefb
f80ff3
e5faf2
02e7f5
f1eb10
0fe9e9
f101f1
f6f6f3
ecf5f2
f303f3
0b0c0a
e8f7f8
fdebe9
f0fc02
ffff04
05e9f4
0503f4
ff09f8
eafbf2
01e709
f9f9ee
f904fb
07f5fe
f7f7e6
f6f503
0102ee
00f6f7
fde914
e5f9f4
f307f0
04f3eb
f4f707
0300ed
```

```
fcedff
05f90b
09f705
02f010
e6f7fd
e9fcea
f9ece9
f5ec07
09eef2
e908fb
f4ff00
0002fd
00e8f1
f5e4ff
0912fd
e5f1fd
f6f601
f50608
09f201
f2fcfa
07fdeb
0000ee
0aeff1
fc0f13
09fcfc
070401
fd0009
00f501
f5faf2
10f9f7
f3f80e
00f1f0
ed0c0e
f2f3f6
ec0509
ec060d
f9f2f6
0e0f04
0becf3
fcf0fa
07f8f0
0609f4
f5faf7
fa0b08
07f0fd
fd0a04
00f50c
f9eef6
f1f3ee
fd0af0
fc0e08
1005f0
fef305
f7efef
f8faff
0f0ded
f7ef01
f9fe09
```

```
f30cfb
fdeeed
0ff8fb
0af210
0704f5
040f0a
fdeff0
f006fe
000b0d
f4fbef
ec05f4
fd0df1
0aeffc
fff7fa
f608f3
f70300
07ff10
f8f6fc
090ef5
f6f508
0b00f7
ef00f2
f1f2fb
f206ed
03ee05
030b00
09f8fe
f3f2fc
f1f5ff
fcfbfc
f703fa
f9fbf3
f2fffa
f905ee
fef5f4
0f0bfb
0d0cf4
01f4fb
0500f6
00050f
fcfc02
0e0d02
f80900
090002
fff0f9
0eee10
fe09ec
ee02f7
01fbee
06fbf3
0cfcf4
08fb03
04ef0a
fef8ef
f408f1
0f07f5
050f03
f2ef09
```

```
0c09ee
09faf1
f2ef08
0bef09
010900
0e0409
05f106
04fbf5
f2f90b
ececf7
090400
f4f700
0c06f7
01f2fb
09fc0c
02f401
0eef01
0b00f8
ecf908
fc0af9
ed0205
0502f7
03070c
09edf6
07fb05
ed1111
f70a05
faf2ff
f0ef00
f5f9ec
050e09
14fd03
0bf0ee
f5fdf7
f00705
00f5f3
ee07ef
1002f2
eff6fe
000000
fd1616
fb0dff
080f1a
2915e8
f40506
00edf6
ddf819
1a0de5
130c23
0700fe
241f2d
2d08ef
0e3253
21f204
051efb
f3f6e0
f8fd0e
09d7e7
```

```
00fd2a
19edfb
0c121e
091efe
f70e01
000fed
cdd0e3
dff136
10f5fb
08cb03
180a0e
f9e2f3
ff0605
140a16
2a1f01
1f161d
0e00f1
fef405
13141b
f5200a
06f5eb
e2e318
140c02
05f607
12fef7
f5e722
2606f5
12fcf5
053835
f60b1d
e2eefc
e4e8e5
1e0506
e5142b
f2f2c8
f9dfd9
eefaec
dad60a
ff0fe8
ee13e8
fb03ed
101ef7
12f303
e01c1a
0f0d03
0009f6
f2fcef
fa0f08
02fbff
0be9e1
e0f424
e7fdfe
f117f4
17fa12
04f320
f9fde8
d4e4e8
fc00fd
```

```
0d0cfa
f3fb1a
00fefa
f30d0d
1a09f2
00df0f
f90bf6
00eafb
2e01ff
faead1
161703
1c161a
05101d
dc110e
0001e7
0cd8ee
0e0efe
172006
050720
0e1013
0cfc03
0f0700
03f100
001316
13fb02
e5180a
fb11fc
0af31a
ea060f
ff0913
f014f4
e2e5f9
130405
f6090c
0b3b20
18fdfd
111c20
0cff12
0f0f15
ffefde
1bfe17
eef521
120407
1b060b
22fbff
020c0b
f5f512
0409ed
f3140a
0bf9ed
e319f9
07f00f
143918
ea070a
e80030
ffd3ef
04f6ef
0000df
```

```
061f20
06e9ff
03fa14
f9ede4
fdf405
d304d5
e8e3d6
e0ebe4
021dd6
eb0408
112212
e7000b
f0f6da
e8c90a
f003f2
0fdae1
2300f4
1d1c31
f413fa
2c1c08
fbe90f
211f1d
24fbfe
f4eb13
07fcf9
280ded
f63d24
1b4205
fb12fa
f04921
322200
19120a
024838
0303f2
260610
05ecef
faf8ff
ff12fe
0deded
f2fc00
0216ff
f8caef
e2f5ea
010bf8
24270b
ec0502
201404
1c0ef3
d6df21
01f2f3
1bf9f3
171320
163c30
3226f0
00f009
00f510
f6ec02
14ffcf
```

```
f3fbf7
091a06
0406f3
f60cf3
e90015
07080d
cbdbed
d60020
16d2db
161d14
0e2b19
fb0a11
07f5f9
040df9
e9dfd3
dbfcf6
ee00d2
0e0a0c
f81400
0f06ea
09fe0e
f205fc
f6dde0
c70018
e0fdee
f912e5
0012f6
150a01
160608
0d00f8
f7110f
051708
cad0b2
02dff1
090515
1111ff
f8f311
f2f2f3
d610ea
feff00
f6e8fe
e8ec05
07fbf8
1010f2
1b0d0d
011f01
06ffe0
fffafc
bff302
edcfb7
1530cb
06f901
0b0a01
d5f11c
0300f2
14eeee
fa02ff
280afa
```

```
f5f115
f6f9f3
ff0dfa
0209fa
fb080f
e4f6ed
001809
1008fb
dce309
11101d
f70e0d
10100c
f9001a
001508
f5ffff
bcea1a
08e7db
0000fa
001107
f91c0a
03f1f3
0bff1a
f214f0
071518
31f6f3
f92518
eaf2f7
230e00
000900
f91b0e
1dfd10
10f201
010910
091a20
fcedf4
28fee9
0ff04b
130804
e7ebf5
f50405
f5d9f5
f106fc
04e9fd
db0108
1bc0b5
dffb13
10151f
31141f
f8e81f
032de5
eda6da
fe0820
f0ea04
08130f
19efe6
f30002
0f1520
1d220e
```

```
f307f7
f50003
322e01
2eeb18
eb1918
f2e9ff
f4dddc
ebe400
f508fc
0afff1
ff261b
f01c15
2d07ff
efe322
f11c02
f0f1df
e5d51c
0d280e
231ae7
020a0a
e2ec05
faf2e2
fc080d
0b00fa
011e0f
3f0cdb
031b1c
1700fd
02e530
392c0a
00ffe7
ed0b05
ecebe2
f3e4ce
f81811
0cfffb
041106
0008f6
f216fd
f0e2f5
0015eb
2201eb
180422
08ea1e
110d25
0600f4
fff20d
fbf0fd
f6f2ee
e7e3fc
1d1613
0311f6
00fe09
fc110a
041714
f41812
fb050d
f3160c
```

```
0e0505
f40e04
1508ed
f2f405
10f70e
f3de17
ecdaf0
ebeaf1
08e3dc
231425
08020b
0f1902
0003f4
f9f6fe
f50d19
140304
08ee0c
fa111e
1f04f2
fcf50a
09fa00
060b11
f71313
f60917
030dff
f40410
0503f8
2f1bdd
d3fc3f
1af122
00f5f5
07eb01
e1fff1
050303
eff90e
0000f5
0f0018
fb1202
f6f2f5
fbf003
f1fdfa
0af2ec
21f906
04152a
fefc00
150008
11fef1
db0df5
1414e4
05f107
0c092b
0f0013
02f0f4
f41b1e
f617fa
23fae8
eeecf0
e62a0e
```

```
01f312
ee04e5
0fedf8
d300e9
f2fad1
d8b8fa
001e09
c8f1e8
d9c00e
ee0af6
c8abc7
f4fb0a
09f703
08f8ee
fcfceb
ede322
010a0d
0b0ef9
fce60d
003c0a
24ff08
060deb
10120f
faf5f8
e40915
00ed09
05fa04
1507f4
fcfff8
06f502
08f903
fc0aff
2813e7
fbe706
e51704
090b1f
11f7ec
f1f3fa
e203fc
26fbf1
0bfa12
f6f30d
f517fd
3805f9
0d0f45
021613
180810
223302
eaff00
004b34
18fef1
23f908
f5201a
080514
0bf405
f8ee05
f9f209
fee911
```

```
f905f1
33f5f9
07000a
1a1dff
121b0a
08fb26
0b0d04
1214f8
fef80b
05fbfc
00f7f5
1011ee
fb080e
0fee1a
09eb11
12f8f5
fc0602
f709fc
0d251d
17f8f6
0b1f1e
000509
001309
0405f0
fc04ee
dde309
171714
0d0529
0aeeff
fe0816
0df5f2
fbf8f2
ec0dfc
f405f3
ece014
f81d37
09eee5
f802f3
10f813
12fe0a
e30115
1218f3
16e7fa
07f614
04fa09
e61516
11f0fe
f1f4fe
03f8ee
f00e27
1becdf
0619e9
02e0e8
000007
091113
0a0eee
fd230b
13de0e
```

```
0a0102
d60cff
dcf1ed
00c9df
102d19
f7fb00
071f1e
070611
f000fa
f2ff0f
fce8ec
ebe6f4
ff200b
524828
01f231
fe380d
f505f7
00e60a
04dbf7
fdf6ee
eaf6fa
1011fc
ed0b23
342f0e
1010db
f2402f
1e0fe1
e5f70a
f72119
0a0501
1501f8
f0f10b
021700
1802fb
04fb04
01fe01
0cf103
06f931
2d1b16
f2e50a
000711
fc0700
0cf1f8
f2dcf1
fcef09
ea04f7
fefd05
120316
49291c
fd0b10
eb1d21
120e1d
edd4ed
d3fae3
f900ee
1918f3
ebf701
060107
```

```
0200f8
f8181b
e1f7f6
f2e812
24f2d9
1d0523
2d241e
002130
0aebe2
10ec0b
e5c7dd
cae8dd
dceccf
03fc0b
0807fd
060d03
e8fe0f
06f8e8
03db00
e50f1a
f4eaed
06f9f5
ff0f19
00171b
180502
0ce5fa
f4dc13
0bfaf9
f3daea
23e4df
f70001
101708
fcf1f8
ebf612
f005ef
ffe2e9
ed040a
0ff0f4
fafa0b
16eeff
171f17
d5fb01
0002fb
f102df
f2f5f1
cee3f9
2f4bd6
f00111
2b1322
f208f4
d11124
0f12ee
10d6d5
fafcf0
fe17fe
07f316
040c12
1af1f4
```

```
0cf819
001a0d
06e6eb
ec1b0e
eb09f1
2008f8
1ffa11
f62213
f3fee1
0005f1
fddfee
e4150c
eff4e5
00000b
0ffe2a
f70b0e
fd0eee
0ae703
fd0300
0f0400
100fe7
f0fefc
0a10fd
002401
130005
d911ff
10e7cf
ebf5f0
1f0a19
1904ec
2a04f9
d81c2d
032896
cfd109
0a030e
06e6fc
050706
ea0900
f40a04
08dfde
e81c30
0b09fd
e41cfe
c1e9e8
04120b
f1dff4
df1109
e3f6fb
081710
0a0de2
f31d1c
140312
f0fdf2
ff0c00
0313ec
37fa11
cad91c
0d2fcc
```

```
03f900
0c160a
0d0318
1b0501
e9ff04
1c0809
f910fa
1b050d
e400fe
0124fa
05e3f9
eef103
14fae9
10f1f5
e7f6db
fa0815
070702
111400
fc0900
fb0122
06ebe7
100905
39e10e
f13349
2e2fd1
e7dcfd
1b0024
17210a
28fdfd
0f1313
ec0afb
000b08
ff0910
1304f3
0afc13
fd0602
edf50f
07140f
f5edfa
2b36ed
c8e114
09032c
0611f3
19f608
fb0f11
1511fa
f3e306
0c01f8
1b0c0b
06f80b
0b03f8
09ebf0
fe0a0c
fcf913
06f102
04f30d
11051d
170100
```

```
e70c18
dbf9ef
1814f4
150a04
f02125
f2fb02
ec10f9
fcfc0a
01e3f7
ee2105
0f17fc
f3ff1d
0703f7
07f508
01110a
2c10fa
0c0527
1407fd
fb05fd
fbff06
fb01f1
f7f917
0e050b
e11251
2ef6e3
eb1908
f5f91f
f40f05
f71302
e0110f
0511fd
000006
fc0318
1ceff7
02e9f9
08150d
04f3f4
c40c00
df07ea
e7f8f1
1816fc
e6e805
191631
3a0019
f9f5f1
050c14
08e0e4
d8e1e2
f0f2e8
fb110f
0b0d0d
d3f710
0ae7e9
fee71d
f8eeed
e2ebe6
0708ef
13faea
```

```
da0ff4
fe2e06
fde0f4
f41a18
2d1ae4
f0f509
ff390e
f5fff7
fd06fc
f4e505
18ec00
eceff7
eef000
170ae9
f600ec
ef0b0d
fd03f2
0f0909
edf912
f400ed
f1d9f7
f4f7e7
e9d0f1
dfddd4
000af9
0e0106
0cf104
f7ea17
fbfe00
01d7e3
eace03
eb210a
1dfacc
f300fc
d50b22
e1fdf7
03e7ef
e1fb0d
f505da
ece301
140a0b
00143a
f910f7
0df9e8
07f6e2
eae5ef
dbe2de
d0fcd1
e3efda
f8f215
11e701
1d02de
de0233
f5fadb
ebe8f9
dc14fa
ec03e1
131100
```

```
fbfa10
1c00f6
05f928
f7dbe7
ddf206
04f9e6
dff1e8
18dae6
05f1fd
171ced
fe06ff
030e2c
230ef4
e9eb0a
f3f600
0000d7
020911
fafe06
090510
f1e2ef
f1e1da
ffe8e6
edcef6
d1dcf2
1a34d9
08eff8
e00523
2806f4
0ce71e
350c05
f5f708
f51907
0ff4f1
f1f5f4
eb09ee
03f3ff
deff22
1b17ed
06eb02
dafa01
09ffdd
f70c19
0206f6
40302d
f9ffdc
0008f8
0cfbf2
f41cef
c9e7ff
0000e5
0b0909
08faf6
0a141a
e60cec
0a0bfe
f4fbfe
d21a07
1ae1f2
```

```
fdf800
ff130c
fe1cf1
0de918
02ff07
f705f0
0007de
eafceb
061902
183525
032ced
ff072a
325143
fcef0b
292121
18fd26
f31b1e
f5f9fe
f4dfdd
21fbd8
c3ca03
103e4e
5eeeed
03f72f
0e6a17
f5fdc9
e2e3fc
ea000b
eee3fa
06f4e2
f407d8
e7f4fc
fbf4f7
f5f5ea
f20810
1c2207
e2e9f3
0e373c
0ff5f4
fd1e13
0c1c07
07e004
ffe7ee
09d4e3
050f0d
1c23f3
0b1125
0b343c
53100d
42004a
f0086d
d9e8e5
f4efe7
05e1fa
f90ff9
1528f1
1df5fa
d82120
```

```
f222de
f0f40f
0425c7
e8f509
1b3254
33e3e0
f32530
13f50e
722412
fdfae5
fcf6f4
f8041b
010ce6
0a0e09
fb0a16
00160a
1e06f2
0af606
121518
121c1f
07ebfb
08231c
f300fb
fef30c
e30f0b
fc00f3
101809
f600e5
fe0b19
fbe911
ece602
03f8e4
0fdeef
10fee8
000008
fae808
0104f5
1205e8
f90900
09071f
290af1
f7fd0c
f4ff25
060017
0d0708
c7f90f
0d07dc
f2c2fa
0311f2
e0e6e7
0cfb06
f5ee19
0807e7
00f7f4
17effc
14dffc
f00ae9
060bed
```

```
090ff9
13ebf4
10f112
f00a02
1415e5
1efbf2
170e19
150e0c
11f804
00000d
f801e4
100d08
00f2fa
17f8f4
f0ea14
1116f9
f803fe
0b1d11
fbfdff
ef03e3
04f7fe
24f801
fafe0f
0ee9ef
fb1201
1cfee8
f7fffd
18e5e0
10ed3d
3b2ffc
ecede6
f914f5
0fe8f1
f2f901
fc03ee
1bff10
0dfce2
f1eaf8
0f0706
15e716
f30525
16f7cd
d7a6eb
fe18ee
f9f3e8
07061a
0ce9e9
fefdee
17151a
1ff607
e10206
0721f3
1939ff
14e919
010825
e7eee8
e4f805
12f807
```

```
f60e01
080012
e2e80e
e4f4f8
290ff6
17f512
1b0c2c
ffded7
da242a
0b0ad7
fb050a
07f505
04ee04
0b0007
0c0912
fa0108
1c0407
06d7c2
ee2a16
10ecd9
0ee018
202a16
0cf9e0
d4261d
0afadd
e2cb1a
1f1022
f700e8
2506ec
010cf8
0b0b13
110f07
0505ef
06f100
05061b
02ebfc
17ef10
07e2ea
f7fafc
edfd07
fefd01
fceef2
1d050d
090006
00f200
0209f9
0b0bf5
140801
f40a09
00fff9
000ef2
0b0207
e5f8fc
0eeb03
1b1410
171e03
fb0616
08faec
```

```
ec02f3
180e02
1df127
f00402
00220e
f0f1e3
eaed0b
1fe6fe
030312
fcff05
11f6fa
02edf9
fb14f3
05e9f6
fdf60c
180300
05fd1e
1700f8
fa100b
211fdb
e70c12
19f80d
101a02
fff3fc
ffecf9
0d00fe
e8000d
000004
051501
ed17fb
00ef09
effbf2
ea00fb
20f102
1bf603
11fd05
fb0201
f5fefa
fcfdfa
f3e4f9
100beb
0a04fb
1f1d11
06ed0a
030cfd
110b1f
011f42
33f0fc
d3f307
032b10
fdf608
0603fd
f5f2f9
d7ea05
0ce8b2
afeaf8
ff1dfa
ecefef
```

```
e6f205
ef0b07
f5bdf1
eded19
02f108
05fcf5
f5f5f3
f4f60d
1200f2
060105
f60907
e32914
2f081e
1d1c00
182911
f2fb06
01f613
e3f71d
f7fdd6
feff00
fe0dfb
ebf1f5
09051b
eefdfc
3409d8
e1eb01
051c0a
290516
fbfeea
131801
1611e4
fef700
ebe316
f3fdf1
ec04fc
2c11f4
ecf83f
122c13
23e217
2f2819
eb0326
194017
1fed02
261919
f8fb2a
010c01
00f81d
f4f902
fe0af3
00f3f6
f700f3
1702f8
0df6ff
1d1b0c
232322
0bf818
f905fd
ee0cec
```

```
00f304
fcec0a
0f0cf0
fdf005
e9e800
faf2f3
08ec0b
f50bea
fff90f
1e19fa
010704
1d1208
190aff
0d0c0e
19ef06
00ebf3
ea06eb
f3e402
21f9f0
ecfe0c
eaeff1
ecf8f5
f4f3e5
f8ee07
fd07e8
daf5ec
eff913
fe030e
f808f5
f0ffef
130ffa
f6f20f
f20300
f0f804
fe09f5
100604
1f0911
0e10fe
13daf4
190a06
e90100
f7ed09
1600fe
090dff
f4e300
000005
e00501
f8f0f2
0a0e03
00f4f4
00ecf5
f406fc
f9f8f4
f9ddff
07faee
eef2eb
ed0aff
01e1fe
```

```
f8f812
f1f3e3
f50b04
ede9f5
05e9f3
f208fa
eaf5fe
fc11f7
fdfdf1
eaf50e
e21100
f9f7f1
f0f80a
fef006
020ce8
f1fb06
08f7e7
f7ec16
1612f7
e30305
fb110a
faf204
def0e6
f9f702
08ecf2
00e6f5
f4e7f2
e80c05
13e7d3
f3e0fc
f01204
e6e709
f60600
f9fb0b
faec02
0006fe
efe5f4
f1f5fd
0bf0f7
e7f507
ef02fa
efdc00
f4faff
fd03fc
0efc09
f0f512
0bfc07
e706ff
f5eb04
eff702
e6f4ec
e9daf1
03f6f3
fbf2ed
ec0df6
0fe4f1
fdebfa
0c090e
```

```
fae4f1
0d0a0a
f31600
0111df
f3f700
e4e802
fcf3f8
07f0fe
f0ea02
e0fd07
f6f3dd
fa0004
faf5e5
e706f2
fffceb
e7e205
09ebf6
130307
04e7ee
fddf04
fcedda
04f400
f9e106
feeeed
ecfa0c
03f5fd
05e3e8
fe00e7
eaf3f9
e6f5f5
faf2eb
05dfee
eb0801
f0e5fb
e8edfb
f8dbfb
ebe7fe
00ecf8
0df7ef
e9f3ed
0cf4fa
edf7f0
faef06
f9f80d
fceff7
daf6f5
f9fdeb
fee502
eafbf9
fff6ee
fe0401
fee901
fc08da
e4ece5
eaf608
10e4f8
faf10a
f4fc06
```

```
160bfb
f6f117
08ffee
df01fc
f1ee05
0000f3
e90c0c
050b01
05061f
00f6ff
f80e0a
15fcf4
ebeee7
f208f4
f6fbfb
cddaf7
fdfe16
13f401
080f17
feeffc
000317
faeeff
0de900
3020f7
fe0040
1dff06
0c10ff
fd0721
f8ebfa
fafffe
e70209
05f7e2
fef7e0
cb0603
0909e4
dc17ed
f6ebf1
e9ea0f
f2fae3
000404
02ede8
06f1f2
e202e6
05e4f7
f2ece4
fe00ef
1003e8
0cfe03
0b091a
faeb1b
1209e8
0304ff
0bf407
f8ea04
0405fb
e2fd0e
fde1f0
1510fd
```

```
203b27
09f821
0f0ef1
e913f6
060414
ff08f7
f3dff5
281207
0508e6
dd0df8
fe01ea
ede700
fe1eef
1411f7
06dc00
0f0205
260413
1c202d
00fdf3
052406
29f815
0f1bfe
daf317
e1e3e9
08ece8
110c00
09f607
0f130b
faf2fc
d50206
0010eb
06fb08
1900f4
02ee15
061df8
fd050e
14110c
eefdfb
0cf602
06010c
e6e10f
f6f5f1
dbe400
f5f8ff
e403fd
fdf3fd
eedd12
f5eefb
dcdb00
05e6e4
1b1f06
0a1105
fd0ced
141525
f5f405
0b0a0e
1731ed
def00a
```

```
011af8
f0e0f8
f5f9f5
ddf4ff
e6fff8
03e1f6
090b03
faed00
f0eae2
fefff2
f6fde5
df0afb
fafcf0
10fcf7
f20ef8
f50d09
13f916
1bfb0a
071120
000909
ea04f1
0a00e9
fcfbec
0000ec
0ff20b
10fb03
191004
07f70e
0c05f6
f501f0
1c06f1
f315fc
1701ef
fc111b
fe01ec
f2f005
fff8e3
00f306
f70018
10ecf9
0c1324
dddf11
ea12e9
d9edee
f7faee
fad3e2
1c1f09
110322
322b15
6a1805
f11e6b
393b00
26150f
f20600
012805
00dbf0
f30216
0000f8
```

```
02081e
17160a
060a1a
fa1601
f9f91b
16fc1a
04000d
fee908
eacae9
f905e8
f500ec
f70705
fdedf6
0cf602
02081d
1926fc
e6fcf7
0ffeec
150919
09102a
180418
fff9d6
ef0202
05def9
fafc10
f6f405
110431
0b0902
0812fb
fa0f17
2f1cee
d8f500
16faf3
faf404
0121e3
f4ecfd
00f2f2
f0f7f0
07eee5
faf62e
350dff
e5f6f0
faf1e4
fd0bf7
0dfd11
fb0317
1801ea
f01114
f1f8fd
dcff02
d1d3cb
f1edfb
00e5f3
130403
faf421
1e020e
ff070e
093221
```

```
000410
0e05fe
17f503
0cf5ef
0cfd00
09fe10
0b1308
12f70c
e1e5eb
0cddda
fc040f
051907
081911
f0dcf1
e61d23
22fe02
180819
080403
02ff05
ef0e13
f805f3
0513f5
f60913
0102fe
f400eb
130b07
fc0008
ee0cf7
020900
040af1
041817
f4f005
1b0ff6
c8db20
fe2d02
03f0f0
00e9e5
0e1622
23fdea
e91dfa
fd16ff
000011
ea010f
f2200e
05f2eb
1f0606
00dc11
fb05fa
2301e7
010d1c
feef01
3a1707
1603ee
fa230c
e40b1b
1d0af7
d8e7fe
f8140d
```

```
19f2f0
ff2c18
04f6cd
c518fa
1eedf9
f2060c
00070a
f2f821
fcffef
e2fffc
04d111
24e7dd
e00a0c
261313
fcd710
0822fe
10d508
00f5f4
e30800
040d02
122818
14f6f3
fe092d
0a070d
fbf000
16f7e3
ed2734
f6f0c5
d2ed07
220fee
e60b00
1a0003
eded0d
fd07f5
0afe1b
181004
e0daff
f1f5cc
c30c14
140eed
dfd603
f2090b
141318
fff4e1
09fb11
f608ee
140007
10f9f5
02ef07
fc1401
0e12f0
f0eb0b
12fa06
ceed0e
1609e7
ecbff6
fafc09
21dcf3
```

```
1bf1f7
060def
0c0303
e50c06
08f402
0be6fc
e90a1a
2d0cef
11f609
1ceeef
17132a
fc08fd
f8f3e8
f507fa
000e14
f6f305
f30b16
ec01f1
17ffff
160a25
1417ff
f60013
1808e6
eff704
000bfc
f10c05
2113f5
f20e1c
d60904
0dddfa
f7e104
f707ed
f9d9fa
fdfe11
e2e3f8
01f0ec
ee0fdf
f1f8f3
000023
1e1def
f30011
fd302b
00f012
11f50e
0b1013
281110
050812
1d040c
02f3f5
0404ec
f202fd
1818f5
f1f720
f3ecf8
2ce9fd
1dfcfd
d918f5
fa0ef5
```

```
edeef3
0000dc
05fe14
020915
0ffe06
080a0e
fcfde6
e82719
f004e6
07dbff
00f11e
ed08fa
181201
311713
0bff25
282310
f607ec
f01700
dcffe1
19f2d6
f2de1a
242f11
270bf9
280910
f72307
0708d7
e6e407
ab0003
1fce9d
e5e2ff
f0f217
251b0d
16e5f1
112d38
3a2a06
061211
d9fa00
e5f01a
16fa06
0c07e6
de0307
ec1103
faf2fa
ef1201
04150f
09de1d
f9110a
1c12fe
0605fa
030501
0606e2
d4d603
071310
dcfe19
efd8d2
e0b6f8
e7fa0c
1d07e4
```

```
25e307
e8331c
020010
f7ea01
fbd9fa
c50700
ddffe9
0fe2eb
ddeb19
001606
e80ef2
101f08
d8c227
0820ff
0b04f4
11e9ed
ed3021
1c32c1
e9d301
070603
0b1cfb
fe0909
050a09
f2f3dd
e5030b
f507f2
1d0ae0
fcdbf6
22121f
160aee
f30416
02221c
0f1601
fce5ec
e9e3f3
01efd8
100f0c
1004fb
f41b1b
000923
0a03f9
ecfcfb
f6e80c
e80dfc
0f1815
ff031f
faf2ee
12f417
f6ddf6
f517fa
f3dbe5
f10513
f8f0f4
f100e8
fd0815
2a0400
0c042f
0200fc
```

```
fcf7f5
f9d5f1
fb0403
e3fb03
0407f7
fc020a
faf905
fafe0a
0ef1eb
0ae7f1
f9f903
1dfdda
e7f414
00ff16
1afa08
0cfcef
d4162c
f0fd02
fae30c
0000f3
00edf9
0c08f8
ec08f3
eb09fb
0a07eb
08ec02
f408fc
0d0ef5
0004fe
f7eb07
f5f8f9
f4eef5
f9fafb
06faff
00eefe
010ef6
0af7ee
0ffd0f
00030a
fb03ee
04fb00
f00d0a
00ed0d
040c0a
fff106
10fcf0
f3faf1
ecf208
02ecf4
f5ec0d
feecee
0ffbed
f8ef08
f20802
0feff5
0af9f7
04eef2
0d02fa
```

```
0003f9
fb09f9
07f8f7
0205f4
edf00d
f80d0c
f3eeff
fded03
eb0903
f4f7fe
fffcf4
05ecee
0efa0e
ff0000
f2ed03
0000ec
0beff5
09f4f0
04fd09
f7f4fb
f503f2
f9f500
f70a04
0df300
03020f
f6ee01
01f6fa
f2000f
edf60b
fa0800
fceb04
fffced
0cf4f7
09010a
f4f803
000bef
070807
03fffc
06ed03
00f7ee
f9f104
fcfef1
f2000b
f4ff04
f70507
eefb02
ec0df8
f8f7f2
f201fa
ee04f8
040400
feeefa
0d0cff
0c0fff
eef2f7
06f8f9
eeed04
fcfafd
```

```
fdf8f6
0000eb
fa02f5
ebf8ed
ec02f5
faecf5
f903f7
00f3f0
04050e
0bfaf7
f4000b
ed01fe
02000b
fbf605
fff5f9
f4030c
0c040b
04040c
ee0300
f2f4f3
060900
010def
f9f9f3
ef08ec
05fd0b
0ceefe
f8fa08
04fb0c
f5f300
f7f3ef
0b01f7
f501f8
0a0e01
fdf2f4
f70deb
04f30a
07f9ff
000004
151a0e
f3f205
e6ef0a
f1e7eb
07fbf6
17eef3
fcf9fc
17f2f8
151306
eef2e8
00040f
e6def5
f5fee5
d9e9ef
08f90e
f6e613
180815
0b4135
0d2a21
2c0d0e
```

```
ff0a0b
002b36
02ebf9
0ef21c
f50406
23f3f2
f2191b
e73ef3
e404e6
e42013
1500e3
faec04
e71610
f4f005
26f600
0ae7f1
00f3e9
00ee00
fc05ec
f4f8f1
020ef3
f6041b
00130b
13fa0f
2b2924
e7fc2f
00e9f5
09f4f2
05f501
06f8eb
0ff9ef
fc0815
3d3219
234508
3218e7
12042d
1b2e2c
2d0810
fbddf6
f10611
e61ce4
1fd7dd
e4fcfa
111208
0e0af8
fa0ffd
06db0b
0029fb
271ef9
12180a
00ff1b
382e18
22eb00
f91b3c
1d0904
fc0117
eafc04
1000ee
```

```
07e9f5
fef8fd
fef20d
030900
0e1cf1
100f03
091a06
15091d
240e27
0be6f8
fd0600
00f60b
fdeb06
1e06fb
f3ff1a
0505ea
f8fbf2
f3f9e1
f1f506
01fae9
ff07f1
03060c
fc1714
160808
111012
f800e9
090aeb
f70718
151316
02030f
191800
d9e518
0ef2f2
d9e5ea
06faea
daf2fd
e80909
f1fcfd
f7e3f1
0af801
0d0dfc
e50ffb
ecfee9
f1eef6
f40efa
f7f702
ed0815
05052d
0f00ef
1126fa
dcfc12
ea1bfd
070d05
fefb17
04feff
000000
1af6fa
08fdf4
```

```
fefd04
ff08ef
dc1106
faf4e5
d2f810
0c1810
f0f2fc
010701
f6f4f3
1ae7fc
f7f4e7
0300e6
fc0700
06f8f0
f9f40f
feea05
f7f918
fde706
05ebe5
e5270d
28020b
0b05ff
fcf7f0
1f1dfe
e30963
5f4826
1bfe1b
135910
2d1710
18ef2d
13361d
ffe2ee
281210
1401e0
000919
021e14
0c0fe9
f3eaec
0cf805
d9e7fc
fedcee
f00800
1317e1
eccdf5
04fe17
0bf222
e300e4
e0d4f1
00eee5
f812f8
1a3b1e
1f1efa
010408
e1040a
000605
2200f0
e3fc0d
0c0c0f
```

```
0613f3
ffca0b
e6001e
0bf6dd
f8efc6
fae2fe
ebe9f5
fddac7
f2f2eb
facff0
bef1f2
ccd7d7
e8c2ba
ddd5e9
0706f4
f9f8d0
f40e07
0909e6
041eed
eafd00
17ff0f
ebe9e9
f5ff13
e2cbf7
fcf5f3
fee8f4
d80502
f9fff4
fcf304
e3efef
00f7ff
f9d4fd
fb15e9
f7f10d
ea0d03
050cf2
0001f7
000e14
1e08e3
fcf202
f0edd4
ddf410
0aeae4
ecf8e8
dbe7e4
e5dcf8
01dd00
13fde2
f9fdf9
f60c07
0b0c18
0af914
1a0d09
fcf2f0
ed1423
0431f5
f0e60d
0818f5
```

```
02f70d
f2f91b
1001fd
1ff204
e509f5
0e07e4
f4f2fb
ccd8d4
cd1a14
01e6f2
e2ea08
001e01
1cfaf8
05dc03
e80ffc
0000d7
06f206
fc0500
e4f9fd
00d7ed
1000fd
f9fde3
240910
02ff14
e701f8
161d13
04f6ec
df0f02
e9eceb
daebf8
ed0ef6
11131f
1e1a14
053024
0026e2
e702fa
ed0014
d3fbdf
0dfde9
f3e731
31240d
fa0301
f0e0f1
2010ea
d80933
35010c
010315
0d00d3
dcc4ec
fcf4ec
f0eceb
fbe7f7
07fc12
fd02ff
1b00fa
edfff2
ec021e
18fbf9
```

```
f90721
0016db
eff416
f10833
f320e7
151817
00f526
0c000d
e6f7ff
12f4f4
090e04
1227e6
e9fd21
ee0032
e50d00
27f1e1
fff32f
f7e6f8
fe0212
26e6f9
032206
021e01
24daf8
0af40a
151100
eb0e10
3e1bf5
d60b11
09043b
fcf6f7
ebeae9
04c411
50250f
14f406
e40210
08ef0f
f10ae9
f30711
091110
2af609
06fa09
1512f1
0f0911
e9142d
f303fa
f4eff5
f5d9fc
12f712
0c0e1d
fb0310
1f0609
f6f81e
111012
ffedf9
01e400
04ffff
152508
03fe03
```

```
161cf7
001a19
e5fa19
0500f7
f10805
f8f6f6
23fa12
1afa1c
00170e
f0f7fd
04e4eb
df00ef
eef1cb
f4efff
170706
fa2028
00e9f8
1cfb12
ee0010
110e0e
f601fc
01fbf5
effb0a
0e06ed
110002
0bf000
16ea17
e5ffe6
fd0618
0300f2
0505d6
f20322
09f9f3
000003
ff0d0e
f7fdff
081c11
ebfaf7
f5fffb
03f109
f2f517
1e15ec
2d2df4
e1ec04
090d09
e5e2f7
f514fb
e6e8f7
16ff19
040224
0d0126
0bfd0f
fe2429
11eb06
fcfdff
f8120f
09060a
02ef14
```

```
f0ff07
3c170a
eb3859
032118
2704f5
cdfaf5
f02cf1
eeedfb
fa0a17
f30906
251d00
ff00f3
f404f7
ed14fe
1dfddc
f4e310
010301
e0f502
12d4d4
00f71a
2b110f
ec0029
071d08
eefbf3
0b0901
0d0fed
fffede
eef706
2c2d14
062e0a
160acd
f20419
171f07
090d01
ffecfd
011afd
f90ff5
1ef603
fce70e
0e0110
0421e3
141101
05d700
01f601
1f1ae4
f2dd01
01f236
341011
20f602
0df642
041209
06031d
000106
10f1e1
efed0f
fe1811
0beb0e
05f6f9
```

```
0c17fb
ee21fe
fcf0df
130514
030413
17ea02
00f8ef
f0fc00
090cf9
11f70a
050e18
1d0402
f6fbf2
f10b03
e9070c
08f805
0215fb
1c0a12
d8f60f
12fffe
10180f
0821fb
f804f8
e9ea22
1c0b15
1af903
2e3206
f70317
200cfe
060006
0e00f8
f8ec0c
081900
1106f1
09f318
0f08ff
f5f80e
e30001
f5fbf7
e9e907
070fff
f50cfc
f21c07
e0f82a
0f2ee9
090ce7
fbff09
e1130b
1efbef
25efe2
ff0cff
0000fc
0104ff
09fcf8
020001
ff0b05
0b010f
f511f7
```

```
f5f206
f0fa14
feec05
f9ee09
f200f9
0aee01
fa060f
0a07f6
eff801
070105
f200f2
ee0bf0
06fdf9
000806
08fe08
fd0210
09f2f5
ef0508
f30503
f5f0f0
f0ed0c
f702ee
f6fd0b
0902fa
f20503
f1f705
f40b0b
fcf20f
fdfd0a
f30808
ff0107
f20ded
0c00f6
f10200
fe0f0b
0d0809
09f000
0e06fc
00ee0e
f60710
06f102
f3f50f
04090f
f105fd
ed0aed
fef401
f0ecfb
0e00fb
fe0912
0610f1
fb06fc
08f40a
eef6f2
04fe00
08ef0b
07f401
00feec
f705fe
```

```
ed0df2
eef9ee
0c0c0e
00f1ff
03fd06
02f1f5
f3f002
f5fffd
001002
010206
f0f3f2
edef01
000708
0af0f6
03f4fc
f8eef1
eeeef8
f8fbf2
fc0fee
09f508
0708f4
fcf0f8
fef2ff
01f4fd
0ff4fa
0806fd
0a0400
f8faec
f7fd0c
10f9fa
0c0702
fc020e
f0f502
02ecf4
f60a00
0f0400
edfdf0
f5ecfe
fcfdfc
ef040d
0600f0
0808f3
0d01fe
faf5f9
0f06fa
f804f7
fe06f2
fc0cf0
030c00
f9fcfc
edf8fe
0af509
f8f4fa
fa0806
0ffb0b
060afa
f4f5fe
070f0b
```

```
0b0d0d
f9ed03
00f9fb
13fdff
ecec01
000b0f
fb0cff
fc0906
0a0afa
eef600
00fff8
00000d
0509fd
0deffd
f20011
1212f4
ef0ff3
f31010
0901fe
06f106
f1040f
ff0e00
03fafa
ed0df0
02fbee
040304
05fff4
07edf7
f309f7
fa07fc
09f8fa
08fde7
0d0702
f30dee
ef02fb
faf204
05ecf6
0af00a
fd05f3
09f7f0
08080c
efe6f9
050fef
fbf2f2
f00b0c
05f4fe
ecf803
f40708
0f09ee
0cfefc
f90ef9
0cfcfc
effdfd
f9fd0b
f800f2
f6fb0c
fd07fd
090ff2
```

```
0b0d03
fa0008
f30d01
0500ec
ef00ef
eeee0b
0af300
000fee
f7f90b
fbf5f8
ecf4f7
f5fcf2
fcf5f4
f40002
10f8f0
05eceb
ff0b08
fefdf1
ef0ceb
efedef
0609f2
08e7fe
0509f6
03fbf2
ff020b
09f5f8
0a0f02
fef4ee
f40700
0cfd02
ef090b
0ef4f9
fcedf7
0905f8
040df2
f70106
0aeaf4
effa01
f5fd0f
0d0007
09fdff
03f2ed
010a04
fefc10
09f7fa
f7fef6
f4f004
f60010
f410f7
f609f1
f40ff8
f9fff1
ededef
f304f7
01f00e
080afd
0ff603
f810fb
```

```
08f3ef
0b0508
fef709
fafb0d
0104f4
fcedf4
06f3ef
f20204
05f501
f507f2
edf5fb
f2ee08
0ffdf9
fef4ec
000bee
fff6f7
050405
f60ffb
f4f4ec
f00af3
f10bf9
1105f4
05f80d
02f000
10f8f0
ef01f1
f001f8
01eb0c
120ef1
0000f1
f0fcff
04f1fa
f7f90d
fcfdf7
0ae8f9
f60bf3
08feef
0012ed
0b0502
ef00f0
ed05e5
f3fdea
f20bfb
fdf1fb
edeb06
fd0efe
f2e90b
0300e9
030efe
0af602
f10df2
fb0df9
e807f4
fb0010
f5f2fe
04fafd
0af7f2
0cfe0b
```

```
fdf600
0ef6f8
eceffd
fefef6
f9fcff
03f1f0
f0f704
01ecec
ecfe03
03ebff
02fb0b
fa0109
000af6
0b01e6
0bf003
fceefc
f6fa06
f8ff03
f500f7
00fb08
fd08ed
03f4f9
f602f5
fdff04
f506ed
f6eb00
010bef
eaedee
0f01f9
09f5f8
f7f504
02f50c
ece502
fdf9eb
01fbf9
fbfbf1
fa0c00
fbf3f7
eff509
fb05e1
e7e8e7
fcf3e4
e5e2ee
0b07f0
f707fb
f9e5f1
0803f6
ebebf6
f6eff5
fcf8f1
efe900
06f6f8
03eced
fbed09
fbf6ec
f9ea00
050107
fa0bf3
```

```
01f60e
f20c06
05f414
f5f405
ebf2f6
ef09e9
0c06ed
0d00fe
05f9ee
f8ebe9
0909ef
fbfa0b
f5ef06
fb0eed
02f4e7
f80df9
fd03f4
08fd05
fde900
0904f5
f30509
030cfb
0306f1
ff0f08
f7f401
020009
f70cf8
08f306
e7fff5
f5ecfd
ede8ee
f1f908
0f00f6
020e07
fdf1ef
0c0002
03050e
feec07
fdf6ff
fe000c
fd0cfd
0d06fb
0e0a0c
f1fa0e
e5edf7
0ffcf2
0af006
0000ff
fd1106
000cfb
00090f
00f5fa
0ae900
e6f4e7
fbde02
0c00df
1b0705
0a0012
```

```
041e26
1c130d
f4fb0e
f705f8
ea0a00
05f5d6
0ee7fc
00fdfe
0900d1
f3ed0e
120515
1e1206
12151e
f0e401
e6f6fb
0cfc15
fa012d
19fcdf
f60a09
f014eb
162003
091514
3d4d2c
fa1314
00f8fb
f304fd
ef1c20
0800fc
04e611
e7f300
ff21fe
1001fe
d50109
f4fef4
06f8fa
1b0a17
212928
17211f
d8e716
fde2e1
0f070e
f70e10
e100df
f600eb
dddcf9
e4e8fe
21fe00
f6fbdc
0afe1d
190018
24fc03
d71425
ec06ff
eafc01
f2d3d0
f1212c
14fcf7
0c1206
```

```
0afe13
d8ee03
061e0a
36e2f9
15ff16
0d08ed
e7e1f2
fb051d
031ff9
fefc0d
090000
101606
f202e9
f2ee16
eae8f9
082306
f00e01
f901f7
0a09f8
1c1418
180a14
f31d10
e8dcf1
00e2f0
f1fc0d
121e0d
1b0119
0210fa
e809fb
fb0be9
fff7e8
060e00
09fe06
100402
001417
140d17
030a21
0000fd
f5ecee
dbe4f0
0c13d9
f4fcf9
f61108
070005
f90603
e7e0ee
051bf5
08ddef
01f217
faf104
130722
0d0803
151213
05f00a
f40bf1
01fddd
00fcf8
0e132f
```

```
111bf6
000401
2c2c10
000be4
ff0216
dfec08
00e5f0
0000ec
fe1f14
13eafc
101324
020103
0bfcfa
e4e3db
c8dcf9
130de6
341f0d
d6fa06
1b321a
e6f3f8
030afc
f3d4f5
05fdf8
17f9c3
0a0111
f7fe13
0310e7
f8e900
070707
22e61c
251f13
dede2e
e8eb00
6a160b
004454
354e0a
0b12e8
e03218
2c1b09
faff12
ff323a
01ff06
19270b
18f805
0b0c0f
f4140f
0bf6e7
f1f315
1d00f0
fbc006
bfb9c1
f1fb02
0eeeeb
dd150e
1bf600
12020b
ddd544
04dae2
```

```
1400f1
f5ebf5
0f420e
2c2b07
03e8f0
11cffc
eed8ff
faf2dd
f2ea07
1400f3
ff2c09
2e2904
11f6fe
f50d04
c3d6ca
c91312
efe6d6
0200f9
2148d4
edcef9
03eaf9
0df7f7
cadae2
e908df
21fbe3
f5120c
0ff3ee
1df9f9
02f0fc
f810fd
eefde2
c51109
fdece2
f224ee
cfdad5
f3fd04
0dea0c
25f4fd
001124
24181a
f0e8e1
0cdbe4
f4001a
0d0d06
f2fa12
05ebed
de17f5
ebf400
11e40c
f3fa17
00ed00
10e9f1
0e2117
1033f3
1b38fd
f400fb
ce0415
f6b1bf
```

```
294be0
010026
100b15
03170b
ec0ce7
f00d11
f8f6fd
201300
030321
1a04f8
0cfdf8
f406ee
0b09f6
e2fff5
f91df0
f0eae4
dc0811
cbea2e
f23dd9
ea19f8
ed06ef
f8140d
1c18f1
c4fc04
00eacf
0000fe
0bfbf0
fb1903
0c0ffe
faf0fa
f9f301
1ef8f2
2c0204
ff2022
200201
f7edef
1e19f6
09f7f4
fe1328
0bf721
f70719
f70611
f301e4
10e1e1
15f533
1ff9d8
ecdff5
e50308
04dae1
e4040d
1d03d3
ee1f11
1908ee
e6f606
180e06
06e90f
d3f832
0106de
```

```
ff81e4
0e1a2b
210fed
f50100
1c13f5
020eff
1f1d10
1c1201
f40a0f
030a04
0d1afe
f9ea08
1509ef
f8fcec
d912fc
f5efeb
e3f411
180c01
08fff9
e7020a
1b01f2
fb110f
fbe9f4
e4e7db
ae0c08
f21dd4
f700cb
322313
02ecff
221c08
1b0b18
010912
f60c0a
18ede4
eefd09
180be0
fef821
23fcfe
eee4dd
ca072b
00dbd9
dcd304
01ff11
0d070a
0b05ff
15081c
070e0d
fd09f6
0604ec
10f1eb
120d15
07f40a
1606fe
f6e505
f7fcf9
0903fa
e6f312
0a08ea
```

```
150f0c
091907
09fa13
060313
03f811
fa1000
031003
fafa1a
151e12
0ff1f3
efeb04
0011fb
16120a
0cf227
f30704
fae0e5
f10408
151015
1613fc
06ea1a
0a1803
e8f4f3
0911f4
1f09f1
0b0c0d
0d4809
10f40f
f6061d
1303f2
11eff1
0c240c
1120f2
19f2fe
16f201
ee0712
1213e7
f80f15
edfeff
01fafa
0cee0a
ebd2e2
f213e9
fb0211
000007
190810
e2f115
f613f8
01fefb
f708f5
eb0a04
ee0809
01ebe0
011115
ef0004
ef11fa
e3f7f9
06e4ee
fbf6e1
```

```
06fde3
f2fbe9
242500
1c2811
161ef8
151414
160b1d
0befee
042222
1d08f1
f50a35
fef9ee
05ff05
0132ea
e3e608
002437
4c12e5
f1fb16
23742a
ef01f0
e1f8fb
02e1fb
e3e30e
090c03
f0e5d4
ecea0c
0410e9
03f5dd
081220
1711ff
09fa0a
ea0518
1ff0f8
fc0101
0a1df3
0af718
060dee
fee40e
0b1f0c
2508f4
2c1129
fd203a
3e0005
12fb35
f8e418
f6eff2
1efbfe
00e713
fa1d09
110e06
f4eefc
d81f24
fd01ca
00f403
001df3
f80e22
121a10
27e9e5
```

```
de0d39
0a37fb
250be2
150709
fde9f9
faf8f0
f0e6f8
f40403
060b1f
fb1528
f109d1
180af6
131c0d
060923
13fc09
06fdff
f6fde5
0305fb
ebf2ff
feee14
fc090e
02e7ef
e40100
f8f9f0
eeee07
02fffe
eae9ed
0ff7f7
0d0e0c
fc0b02
e5f7ef
f20df9
fc140c
050915
1c021b
f2daef
e30b10
f0f9eb
04de00
f40c10
00ece1
e0f704
1308fd
ecdbf0
02ee13
fff7ee
0508f5
f20303
0bf9eb
f7e208
0012f4
fc0fdb
06000c
0b0e08
080015
f205eb
080018
04f9f8
```

```
fc111d
0afd25
2709f7
0000f8
07f202
ecfefa
fc0008
f3fff0
e60014
05ea0c
e61013
2500e5
2502fc
d5d801
f522ea
e5d8ef
01130d
0df0fb
f71c10
07f9e5
e6f913
1405f1
0c052f
440620
fd1916
173914
0b212f
1df101
ddf80d
161109
20ff0f
e1e12a
26fd15
2206dc
f9ede6
06261f
3823e2
090114
18f0f6
eefdf3
fdd9f5
e4e3f7
faf3b8
e3d707
10ed00
f1fb04
ff0911
ebf81b
262003
f7f1f3
190b1b
f70e08
fc07ca
e7e207
01fcf0
d5051f
fb200b
f2de00
```

```
222500
d6072c
423a07
060136
efec24
13190a
17e9e7
d0d6e4
d9fbd6
fc12e9
1bf3fe
0b0516
c1140c
0dfef5
e9dc0d
f70916
0f00ec
362918
150b3a
1e202c
0bfc04
fe0be8
e103ec
050012
0be4fc
ef0cf6
f30ef9
fbf004
0d07f1
131df3
19f70a
03f9fd
142d13
0d0603
1af7ec
d8fe22
edf102
03f5f5
eb050b
1a0ee4
e5f4fe
0f05fc
d7f800
0d12f0
f4eee7
f303f9
ee0dfd
eb000e
1c09ee
0cf3ff
00021b
11f50b
2dfe04
ea0217
15d017
f6f91d
1c22fd
06efed
```

```
0c1e02
d0e2e6
ff06f7
f4ddee
fffb09
e3fe0e
f5f3e3
d50612
f7fff5
d4e1e6
fd0efa
f5edda
000202
fcfd11
e918fa
19f2e3
28111b
fb0eee
1b12fd
14cbf3
fa0005
0000ee
ebfcec
f6fafc
f3f50a
0d04f7
f701fc
fe02f3
0806f6
f3fafd
f600f8
f9f201
faff02
f7f3f6
feecfe
08e9e7
fb060b
f700f6
04f4ed
0bf807
ee09ee
0df4fe
0af305
f3ebfb
f40b0d
0ff6f4
fffbff
01fcec
f005ff
fafef9
e9faeb
0f05fc
0001fc
fb07f8
0500ec
f2e6ea
ec06fa
ff0100
```

```
0c08ec
0205fb
0a09fe
f90ae9
fbeef7
f3fe0a
0009f2
fae805
ef02f1
11ebe5
e80000
05fbe8
000003
f7ed00
08f708
07f903
ef13f6
000fef
eefbfa
ecfcef
ebf711
f216f2
07e702
f70600
f80602
0a0fec
080702
fffcf6
03fef1
ea03ec
03eff5
ea040f
0afdeb
e8f902
f7edfd
f6fe02
02f3f5
09f1f1
f1ff00
eee7e7
ef0004
ec020b
ea0bf1
0104eb
09edf6
f304ea
08ffe8
00ecf1
04faf9
faeffa
fff0f1
00fae7
0f03f3
ff0307
f20d0a
f3f9f1
f9f205
ef0106
```

```
09f7f0
02f00a
f6f1ea
f2f3ea
04f40b
f20af5
fde701
f6f5e8
fc00fd
0400f4
fff800
edf7f0
0709ef
fff1f4
f8ebf0
08e800
e8000f
ecfdfd
f3fefc
0aeef7
f5ef0a
f606ea
060507
04f4f8
06eafc
f2e809
f7020b
0404fd
eae9fb
eaec0b
04ecfa
0aeff0
ebff06
fc08e3
f4e902
02fffa
12f506
fafe07
0bf6fa
0000f0
0af308
04fc03
101408
0908f6
f103f9
f5fef6
f9fc0e
fe00f3
000908
f7f60e
fa0006
070b0a
f2f6f4
f3f7ee
f3fbfe
000fff
0b01ec
04f7ee
```

```
ff0202
06fbee
fd0008
01f30c
ed0801
08f50e
ee0e01
ecfbf7
000afd
f7f3f1
03fd0f
fdf9f7
f7fd0b
fafdee
0df708
eb0bfb
f608fb
0b0805
ed0005
fef0fb
03fc10
fbf8ed
fe03f1
05fd0b
000506
f8f407
f309ec
eeed0f
eef8f3
fa0909
07faf6
00090d
f70f02
fd01ef
f9f7ff
08fcfd
f30c01
edf7f2
0f02f6
fdf1fb
07050d
08f1f9
02f604
f30df1
f2eef9
fc070b
ef0002
fdfbfa
fd04eb
0b06ff
ff0906
fcfb06
0d0cff
fbfef2
06effc
f6ed0c
f109fe
edee09
```

```
02fdfd
f1fcee
0e00fe
010ffe
04fa11
f8f7f0
eb07fa
fe0ef2
03f90d
f4fa0f
070610
f5ef02
05fff2
0509fd
0005f3
f8faf9
f8fafb
f804fe
0704f0
ecf7ea
0c050b
f8ebfc
ff07f6
090ef3
f6fb0b
0a0b0d
02fcf3
02f7ff
04fe02
ebf4fb
eced07
ff0ff2
fdf5f3
fd02f7
0d0cef
000dfd
f0f801
f10aec
0b0bff
fd0308
fbebfb
fff6ff
0cff0e
0b04ed
0701ec
06fdee
ef05fd
ef080b
fc00ec
f6f605
0fff04
02f506
f70bf4
0b09ef
00ef0c
f9eff0
09fafa
000003
```

```
fe0afe
f1f809
ff00f3
f2f40c
fa0f08
27fb08
20ffe8
e50c10
0f1803
071d02
e4e5f1
d9220c
00f0e8
070f01
060215
20f40e
291012
ea031b
f827c6
dbecff
09f5eb
0b061c
d8f108
f6fdfc
0702f0
3500e5
e4101c
effe20
e4f823
fef5ef
f7efed
ea0cdd
b5ccef
ec0006
09f501
090ffd
090f06
09fc10
f9141a
291e08
040701
121a10
02effa
162be6
050901
021104
05eae3
e9dbf9
020f04
35010e
ff0ef4
20edf3
f30432
ed12fa
12f9f8
fd0d11
fdec36
110813
```

```
02ebf1
e8d9ee
f41c0a
17efe2
0a11ea
071c3e
121b0e
21fb0c
0ef908
071c0f
0f0700
f6f5f1
27f906
0d1427
252e0e
10070c
060a10
00fb03
e50714
0af8e0
0aef01
fb0414
031a06
0702f9
02f00a
001517
effb0c
f6090d
00f409
f3f807
0c0ce5
1c0b05
0318f8
fa130f
0319fa
fbde02
1515fe
090af8
f2f500
170cfc
13feff
14fcf6
fc0102
12040e
e3f8ea
ea01de
0501f6
f60cfa
132029
fe160c
e90a10
13f409
f9e6e7
0e1907
fe02fe
f21910
0be811
e60223
```

```
f21bf2
1e11ff
0a041f
f707ee
e2fa04
ff14ec
14f30a
11050f
001408
fdff39
00e7ed
e916f1
f8dbf6
e9db0f
1c0fe3
e715f0
02f2ff
000005
fef505
f50afc
0af6f1
0900fc
f605f3
f5e20a
070015
17f4dc
e8ef15
1c1904
1700e5
010018
f4f5f8
0f07f2
141b0c
0df91c
0efb0c
002413
1a09c1
d5e70b
0c0a13
2622ee
041a29
0306f8
f3eb11
d4dbf2
03f80d
f3c3e8
eef912
03d9c4
e5e6f3
fd1618
0f1516
f5e521
fbff17
05feeb
f1151f
f81902
fafcf8
d80604
```

```
fd0609
2d16f3
e9fb19
0b12ec
11fadf
24190b
171e16
f72c30
f9f9ea
02f7fc
f30a07
fd07f9
e2d7e7
f4f8f0
f7e2f3
f00113
310ad5
f8ee0e
f30815
e60503
1efef7
f5fc1f
fc0515
f81a05
f1ebfd
ff1205
150409
ed1e20
0b18f4
daec09
161a3f
36f9fd
02123d
27220e
d8ce0a
17fd0e
ee091d
080e0f
0b0d0f
0c090d
f708fd
18eeed
ef0af0
29180e
020725
040b0e
021212
0f1a15
1b151c
ea0af8
e80506
f7ecea
011f04
02ff01
02180a
00edfb
f8fb11
02f9e8
```

```
0909ff
fd0c10
0e101f
0e0721
fdf3fb
fff40c
151500
051325
21fbf8
fde1f8
e5f5e9
0e0004
f4daf5
101118
20fff2
ed1803
fe05eb
f3e900
fe0020
070902
02030b
fd1323
1406db
02f5f9
0d0dee
f902fd
fcfd01
e0fd25
2be0ee
ede700
002607
090100
e9121f
10fe09
18fcf3
0000f9
34fafb
d5e51d
2e17f0
dcdaee
f6081f
060005
f3fa09
231de6
0ef7e6
c8162e
23fdde
e0f128
160fe8
ecf0e3
f2f714
17e5ef
120d1f
b0ddf4
01fc0f
dad103
1d0208
1b06e4
```

```
141e26
07e31b
110c1a
02280f
131221
33d018
ed0b12
0d4200
2f03ee
eaff37
0c3216
1decf1
c5d303
05070f
efdade
e4f904
07f2f7
23fa0c
02ede9
1d04fa
d3aae9
0307d3
bccfe5
f30e00
0ffdf7
effc1c
0ee413
0d071a
e000f2
0bf7fd
3adb03
e5cb21
172ac2
0b14ea
121519
200f1e
180535
f0e5df
cf00ff
eb0c0c
e8dae7
15eeff
0409ee
e6280d
0b0f0d
1109fa
cdf00e
340cb9
cef1f1
09e7db
011507
270816
211d04
0effe4
e6c313
f716fd
03ddd2
e00003
```

```
00eadb
fa131b
0904d9
001911
bebbee
fd06ff
e2cbe0
f20205
fbfdf8
0af3f4
f8f3eb
fe0a10
141b21
11e3d5
f31a1c
faeeef
e8f5fb
fe0907
e2cdea
f50ff9
f71402
dddd00
18f5d8
e3e000
21fae8
01f4f4
050a1b
1eed04
e8e325
e33818
28e0f1
f2e330
1928eb
d7e7e2
f707f0
f0e8e8
fdee18
ec20ff
110bbe
d6f20f
0c06f7
e4e4f9
eb14e1
07e400
11f504
d3f406
efd5fa
0fe8eb
03e8ce
f61908
000e09
05f805
1b0cee
0000f2
0000e8
fd07fe
10f100
0df0e2
```

```
01f706
f7f7f3
0b0f03
04f4fb
e4e8f6
f1d8d2
f101e9
f4eee7
010908
1bf604
f20afd
010000
da0005
08d3c7
fcdf2d
4b47eb
0110e9
101815
f6fefe
101be6
f6f712
011915
38e4b8
bad826
0425d4
10faf9
0de015
0e2c17
0b1c04
13eb0d
f9f9fe
f21b06
00e0df
e2e6fa
0bf6d0
e1f10e
00fffa
d3eb03
0ffcbb
fac22c
32421c
020cef
06f0f4
e406e2
1c05e5
de03f9
03fff1
0cfb11
1d0014
e4ec08
1a1714
15fdfd
040e21
dafb02
1ee800
fefc0b
f1ee0b
f2faff
```

```
ee0ff0
fdfdf2
e0ef04
091105
cdecdd
1600fc
bfd020
252b20
32f2eb
0f3624
e11d13
0d0d03
d5f018
05e9e4
ddf9fd
0a010a
23dff0
ec130e
f514f4
e2e802
11f804
d7c204
191cf9
0aeecf
d81807
f30910
0d0edb
f3f50f
01fefa
f1e4ff
ecf0f0
f7eff6
f5ecf8
0b01ff
f70408
15100b
edfc01
f601ff
efd1d7
0502f6
0c0dfb
f7dbff
001b2c
09f412
e90c08
eb0900
edf6ef
f2ebf8
110ae9
0cf50c
f81107
f7e4ff
10fcf2
04e4f6
d8fc0c
f6fafa
1cedf0
e5eff4
```

```
eb0801
eefef6
120915
00fcf7
dbf70c
11c1bb
cc1e1a
2afbf8
0dd309
ecf314
0a0503
0406e8
f2f41d
0000e4
fc0a00
e20bea
fb1af4
f206ed
0107f1
11ef0b
eef912
f410f0
eefbee
d9f3f2
edea04
03eff8
ff02de
e81101
0512f8
fdf116
0bf40b
1ff507
0e1d46
29e400
fbebf6
161200
f4f901
1115e2
0207f0
141dd9
ee1d20
f30b14
2ff7d4
e021d1
f60d09
f4f1e6
ecd316
0fe518
12100c
1ef3e7
dd000d
e4fc0f
f9eff7
fd1d0a
ff07fd
00e518
29fb07
f5fe35
```

```
1219ec
fdfb11
f6f621
e6eafd
fd1107
170c02
f8f4e3
11f0f0
283214
10150c
2828df
ff180e
1a0b08
080716
02d1ec
180617
0907fc
fe2121
13e407
02f006
d63108
0b1804
fbe5f2
0000ef
f810f5
353905
eff41a
141906
0b0a10
f61afd
cafbf9
ea0608
f8fbfe
07faf2
ddf020
f41113
02fc07
0ff7f2
091e0b
051003
edf5f6
f2f60b
120a16
1af7e8
e700f0
fc04fe
1ff3fc
17eb0a
0ae300
f8f8fe
f1edf6
0de2fd
f41216
1210f9
131310
ff140f
cbfbf9
15e2f7
```

```
260a0c
e3130d
fc0a08
00ee0c
24fbf6
140e24
1a13e7
07faf0
f7f912
1901fb
12f6fa
eb0bec
ea0015
fdf900
e90b05
fb08ee
05ef00
f2f30e
01efe6
070ae5
1a15f2
000bfb
faf30f
e0020c
050fe9
fcd6f0
cf14f3
e31222
f1d9f6
1812e8
f0ff1d
000002
0eedf0
1313f7
ef1011
08f2ee
e7f0f6
1befe9
0bfe20
1d2124
ebe903
020614
08fef3
e90d0b
f4f6f7
f5ece4
001811
f90824
ff0c17
010702
080009
f5ed01
e3e919
dd1ff5
18f0fe
020214
2109eb
1ff5eb
```

```
d6f025
361700
12f500
fdea14
e4071f
05efdc
dcb7f4
f906ef
0b0d0a
110ff8
ea0313
fb0418
0f0301
f5fbed
f304fb
0b01f8
0c0006
0d0c04
0c1dfb
e3f70d
e210fa
03fff9
030a0f
13fc10
05060b
0517e8
122428
271209
f6fe00
f21111
dc201c
22ddfe
05fa0e
f603fd
f60f00
13f217
fd0403
161ef5
1419fe
1a0be6
0f011f
fbfc00
2a2207
f0ee00
ea0a10
eafa00
11c2bd
f0c73f
2c2428
f20ef6
01ff00
eff000
e00903
f5080d
f9f8fa
3febeb
f60630
fcf300
```

```
1a0cf6
0c0c13
0d130c
e7f7df
f3eb11
ef01f2
1c0a1b
fb110b
0305f4
f5021b
00f100
f00001
eff5f0
fb0e0a
0f21fe
150b10
f909f3
121604
0dff0c
1208f9
fee0fa
f1d9f3
0cfc0f
2e0522
f00025
ee0f1b
fbf7f1
f1f0fa
0d15e1
f800fd
ff1604
183e0b
fff50d
16000c
000000
f7fa19
11fb09
0ff600
fa0411
edfff9
0203fe
edfdfa
012518
0b13fd
010105
fb0100
0df7ed
17ffef
0b0cf8
00000e
f7fa00
fbfc07
f3fcf4
f7fff7
f9f306
fcf815
f002f1
f3fc11
```

```
f50ff7
0604fa
04ec0e
f60df7
fef2fd
f30009
0dedfa
fe05fd
000800
0b0802
10f5fb
060700
0af8f2
fd09f4
02fafc
0bf3ec
0200f3
f8f109
eff107
0b00f4
f4fdff
06f910
fbf900
fe02f1
fb08f1
f309ef
fbf4f5
fd1000
0f0eee
04f0ef
06f600
ecf505
fc05f8
0f00fa
f9f7ed
0af206
fb0900
08fc08
f3fcfe
eef002
fbf3fe
0600f0
0d00eb
f20107
0ef607
0aecfa
f60f00
02f1ed
fbf400
ff0905
f9f408
f301f9
04f4fb
f9f5f3
020b00
09effe
ef0a0c
0c09f1
```

```
0e06ff
0ef8fd
f2f602
0b01f3
030aef
03eff5
06f101
fb0ef3
f00410
fa0e03
f506ef
f6f8ee
f4f807
0a06f9
0303f5
0c0801
090ff8
04030f
12ffff
eceff1
00edfa
fefd00
050def
03f600
efeff7
07eef6
ef040c
f00806
fd09ef
fdfcff
edf7f1
020dfb
ecf7ee
0406fb
ed0efa
ec0cf5
f403f9
ec0704
f5070c
090406
f40af1
030c02
080b00
fbf9f5
f2fdfd
f609f3
ee00fa
f3ecfd
fef302
fbf10c
fefe09
ff07f4
eb0200
f20ffc
0303fd
f70e06
0b0909
08fbfa
```

```
f1f504
0c1001
eef500
f20a0f
010c06
0ef1ed
f3f6f4
f80af3
fc00f7
000005
02fa10
00ef0c
f400f4
00ff00
00fef2
f7f3f3
09f103
0401fa
0c0a0b
ececfa
0e0ff2
f30df0
00f007
fa0df4
fafef6
fa02f1
0508f6
f404fb
06f0f9
0c01ff
00ecf4
f8eff3
ed0002
0eeff0
f5fef9
f5f7f8
02f300
0c0001
fb0bee
0cf2f5
f61110
fef7f6
0d07ec
ed0e0b
f9070a
10fb04
f0fdf8
f9f1fd
f6f7fa
09fb0a
fbfb08
040c02
f7100c
09ecf5
08f2fe
05f60f
08050a
fb10f9
```

```
080bf7
f1eff2
03fefe
fffbef
fbfcff
fe0203
f2f00b
ef0b08
00ec0d
f8eef6
f6000b
fe0af8
060cf6
0dfdfe
0e0f08
0708f4
f8ed06
0f0400
f307f5
f8f408
0afc07
020505
0705ef
fbfefb
f6fbf7
ec0e10
f2f6fc
f30d0f
f50c0c
f6090b
0608f1
010005
0c02f4
f80004
edf003
f60c03
f8f8e7
ef02f4
0305f1
02fe08
01ecff
f3feed
0bf5f1
050100
09f2f4
f7ee0a
0a090a
ec07ee
00fdf6
08f6ee
0aee04
0afff0
fdf3ed
020bf6
10edf5
0b06f1
03fd0c
ed0307
```

```
0600f7
ed0705
f8f8fa
f0ef01
f9eef3
f7fdee
fff9ec
08fdfd
f2ef0f
f5fc08
fcfc05
f9f8fa
ecfdf6
06fc07
ecedf8
f6f103
0efdf2
08fa07
0aecf6
eefcee
f3ef0a
011112
09f40b
fd02f3
080009
f504f5
f3ec04
0000f7
1103f1
e0070a
f600f9
ef0bfa
0c19f9
1a1029
f8f316
f70009
e8fdf6
031512
efe9f8
e90007
1805d3
170b04
070100
07010a
170004
040cfb
0a1bf5
06030d
10f609
1d0e0b
03f720
130eec
fc0305
0b19ec
f40823
09fe25
160e16
000acf
```

```
efe9e9
02202c
fa0cfc
f0e028
1c0606
040de0
f9000e
03f70c
04f900
0f0bf6
14fbe9
07ff25
09e800
1724f8
271cf9
0c000d
11ed10
e2e704
f6f6d8
fe130e
f310fc
05f90f
030623
f4e8eb
150ebe
181c10
4af6f8
11fc37
0ff5f7
03fa12
ee1de8
f201f2
06f8f7
0efcf3
ed0c0d
21162d
f2c70e
eff60f
ef05eb
0dfbf2
e5f012
1a1508
061e2c
0f3edf
bdee01
16f100
d81115
02f6da
eaebf6
eb0413
180c12
fe0902
0d0df3
0528ff
f5f606
0f0f0d
09fbf1
000cfa
```

```
ebf7ee
e5f1e1
fef4fa
faf9f5
0ff408
f4e70f
fc0a0b
09f3fa
17050f
140df9
04110f
fbf6f3
f224ff
fd1100
f90b08
d2e2fa
f5fbe7
f40f14
2802f4
f70b0c
e4ede7
130812
f5f50e
27f90b
1dff01
ee11fa
f809f8
060000
00140c
000e00
f7fc12
f60b12
ee07ec
e5f903
0107eb
00f216
f20ff9
ee0115
001105
06fcdd
eeda0d
d9000c
fde50d
0112f6
f912fe
000005
f102f8
e90b0d
f8e303
f4ddfe
f912fa
13f9fb
10020c
001f00
ee1100
100a0b
edf3f2
e703f0
```

```
0e0fd7
f005f8
fe1015
fff31f
201404
091639
ec15f9
01ed05
eafcf6
f61718
f5fa0a
f0f309
05fafb
26f2ee
de2c07
e50808
f40dfa
e6f2ef
ef0ded
0308db
d3daf1
ffd901
1b0216
0501e1
04f212
eafc1d
1b08e2
010ae8
140d0e
fd1407
12f308
181dea
05090b
fefa14
0903f8
f5ef0e
eb0506
020bfa
f519ef
0c0800
1a0021
1d2ee9
1e03e6
e9f204
090f0b
35fa1b
00eefe
dbf912
fb1f0e
1bfded
030216
f31120
fe0df3
1d10fa
05e925
f92015
2618ef
050b04
```

```
dae919
2d321b
1207fb
090d34
fa0e05
fef51b
f8040f
1107d6
efe6fc
fb0908
0704f9
19f702
f51107
142c17
ed1409
f9092a
08f8f1
0004e7
e9f4f8
05e8f8
f60dff
060709
05f609
f3dc0f
050de7
f7edde
fc0104
15fb10
0a1402
080906
f31216
1a160c
f1fc1a
e80ef1
1309f4
d7e806
1b0618
000513
09160a
06f0fd
f0c7f4
dd03fd
0811f2
e4f7f1
fe0304
f71621
f4f9ef
1c0c04
fae607
ff010b
f006e6
07e7fb
f70bf5
f50802
0ffe0a
dffd2e
2c0df3
e707e5
```

```
edfb09
f91709
02fbd7
010502
0204ea
000022
05edfc
e60d17
f805f2
f5fc0f
0009f9
ffe50e
0d061a
05f807
d3e4ff
161c04
f0f9f6
f20906
180209
e1f9fe
0c040c
01ed1e
180c00
f5000b
f811ed
c9cdf1
fc1718
f704f6
082927
0e0cfa
0beaeb
cefafa
fbdc0c
f6daf2
15dc10
14f4ea
ddf9e6
f1f705
f7dbeb
e8ffee
dfff10
0f04fb
e1041c
ee0300
10ffee
f7fbf0
08f609
1d1b0a
fdfd00
011ded
ebd3da
0bfc19
040813
f7222a
1616dc
f6f3fa
d50404
0d02e9
```

```
e6e3fb
12090f
f7e706
e80e0e
1516fd
04f005
f5080a
f7eeef
260ced
f4f121
e3e821
f4f0e3
0b0afb
1718f2
052210
f2f118
f7030f
00f0f8
f60e16
0609f1
08061c
1aff0e
08001d
f1ee00
0af50e
13f20e
f3efe6
0a00f9
f8feee
2a00e4
fbf612
060d08
d2effc
020d11
faf3df
022313
0afcf7
0005e9
0e0415
ec1506
1ffce9
eaeffe
0e15f8
1206fa
fefb02
1dfcf5
fdff03
1df306
fe0002
0de302
fffe05
1d050d
f40d0d
d4f60d
2413f9
1b1a15
e0ed1d
000eea
```

```
e8dbef
fd0907
17e9ca
e1091b
0c07f3
e80c05
f1fce9
07f7f4
fff900
0b0815
ff0ee4
05edf4
fb0201
04f516
2300f2
d7fdee
16e2df
01d605
1d0c11
090ff5
fe10f9
210200
06090d
000011
ff05f6
faf9fd
fd0bfa
ff0300
0cfe0a
f609f2
f5ff07
010108
f2faf2
ef09ed
01f2fa
fdfff3
efec04
0ffef5
09edf9
f30a0b
f406f5
f91108
f603f1
09f803
f80af2
02fff2
010705
f1fa0d
ed0905
f300f8
09f7f5
0ef4ff
0011f2
f7ef01
000506
f0040e
ef060e
0a0c02
```

```
fc02f4
fd01ff
0402f6
fb08f0
0ef900
fdf1f3
f1f6f4
0f03f8
090605
0200f6
f20e09
03ee05
080700
0fecf2
ef08f1
eb0900
0505f8
f4f0ff
ff0307
02eeec
fafeff
f5edeb
08ee08
fcfb0d
0aeffb
effff8
0407ff
fbf9fe
ee05f1
fbef0a
f4efec
f001f2
1110f6
fa07f5
00f40c
ec060c
fbf700
050c0a
f500fb
06f802
08fef6
0dfb06
f40f0f
00faf1
06fdec
effaf3
0707f4
04fc01
f4faef
f1f70a
f7fbf2
0e01fb
f8fff4
f30e0e
f1f8f1
000ff8
f3f30c
f4f40e
```

```
f70bf3
fef7f3
050aef
f8080e
06fcf9
f607ef
090d0b
fc0009
f1ed05
fe06f7
00f6f2
000703
ecfeec
f5080f
f707f5
f3f0fb
f5f1f4
0d0a0c
f6f3f1
fc0ef8
eefcf1
fcfd06
0e07f9
fa00f9
0400fc
eff308
fe0201
ed0dfa
0c06f6
0100fb
0cfbf2
fcfd03
0f0ff0
00f5fa
0101f2
060009
fefa07
fd11f4
060afb
f306fc
01f0ef
00000b
e8110a
ec0df1
f7ff04
1003f6
110f13
03dfee
f8100c
06f208
faf203
130401
08feea
f10ef7
021415
15100b
1a1331
0f0508
```

```
02eaf0
dbd819
fdf91e
0be7ce
cef20f
ef1bef
f4fc00
0d1ede
f6dde0
df0702
05ffc9
d3bc18
000603
ebbca4
b4f20f
200e02
179ad7
0c0c1c
0503ef
f904fe
12160d
f70e1d
131a0a
1002e6
eff2f6
1c391e
08ecf8
f4f50d
1f04fa
f9ea08
e9161c
ebfd0f
f10ae6
ecf3f8
09fbfe
f61700
effd01
f9011a
17e6ed
def2ef
f5fae9
e3f2ea
0a05cf
0f0e04
0ff911
041bff
04ed09
fbe816
e7000d
49f0ef
01f53d
1a0618
19fb0a
0b171b
2c031e
0a4457
1a1903
1313fc
```

```
c5ed11
f8140c
fd0cf8
140d11
f9fe09
17f70a
0c1004
09f4da
02ed16
0402ff
fcfcf5
f4f910
071208
221203
0b0019
f001fb
1b0108
14fbf9
0de604
05ee03
0d11ff
010406
0ffc04
131402
03041a
fb0206
fc08f9
fd050c
120606
06060a
05171b
07ed09
f0130a
e81307
15fcfd
01e0ff
080810
1ff7f3
0803e7
dce502
f6f003
fcf8f1
0419f7
00f30d
0eec0f
14f6f1
0efef1
172b00
22ef03
02ea19
230012
ea0a11
03fce9
0ff8ff
00fc15
eaee1f
02e9f0
fdf0f1
```

```
eee205
00001c
feef05
0df5ef
070c00
03f2fc
f2f609
f00c0b
ee010a
0002f4
0df2fe
01f7ed
ec0909
f6fa0c
f00903
ed0a00
fa10f6
fbeb00
0bf40b
fbf9f7
03ecfb
0bf0f2
000500
f0ff0a
02f40e
000f01
f002fb
fdf3f3
0206f4
eb0df3
0cf2fc
0af2fb
f20706
00040c
f8fcf2
f601f1
0a01f8
0cff01
f8fdf5
0204f6
f2edff
fdf40c
f306ff
0f00f5
0a050e
03effb
f2f00b
fcf90c
f70ef0
f203fc
06f3f6
f8f2fd
ef0a04
080008
f4f309
fa08f3
f6f80d
ec00ef
```

```
f50ff8
fdf20a
ffef00
fbffef
f4ffed
0c00fe
0307fe
f6fc02
000e09
f8f205
fb0ef6
03fc04
eafa0a
eef000
080904
060ef6
f00c0c
0ef707
f308f8
ffeff3
f5ec00
03f7f5
010508
f5f3ed
0e08fb
02f402
000f03
fbf709
0700fd
f1fb08
0aeef1
fef7fa
02000c
02fe02
fc0400
fef6f6
00ec0a
00f7f6
f1f700
fdfa10
eff4fb
ec0900
000bfb
0cf00c
f40b00
fafef3
0af1ff
0ff4f4
0e09ec
f7000b
f5020b
0d0ded
0df9f1
f1f0ec
0d01fb
10edfe
00efed
f4040a
```

```
0708f7
07f306
f20505
0bf610
fa0004
09eefb
f1f7f9
ee0910
f1fcfa
f907f2
f8f1ec
f10907
f4080a
f6fcf6
f000ff
07fd00
f8f5fb
06f205
0a10f2
0000ff
fb0cfa
011609
1a18fd
f2140a
0205e1
fbfc24
0bd8eb
fef7f1
fd00d4
fef2f9
130205
1916fc
0cf1fa
011e24
edf30f
130e18
f4cadb
16fa03
01f32a
2305f6
fc01df
deebfd
05f600
3b2634
11263b
00e3ff
d9c0d4
e10b1c
fc0cf0
28eef5
b3de02
0d29d2
e09cd0
ed09e5
02eff9
eeec04
f10eeb
f1f6f7
```

```
ecec0b
00f0fc
cacfe8
f5fec4
0aec03
101b05
fcff05
eafffe
ecf2ed
1e02ed
220e1f
032114
000eef
140c06
381e0c
fe0538
0f06f9
f112ee
d9d000
151402
09ecf4
190bff
03f701
021d00
001607
e8d6fe
eefb00
f5dda4
0b0c0e
292001
0bfe24
2e3afa
fc0efe
f109e8
18fdfc
121af3
081334
0607ee
0c0af1
14fa08
f0110b
f5e6ec
f21012
f3e2f2
0604f1
fce8eb
05f3d4
f41606
dbefff
13f0e4
151318
fa0c03
210927
0aeb09
d7eefe
fef6f8
f71500
f1ff1d
```

```
0f161a
f20109
f517fa
d8d907
faedf8
08ea06
38180a
fcfc2d
110dfe
ea040b
0acbd3
f50a0b
0714cf
e80122
e01800
15eaf3
d6f300
0a27e5
eeee0c
fbe7fc
f8e3e4
e9e0f6
eceff5
f4ea0b
fa05ff
f51004
fe0302
feecf9
d3e3f0
f6eef2
f8df04
f7e51d
d5f0ef
220df9
fa0b20
361bfe
d50617
0000dc
ebfbf3
3df0ee
fb011d
1a17ef
f50912
111d00
e4f1f5
1115fe
201a02
e1dee2
13271d
55f80c
f5214b
0b0910
f1f500
070601
c40e0a
110bd7
fcda32
492cfa
```

```
161e1d
1d2404
0a13f7
e20020
f4f7ec
0a012a
18f3e3
feedfa
1ae404
fdf21c
231001
0d1b26
5004fd
fe1e13
081405
02fb2c
0d09f4
f6f2f6
08030f
ece10c
00061c
d1ff02
140715
f5d51b
2e1a00
031a16
0b1b18
fef605
f6fffd
d5eee1
110f0d
da0607
dc1de2
edd0f8
df0122
f5fcf7
ef09ec
f60907
f91af8
121b0a
e8ebe8
cff80a
f0fef8
0b16e2
e6fff2
edfcfb
d30dff
0ffbe6
fcd40d
041416
2c25fc
0fe309
00fd10
fffd00
f40c0d
00f8fb
2afef9
f5070e
```

```
061af8
e5f9ed
01f205
fefbff
0311fd
fbd8fe
25fbfa
1afdd7
122008
0f040c
0e0e0d
f1ef08
f50005
fcfbed
f602f7
0848e6
fbeef9
1a2c36
0e10ef
0e0cf6
f002e6
f205fd
fff607
0b0f0d
fe14fd
2bf80e
ea0401
13f6eb
fefef0
0113f3
29fefb
f4f213
0a4527
0b01ea
0b0707
0eeafd
091b05
1d0ae7
2004fb
f6e9fe
fa1fe5
fbed03
fe051d
f6090d
fe0c14
fdf704
0000f1
03f1e1
ed11fe
0d2104
2f3d0d
1206f9
040b10
11dae9
f5faf6
0000e7
fdfe09
0cef09
```

```
0afef3
0efcf8
0af411
fceaec
fe02f2
f90812
fbf30d
f40102
0608f3
ff00fa
0e08fa
02f2ee
fbfef2
0ff309
fff3f6
0cf3f8
e9f5ff
f5f9e8
f205f3
fbfb01
f807f7
0dfd06
0e0aff
f5f1f5
0cfe00
ede9ec
0cf307
09fc00
0501fa
fd0cfb
fa05ef
eff00b
ebf30e
ef09f4
ed0c07
050b03
ff0afd
08030b
f30100
fe02ef
00f1fb
070efa
0a0bee
f00407
fa01fc
040302
fdf9fc
ed03f9
ec0dfa
f10af7
f6ebfe
f3f500
0103ed
f20b0b
02ec03
f6fcef
030200
f8ebee
```

```
09feee
ee0cf2
02ffec
06f1fa
ffeb0b
0106fb
f9120f
f1ed02
ef05f9
f90708
ff030c
090b07
0000fc
fdf801
f8f1fe
fc010a
fe0def
fbf9f1
f30dff
ec070f
f90aea
0707f3
f10e00
03e9fe
f1f7fb
f1eff6
02efec
03f810
fbf5f2
02090c
0d02ed
f302f8
f2ec04
ed08fd
ff0eff
fe0a06
f9f5ef
fcfc00
fefbee
efedee
00050e
060df9
08050b
edf2f4
f508f4
edfdfa
f1f602
05fcf5
f2040f
f006fc
fa05f8
eef5f5
f3fafe
fbf005
07f6fe
0ff606
f6eb00
ecf7ff
```

```
fb0800
0109f8
0cef0f
fdfcf9
01edfe
f9080a
0bf203
06fcf9
f2fdf7
eeff05
f1f012
09f401
05f20b
08f805
03eb03
000002
02eb07
f501e6
fe01f6
0deffc
1515ef
e703e8
fb0b04
f30e0c
060006
ef0109
f30c06
e4ed00
f9f503
fbfc0e
fa08e0
daf6fb
02feeb
fb060a
000909
0c04f5
00e1fb
fae801
fc170b
0df7e8
f1f801
e1db0a
fc02e6
00e7f2
d2f0f7
fa1703
0aeef8
fadefe
da06e2
ebf6fb
d904e0
f0f50b
fadbf6
d9f6df
d1f80c
00fbea
00d9f1
e40a0a
```

```
f810fe
1000f7
f0e609
fdf2eb
f4dcf1
e602f6
dbf5ee
e3eddb
efe808
dcead9
ebece2
f2f5fc
11f603
fc0dd3
fd1519
fffe07
0104cf
efe2ed
e7f7e2
0b00d9
f0eaf6
ebf519
f304f1
1108eb
ece510
eee3f7
00ebf5
fdfc00
feffdf
fb15fd
f900f0
f8fbef
e5fa0d
e4fdfb
eedcf0
f3d4ea
e9fafa
ee19e8
f0eae6
fe06fa
120704
001314
fbedf2
f7ecf2
f5e706
ede90b
f602cc
fe000b
e2f0fd
02f700
ece500
f0e6fb
d7edfe
d8f1f8
01f8f2
f9d1fe
ee01f9
03f4ca
```

```
f5fb0b
ddf4f5
e800e0
faeff6
e4050c
f2ee03
ec13e5
f8f6fb
08f10f
f8f7ef
030df3
0002f7
ffe6f7
ee01f1
f2d5dc
d6fae5
030dea
01e912
f702dd
e2f6f5
d4000d
f8efe2
f7e5dc
ece60f
0614e6
040a12
fe01f6
f5ef02
02f2d9
e1fced
0613fb
00ea03
f2e3f4
000017
f5fc0b
040c0b
f0fc07
f201ec
f3eef8
07fbf9
08ecfe
0311f8
0bf104
f910ee
f6f0ff
f2ed01
f7060a
01fef1
f3f5f9
08f60b
fbf8fa
eafcf5
e90ef9
f311fa
100bff
ff0dfd
0d0cf4
f4f9fb
```

```
f40204
eb06f0
050908
0d0907
f3ebfb
1204ed
0b04fb
0f0d04
00f208
f00dff
f6f3f8
0fed03
f5020c
08f00b
f2f7ee
10f3f4
ee01fe
0000ef
e9f4ed
e9fdf3
f9f9fb
0d0b07
fcf301
eeee05
eef3f3
eff0e7
f607f8
ff0706
05ecfb
f50aff
fa0af6
030ffc
fd0bf3
fd0f03
09fc10
f7f10a
f9f303
0802f8
f9f7eb
0df2ed
0101ff
0802fa
f50afc
0807f7
030afb
0507f7
fbfa06
090ff0
f8fdfa
0dee04
040af2
f6f8f5
01f1f1
fcfe0b
0ef5ee
0200f1
fd0407
030703
```

```
f700fd
03f706
04f0fd
00f702
ee06fe
060ff9
f6fefb
0001ff
ffefee
0df808
eefd05
fa0f04
04f4ec
00fb0a
fefcf3
ec0e10
0e00f8
eff5ef
00f9fb
f0faf6
0501f1
fa0d00
0fec03
e8f602
0206fa
f6ff10
f9f10e
0d0212
fc01ed
ec0cfc
08f30b
090e08
fa0d11
08f800
ff0108
fc0200
f2f5ef
00fefc
fcf70d
0b0af5
f3f800
f5f00a
08fcf5
fd0105
02ffff
08f10e
0c090a
0b0905
020ffb
edf800
02f5ec
00000c
07ec02
f506f2
0cf9fe
02f526
f81210
051c2a
```

```
fefae7
ed0012
fd09e2
06f9f1
def8ea
0e21f8
0404ec
13161e
fcfb1b
0d14ff
09f6e5
09f6f6
e4f80a
1d1b04
f702f7
2a0015
f7f608
001100
0d020b
4021f6
1923fd
fb122a
0a1efc
162316
11e5e7
1333ff
f4bbfb
f9030e
000414
08fd0e
140302
16e30a
0afc1d
1a0f09
1af4d8
f404fe
191b0a
f0f6f6
fbf904
e30603
11c9d0
e7d9ee
f606fe
080618
0705f1
10ea0a
151338
ebe32c
1516ee
0531eb
38d910
f5172d
fefcd1
fff7f5
ed1cfc
f217eb
fa03eb
03e407
```

```
da0626
131700
d5f100
08f8e9
deee05
351ee9
0cd6c3
06f418
f51818
fb1cc8
d00a1e
f20109
fd0ef2
120200
02fb0c
f30114
f4220c
ee0e0a
020903
060b06
ea08fe
eae7ce
090109
e50d0b
041fdf
050afb
1901f7
f3fafd
07000f
1602f7
fbf6fa
0c0a0e
1b08fa
040c0a
01fc04
fef008
01ffee
f0fc01
05f300
e5e5fe
15f2ef
131118
0feaf2
0e1709
fcf3e6
fff20f
120b05
24f414
012110
eb27fa
fd040a
e8cd16
f9120a
fdf5fc
19faf3
0a0605
00fffe
ef0e0b
```

```
17fbff
1b0717
e3f603
03fa05
d6f60a
f7f6ef
faebee
f4e4fb
fd04fc
fa2809
f5fd00
0000fc
dfebf8
0af4f4
1b0616
012503
fef506
011012
ef03fc
08ecec
1f0dfc
c1ecdd
e10606
01e0e6
f2180a
050d00
fefb1e
01fbf5
fc0700
f9e9fe
d0dd21
14f0ce
e2fcff
20fceb
010312
1a140e
0c1b1f
482d18
0e2737
d61e1b
27f1d0
e5c9f5
f808ef
e61dde
ecd710
01f113
0003f8
fb18f0
fcf512
f30b15
f6f6f4
f21b0f
0cf0f6
01f902
e8d1d7
d1dc1c
0700e8
eb14f3
```

```
10040b
fdfff4
0f0c22
142523
f0efdc
14f2fc
21140e
11042d
101d0a
f126f0
01eaf8
f9010b
f2e7ef
130e01
001e24
0006fb
ff00ee
150015
e10e15
eafcf6
f1daf7
f7dee4
f90bf9
25fff9
12f2dc
0b0a1e
07091f
ef0f04
08e200
f1df00
0de3f6
0518fa
110710
050900
0d0c02
cb0c06
fefedc
d60205
f2b6ab
eccbda
0206f3
fa080d
00190a
fb160c
1d17fd
fa110f
e800f5
1814da
fce9e7
f900f8
0e0dfc
0efd12
e90115
06000c
cacbeb
f1f5df
050808
1b0c0b
```

```
103012
1a1322
f81018
e4feec
211cea
f6f61a
0e0b1a
1f04f8
f1fff8
fefb08
ed01e3
e5e1e1
081906
fa00f1
05ece8
d3f2f9
040700
fa1a00
1e16fe
100009
041806
d2db0d
e013e0
fdf9ea
1c0600
eefa03
250b07
1a11f4
03f81a
000008
01dbe7
e2f905
101518
de04fe
ef19ee
eaf81a
07fe0e
fd0afb
02fa0a
02feff
1506fe
000108
022a20
e0f904
ef2009
ecfd10
f2ee10
e8e8ff
020f08
12ce01
271a1e
f302e0
f6350c
e4f2de
05f3ea
ec17f9
020840
f7cb19
```

```
49fa0a
2609d9
f60907
0d1e2a
0c13fc
f3f01d
020914
26f7db
e4f91b
eb000a
1c02f6
f8faf4
140af7
fae5ed
0cedea
e21313
17e6d0
140df7
ec2536
f61025
0706f5
05fbf1
eef7ff
f41801
c3e5f5
eaffe4
09f3df
f70020
2308e5
e80a1c
fbf4fa
f9fe10
1609d8
e5f01c
f1c911
0922e1
fb17e5
fb0903
e4e10b
ebddd1
fce301
e3020e
d5fa13
15f0d4
e70e2b
22f3e4
e8e91d
15f1f4
e2fe01
051707
edf7fc
e305fb
02eafb
2ef702
0c161b
f90ff8
02f600
f0f5fc
```

```
e801ea
1c2a1f
ecdc06
f70d07
f5fbfd
fbfdf1
0308f3
12fc06
0f0c22
15eee9
f1fb02
0213ea
fb0500
e70004
f80915
f50400
fd00eb
08040d
dd0ef7
cee510
340df2
00f409
09f0f6
0d130c
0f0cf1
f50220
15e9c7
040c0c
0a1bfc
03200c
01f514
f601f4
e7d905
17021c
1302e2
0fdff3
130604
ea1606
e5f60b
e7d2fc
ff080f
0fb2e0
40031b
e10619
ecf1c3
0cf008
0206f3
0000e3
f31313
0cf903
f7f4ee
ef03f9
07f402
fe22fd
f303de
e9f8fb
fdef0f
09ebf9
```

```
1ef908
f90f08
0efa00
0bf305
0ce606
0410e5
f7fd04
d4ecf6
0ae1e6
120df7
0efaf6
051223
0406fb
141703
03f419
efff05
31f6ed
101ff8
e406ff
101b27
40f1de
f91416
2b5928
05eff6
05040f
020b03
0eedef
11ed03
e0fbf4
d4ee0d
00f3fc
07d4ef
e1dee3
06f900
0f0407
0108e9
23000b
0a01f0
02fcfe
eff0ee
241019
001521
f3fdf2
eaf300
f4f315
1ae4ef
10d8f0
fce7dd
111409
0a0cfa
000d09
fc01f5
0014e9
f4f1e6
ddcefe
d50b00
ee07c7
1e230c
```

```
f10500
00fafc
1411fc
0204fa
deff08
0c1fd0
05ced6
fe1700
02fb08
0afa09
0506ff
0d00ef
e3fc17
dcfc05
04fbee
03e9e0
ec03ea
0008f0
051900
fbf900
0b0817
0cfd02
d4e1dd
04eef8
0a0a11
fe0800
fe0709
f6f6fa
f50af1
04fb00
faf507
ed0bf7
1ff100
f91608
f51e03
f0f419
fd0011
070d10
e700fe
d1d6db
08ffde
f7fffc
160aee
f0130c
f7030d
28f707
0b00fc
02e500
050702
fdfb19
0604f4
0600f8
f3000c
f4ee05
0110f0
0a01f1
e4f012
00fb13
```

```
edf1fa
0d2f1b
ebde0a
0b15f9
eb080c
d41a18
0de4f8
0000f9
efeef1
02fc0e
fcf4e8
02040c
fffe1a
0d03ec
0418f1
ead4fb
e7f40d
0507f5
f7ed02
001bf0
f7f6e7
0c2406
fef915
f80b09
1f14f6
fbfe31
fdea0d
f822ff
ddc9c8
ded0e5
fe1804
261c08
261c10
0604fb
feeee0
110006
020bf6
2f0412
15ff01
e70609
04ffc3
eaf2fe
e6edfa
0c1507
1c11f3
180816
0d0528
230fef
f5e8ef
1509f0
18f517
15000a
daef01
cbd3c9
12dedb
00170c
241e21
011720
```

```
ee08f3
2204f1
141831
09f738
0b221a
14d0c1
d80624
dceced
1bfff4
0a0017
0a1e17
0dfd00
231503
ffe01b
22e10c
ded6c8
2006ce
270a1d
fd061f
320e01
01f0d0
95131f
e8dbc8
150809
1dfe27
e1e0e4
15fef3
ed0c0d
1a0bf6
24fcfe
0d1c08
0313f5
f2dc0a
f2e8f9
edf6f3
feeacf
e7f7e4
eff5f3
0a220b
040317
0022fe
f7240c
eef902
08f6e9
040606
000b15
120c00
090d11
131efb
ffeff1
e8ddd7
f4e2e8
fef5f3
1f1f0e
21032e
0401fd
edf50f
f8f3fe
```

```
f7ef14
fb00ee
eafef7
090804
020d14
1813fa
f40814
f20706
f606f7
f6eff8
ec0b02
06faee
040a08
1c0904
fe0214
0aff1c
18fde8
fdf3ee
daf71f
0f0211
efe3e2
fc06f7
1708f8
fd0a08
fbed13
000010
fefef2
090907
0908ee
f10e05
f004ef
edf801
ff0809
f0080c
edfcf4
f90df5
02e7ff
0c01f7
0c02f3
f9fdfb
04f603
00eff2
0607eb
03e90d
f0f1fd
02ecf0
0602f3
06ee08
0b03ed
0705fd
0505fd
fdee02
f0ff08
fdf7fe
f5fbf0
09ec00
eefa10
ed0afe
```

```
10f609
0bf2ea
edf5fb
07fced
0a0500
fef0fb
efff00
0a0b05
ee0ff2
fdf80c
0bff04
f30402
f0f5fc
000a09
eaf1ec
06edfd
1000f8
f00de9
00e905
08ed0b
f8f9f9
f2f0ff
f90f00
f2f302
03fc0d
f6040e
04eaf3
06ff06
00effb
f30af4
f0f5fd
fffdf6
09ec03
f2f8f0
fdeff6
fbfa01
ff0b08
f5f9f8
ecf310
ed07f3
0cec06
fc06ee
f81402
00ebea
f8f602
fdfcfa
f2fced
07ecf4
f1f9f2
130cf2
0cf7f1
f5fc0d
0206f8
fe08eb
feecf1
0502fc
0c01ef
04ea0e
```

```
05faf2
f506ed
05eb05
0aed09
0bfdee
f1eef6
f603fc
f1090b
f8f2f7
fb030e
f0fe0b
f80a09
fdefee
04000a
f202f7
fbfcfd
f1f8f4
f50f06
edf9fd
f7f6f1
04f2ea
fe0bec
fb0200
f10a08
fe0400
0cf8eb
f2ee06
f8f10e
0cf8fc
00f1fb
f5ecf5
f6eb0a
0dec0e
010b08
08eded
05f901
ea0eff
ea0703
f30afa
f203fc
0cf700
ed020e
fcf8f3
0000f0
f305fa
f40211
020207
fcef00
070ded
070000
f0f9fb
ed02f0
f3ff0f
f2ebfc
f20d05
070dee
07fdf3
080eed
```

```
fafc09
00f2fc
f5fa05
ecf500
09fcf1
0af0f2
fdec07
0feb0c
060505
05fff6
010fec
fafa05
f3fef6
ee1005
f8fff6
0eeaf6
0ef407
ef020b
f00b09
0bf4f9
070d0a
f9f206
01f803
f60dfc
02f5f2
020bfd
0c0fef
05ed08
08040d
eb0004
01f1f1
0df2f6
0df5f9
0ffd01
f7fa01
f802ef
ebf706
f8ef00
070805
0c09ff
f10c00
ec000b
f3f802
0dffeb
f5f1f5
ed030c
ebfe0e
f1fdfa
f7f2f5
0f00f7
eff6fc
ff0409
0407ef
ff080e
f900fe
0605f3
000ffe
0df90b
```

```
eefe0a
fef9fe
0404f6
f206f2
fb0c0c
0e02f3
f30a00
f70ef5
fffa0a
f70108
0b0efb
f60cf4
f0eefe
00100d
ff0507
0eee00
fa0400
ffeef8
ecfdf8
0df808
f00cf6
00fa0b
f109fa
f8f40d
01f00d
030efe
02ee00
06eff7
02faec
f8edf1
05f1fb
ef08eb
ecffef
0bf4f9
010409
04eef0
fbf000
f6fc00
f6f903
08ecf2
0cfef3
02f1f2
fb0105
fcfc02
eef9eb
f3030b
0bf6f0
f3f7ee
ebf5fd
faf506
0602fd
f9effa
f40800
f3edee
03fd06
090605
f60008
01f50a
```

```
0cec05
f5fbec
01f0ef
0000f2
07f107
02fe08
e8fd07
fae00b
0611f8
070e16
28f6fc
f0fb08
04ee06
1c0807
ffedf0
ed110c
0009de
f819f9
fe1323
fafe22
24161b
060b23
ea0ebc
e50e03
fa09cf
02e304
fc1c0e
00040d
140b06
1419df
f1fe1e
1cecf9
e6031b
0c08fa
1fefe3
d0f7e8
c602ee
e9e1f0
0efb04
0613f4
fbf9f3
0fe310
eefd0e
08130a
e5e902
0c0500
d4f007
f8f0dc
d9000f
f90ef7
13d9ee
00eb13
ff0011
282008
0deefd
3bf2e3
fb0124
f02203
```

```
e3f600
180ef6
08d610
04f513
ff0100
e6e604
ea162c
2afdd0
1bf9f9
071837
f5f404
08e8ff
2400f3
062734
2f2aea
d8e0fe
2902e7
0af622
f00e08
0c19fe
3af800
06eaee
fcf303
ecf7f6
1300ea
f90a20
f8fa17
1d0e14
f6effe
ec02ef
efee03
0ffae0
05ec0d
d90503
1010fd
162618
fa1a04
08081b
0112ff
120204
06f806
060a05
0ff709
03ec0e
0414f9
ed04f2
01f1e2
17f5f3
e2000d
0621ec
040509
021800
191f1b
f001fc
f9ff08
0bf803
edf306
f500f1
```

```
fd07f7
1713f3
effd00
dcf923
ec0000
07f6f7
14f824
f50500
ee08fc
02f90f
120dfc
040202
0e0de9
e5f42b
090b00
eefbfe
1b00ea
07f1f4
3134f4
0532fe
0e1506
000009
03ef02
ededf0
f3f70a
000605
09fef3
f0ef03
0012f5
f30a09
04fdfe
050b0d
0505fc
020a0c
0cfbfb
f2fbff
0bef05
f307ef
f1f9ef
0d10f6
f2eefc
f8f6fc
0e04fd
f7f108
faf90a
fdf902
0c0010
f2f30d
f0f7f1
03eb00
0eff0a
01ec05
faf4fc
0704ee
fdf4f2
f50705
f0f2fa
ff0df2
```

```
06f5fa
ff0ef7
0ef009
06f7f5
fdf0f9
f90dfd
fa0203
f8f8fa
f60a0e
f9f6f3
0d08fd
0506fb
0210fd
f0f5fe
0af400
fdf801
04fbf6
02fef7
09f711
fa01f0
f7fc07
fdec0d
0a11ff
edeefd
fe0605
10faff
f8faf2
040bfd
07f700
0ded0b
0a0d0e
ff07f2
fa09fb
eef2f4
f8fbfc
0c04f9
10ffeb
0cf801
09f511
06ec04
edf603
0611f4
f80cf0
0b0df8
ef06ed
07030a
080506
f10dfe
040204
ecf7e7
f400ea
f30703
f8ef11
06f80c
0ff304
f3f0f3
ed060f
0a00f7
```

```
0bfbfc
06ffed
fbf40d
fe0d04
0df1fe
0c0bf9
f4ee0b
100dfb
f604fd
ec00fb
02f0ee
01f1f8
ecfc00
f1fdfc
f007ee
fdfff2
f1ee02
040100
ec07f4
f5f7fe
eef005
f800f2
f408f8
edfb03
f50009
f6f9f6
0cf201
090bf4
04f502
fdec05
02ec0a
f3efec
0df303
fdeefc
06fb09
fefef2
fcfbe8
e500fc
f90700
000002
03060a
09eff4
fbf818
08f104
1df1e5
e713f9
ef0dea
080e0b
1e11ef
f20b12
0b120f
f50120
f8edf6
f8f6fc
ebe7d7
f107f2
170107
2a1215
```

```
0e111d
f9211e
180cfa
00fb29
04f2e6
fdfc36
2d311c
23ff16
0befdf
1f3fef
dbdd07
092f6b
560900
12dcf3
e1310b
070ae0
0a00fd
dbfc19
0ae8df
f5fde8
e90c09
03f5f9
f802ff
e9e1eb
0e23fe
130cf4
ff0915
e80f2d
09f4cd
12e5fa
000119
0df3fc
0ffde9
1608fa
1e2d0f
282119
010824
0f0d1a
ea2b32
231606
fb0d24
f1f5fa
e8f91c
26efd0
df21f1
1f1d28
2a0018
e5f7f5
e11ad7
120bfe
1f330c
f602ee
0cff17
1010fd
0805f6
d8fb15
623713
edfa07
```

```
0ff209
ebdf14
101109
f406e2
030d09
031b1d
1e00ff
fff909
0102f1
f90d0d
1508fa
edd311
0406e8
1cf4fe
f3fd09
0c2421
f2030e
15ffeb
05eef8
ffebff
e40ce3
03e2ed
0bf7fd
fbf3f9
12eeeb
000df9
fe0b07
fb2410
18fcf4
0503ee
ee2318
0f0013
232938
f5fd16
0d10f4
d808e0
e8fa01
00d9ef
f708ff
091608
00e6fc
050119
14f8f3
fe01fc
060227
fd01f8
f200fd
f4fcf5
f4fffc
1c06e6
f2fdf7
002d11
fcfd01
fef5f7
17f0f6
fd0035
1b0f0d
000019
```

```
ecf9f6
171be3
040a1b
002919
f5110a
1eed12
1e15f7
ebe913
f8e500
0df601
fc02e1
1dfcf6
ff280b
1e1514
200c2a
031c14
f30bfb
eed6fc
f50d21
1b07e9
e4e8e3
f625f9
0405ea
061aeb
f613f4
fd0f1f
1ff4f6
f6c71e
f210df
00d3df
f2cd19
171400
02abcb
09052a
fe06f7
ef1ffc
131400
09041c
032015
011501
01fa03
eb1516
02f7fa
f7f710
240ff3
06f6f5
f505fb
06f903
0816f7
070300
ff00f5
1000fc
13f80b
fd0a0a
1a0103
de04f7
f8050a
ef1600
```

```
0c05d4
1911fd
fbfb10
13fc1a
15f607
fe1c02
f5f121
2a09df
f8ec0e
f600f9
0c0904
1d17fc
2206f8
fc2822
1a130a
11ed0b
d5fa25
faf2fc
f30707
09020a
17090f
110d12
ef0bfd
1ff9e3
07fb20
e4f119
17fc00
ffedda
f00cef
0f04f8
1005f7
00e80c
ff2a26
ecf301
01fcff
0ef908
0007fe
021dfd
0c1311
031a0f
121814
f80ef7
ebdded
f60305
09080e
160000
0a0421
01fc22
1f0004
f1050b
ffe6f3
f8ff05
170f1f
1c0108
051f12
0e0b09
f5e704
f5fb0d
```

```
fc0601
00060b
11fc06
04e1f2
09170f
fd0305
1e0501
1afa16
1bfdf0
edfbd7
f20a01
01efe6
11f50b
04f717
f8f0fd
31f7d9
ec021d
0000fd
090af4
f5e3e8
f30800
f01ef3
0e2015
16f3ff
f01815
f705f6
fe0615
d4f6f2
ffe7f1
10db07
2a1c02
14f612
0d1200
fef6ff
fbfaec
fde203
eb0520
2bf1db
062224
080f18
01fa07
ff03bc
d5dcfc
d60100
112ed0
cbc213
210cf0
ddf4cf
f8d302
11080c
fad2f5
00ec1c
f00726
2006e4
f3cefa
dde31d
0800c5
05fef6
```

```
f90ef4
ec1f16
220207
e7fc34
3a2af6
250a0e
ffef00
d3f81e
ff15da
fc170f
070cec
f7f0ee
f6e809
0119fb
3119c8
fc143e
2611f9
0c1710
f3f4cd
fb1010
f60cfa
0505f3
04df0e
000500
f022fb
2904f9
e4fd0d
ff04f9
efe0de
26061b
020325
244514
09201f
0e04fe
c3e608
faedf9
080012
0208e8
d8010a
e5f0fe
fef819
0104ef
fd1818
0406fc
18f50d
00f21a
020011
0f19ff
0015ee
e8f6fc
17f90c
0dfaee
f400ed
fef8f8
d8ef03
1306ff
ee0e11
16f80c
```

```
f3fd1c
0ff9f6
fa1003
00f8f2
0c13f0
e901f7
031bf7
e8031c
270e06
f0080e
f5e8ea
fff5ff
0aee0a
13070b
0b0af4
cbf517
f20909
fc08eb
f303f4
04fcea
08f9e2
f1e31f
0c0cff
eb0de4
0b0bf1
02031d
0a1812
ddf3f6
0613ec
1cf6e4
e72112
c002f8
fadf0e
3601f0
f7f6fb
000017
05fcf2
0002f6
f8090f
f6f403
02f4f7
eefd10
020a09
0df5fb
eefff8
f801fb
0df001
f4f4fe
f6f4fc
fff300
fff5fc
f1f5ec
0df50e
140f0a
fa0ffc
ec0cf2
080bee
020e08
```

```
08f10a
f8f400
09fa0e
fdfa03
01fe03
0ef1f2
05030e
f208fa
0fed06
06f502
00f2ed
f00a0b
0906f7
09fc0c
f1f8f5
faed03
f20ff8
0c0cf9
04edf9
0b0ef7
fb01f4
0100f5
fa0f07
ed0cee
02f105
ef0cf0
0410ee
041004
ee0008
f2f6f0
00eff6
fd0ff0
eef8f8
070cfb
f0080e
050f04
09faff
08f903
05fcf5
0cfa0a
01f207
fafe0a
0e02fc
00fa07
020605
0d0500
06fff0
fdf007
08f9f3
f3f2f8
fb0409
ef09f0
00f3ed
02f4f0
07fe04
fd0209
fbf703
00f107
```

```
f00709
ee0bf1
09f20a
01f7f2
02f310
efed0b
02fcf8
03f80e
ef0906
faf507
0a0ef8
fdecfc
0dfa0a
07f8f0
fc03fd
0ff8f4
0eecfe
010500
010108
040008
fefb01
02f7fb
07eefa
ed0d03
03f6f7
ff08ef
fffff1
f0ec06
f1f300
ed01f5
ec0502
eef408
f00f02
ef00f3
fdf401
0df4f9
f300fb
eeeefc
0cfb09
ec10ef
0ef40d
ef0e0a
f2f5fc
000af4
ed050d
f91403
fd07f4
fc0101
f60000
f908f9
07f2f1
0e03ee
0df200
00000b
f6fdfd
0e0d06
160b17
21fa0b
```

```
100103
f104f8
eb150d
1808f4
ff0711
070704
effb09
f607f3
fcf901
fd0008
07130d
0906f8
08fe18
e4d9ea
fafcd4
c1d8e8
ebfaee
00d3dc
2122ff
f10718
27131b
2b1413
020b4d
2b03f6
11f6fd
16ece6
001e14
03e7fc
0b100b
000f0b
0504f5
1a1209
012d20
ff040e
f6040a
f4e518
0a0411
33ffff
c1d8e3
e2f6fc
ddd1da
e7f5e5
0e12fe
2c1914
f6f915
000800
fff014
fd0104
cddfdd
fefaf3
e8ef13
07f3de
cf0bf7
e3deee
000222
f9f2fe
15fa1e
0d0624
```

```
19241b
0cfa0c
1ef8df
cfee07
0b13fb
f5ea05
e911e9
dad6ee
1904f7
edd8de
f0ece8
07faff
1c10ee
ef0406
1eede6
ee010a
1702f2
200714
f60900
fe0b12
080cee
e407f3
f3e6df
dddfde
06dff0
1f1d18
151509
0d1727
fafff9
090c14
f9080e
0e100f
1a0010
170dfb
071116
041105
f4f60a
0afcf7
f0f8f5
01ffe5
d602ff
2c07eb
1c0d2d
1a05fa
d31a07
04fcf9
0aef1e
19000e
0b0411
0118fc
f9000d
fc0213
1a160f
1412f3
080012
261600
000f0c
1406f4
```

```
071300
08f102
f6001b
0b1207
f9f011
f7eafb
040af5
e7e9db
2c020c
12fdfc
fd1b0b
e40107
13fffe
0000f6
031106
2a03f0
fd210c
031eed
0c040c
00eee4
1aff14
133017
17f900
100f04
121213
04f1fb
10fc15
e1f402
1c1816
071a21
061817
fff1e0
f317df
f6ecfc
04fef0
e010e1
2416f8
fb0431
0b1110
28142a
011064
430d0a
28e812
040506
114018
06d11d
0e1b0f
12fafb
0b1010
fc050b
0f0a0c
f9041f
150c09
f7fbec
f2fb0d
f007fc
dfe8ed
dbe6f9
```

```
f6ece7
ed0a12
fa380e
062217
f0fc02
270ee2
edf3f2
f40d02
f0fbf4
283de4
fee912
f7d8f0
c11207
1febb5
e2fd2b
100113
1f1804
330309
f2f513
fb2318
36feec
fefef1
091600
e2f5fb
103404
ced1fc
f80f0a
dd0cfb
04dfe1
ecdb40
46140c
fc14fc
090af0
0d070f
031402
fcfaf6
fbf9fe
09f912
010808
f5e907
ebdee0
fce8ee
00f1f1
131717
06f111
122410
0f151b
000b15
fd1212
101701
f20513
161007
0c060f
fc1009
1825f0
1704fc
e0e7f6
f4f7f0
```

```
000a18
181b02
ea1106
02f3fe
f724fe
0e08f5
391f39
fc121e
0d1408
ee0af2
0400fe
2afdff
101921
2f430a
0d00f9
2018fd
02ed09
0205f2
16141c
2adfee
fff616
ef0f04
040e06
cbead7
f82ffa
02defd
28040d
091b28
0207e0
fde702
22ffe1
000027
f5fcf0
05ffec
f21519
020201
01f900
21fafe
fcf401
021c29
16f0fb
ebeaf9
161700
03f5f0
140915
020f0f
0afd1d
160e11
ddf10b
03dbc2
16ef48
221afb
eefcf2
f226f7
16f2f3
ef1208
1b09ef
231e11
```

```
001920
051a1a
2d16f6
dae118
e70b16
18f7e6
eeaaef
030920
201800
fef916
2212f1
0efa09
221506
0e00fb
f50af9
ce0aff
0a00c1
eeee1c
221614
f6e5ec
ee10ef
0c00e4
000c0f
1d1714
f3ec08
000107
101df7
042112
12ffff
f2fae1
cd0606
f1fee5
1100e1
130e0e
f0ed0a
1c0d02
0ff212
0005f8
f81306
1400da
ebf8f8
fde5dc
0efd09
1b1b01
18feed
da120f
03e7d0
f1ef12
1dff0d
f3ef02
0c00e8
f80b16
07090b
03fdf8
fafaf2
0306f6
080c15
eef800
```

```
21f0f3
09e7d8
f8000b
f2060a
edf80a
0304f1
2422ff
060f1b
e20a0d
1819fd
ee01f1
150a0b
151705
130914
0c14f9
f81802
e9ed08
e9f4ff
170cfe
021f02
180311
fae400
f50716
001810
131815
f40f24
030f02
f2f401
050af8
16f2f2
050302
1a2bfd
f50f04
f1f217
0f1206
03edeb
050c00
1c1706
fc0a1f
071801
fd07ff
e7f4d1
e318fb
f60bf5
170809
fcff06
14eadd
151604
06000b
00000d
f90016
05fd09
0af708
0ef6ff
f50800
11f2d6
d90cf3
042d14
```

```
190710
02110d
171e06
08ee10
f1f30b
12f6e5
eef618
08eae2
e710fa
1404ff
ed070c
15f0e2
d7f42d
e814e2
f1d5ca
eefb1c
16f1f5
08d90d
ff0af0
1b18e7
28000b
04bc1a
e40c2c
29f2b8
d9a1fa
fa2b07
25efe2
e00011
031b12
0d1ff1
11ed00
080ff3
ea1f07
0df311
fb0afe
06f716
fb0f1c
d2fc14
dd04e3
f1dfc3
03de06
fef9ff
02fd12
dd100c
2cf5f9
0efe12
fafb06
f70516
bf1c01
19f7a8
f61cff
071606
f9ccf7
f605fd
112929
fe18fc
1911ff
13e9f5
```

```
162e20
0e1c09
e7e105
f9ff03
160b04
061020
05e9d0
b1ae17
2132ff
ff181b
00ecfd
080813
ea090c
fd16f6
0006ef
010300
18ff0d
06040d
10000a
f91328
0ceaff
e7e3da
f7eafe
0af7f7
08fc0d
1d0110
060d14
140a09
fcfbfc
f81a0a
e9f102
020c13
110403
051d15
001103
22eafd
050bf8
eaea09
04f3f1
08efe7
eafc01
2812eb
230e1a
e20404
1d2bf7
d00600
fe0f00
d9d312
fc101e
1704ef
0bfc15
11e717
f60000
0910ed
101100
09e80d
eff505
0e09fb
```

```
180b07
2a22db
1616fb
eb2c0f
de12f4
0be2db
ed00e2
f3f9e7
fef4e4
000005
f6edeb
071914
faefe3
0cf626
fde6fa
13110d
33f8fe
fcfc28
e2eff6
2d141a
0dd5dc
ea270f
0b000f
1600f8
f00000
fe1329
1efaff
f50900
fef5d8
e6fcf2
010de7
d3f500
fa04f8
13fb17
211d12
fe100f
f9cc0b
28dffa
06012f
540e00
e7d912
0a01e4
05b7c9
fcf8e4
e0fb0e
03fd10
0a2007
1d00f3
101819
1201f8
fbe8fd
1510fc
0a1912
0b08f3
b5e20e
0008f1
ddec02
0c250b
```

```
050914
230910
02151a
17faf5
e5e420
f1e510
e6fe0b
0201ed
dcf812
e20413
0f311a
eaf2f8
0d1006
07120d
01140f
16f100
19061f
1a08ff
2c17ed
0f0d10
fcf602
e0e606
f9f2d8
d8e2fc
e500f5
17dcf4
34161b
faf1eb
0d1205
fe0d11
0af3fc
190714
080ffd
200eee
14030d
f5d700
07effa
0df3ea
cc01f6
eb0b00
09f204
15feef
072414
ff1c00
180101
f0000a
1c1512
fff811
1b07ef
01fc12
ff1206
fcf109
f9fff3
03ff09
e7ee0c
0de614
00f204
ff0eda
```

```
03eff9
131007
e20015
1a17fc
e1eadc
fdfcf5
eaf505
271ffb
01011a
002308
02f0f2
190602
091022
270e17
f6f519
101716
13f810
010601
0504f8
07f5d9
ecdd1b
fb0002
26e306
150100
ea11f5
1108fc
f00409
000006
f512fc
050cf3
05080c
11fde9
ee0513
08fafc
060212
04030c
f8f8fb
e0f8fc
ff060c
2aeefc
23241c
13f211
05210d
0312f8
da09f5
fac6d4
1dee51
3b1ce7
fd06fa
001709
ed0005
0005e5
edf8e9
d70222
17f7b4
c6d019
2305e1
ec000c
```

```
ece217
1b20fc
ffb8f1
1e0f0c
0cf713
fffc16
0fe6e1
ed08fa
0412f0
00f6f8
09f503
dd1c04
1a2003
fbe82b
2327fe
1602ea
f20812
f503f8
0b03dd
fe0002
fd17f9
d00511
0af4d0
e8f413
161be5
e4fcef
f1fb05
f305e7
fff9d9
0e03fd
fcf3fb
e212fe
f30001
0800fe
0210fd
28dde7
cf1b13
1bddcd
f6df26
281912
1518f1
e52803
fa04e1
f9e206
d2fc07
17f9f4
26fe08
11f60c
0f03fa
e5fbf1
1b0cf0
efee03
101c13
00e00d
0f0b0e
0ffcf1
f6fd17
1009fa
```

```
eff100
e1f0f6
fa0f09
15030b
e8eb06
001df9
ea01fc
100e0f
050a01
05fa08
0c0312
0a0b00
f6f70c
fdfd0b
250efc
0a021a
06f518
fcfb07
f902f7
031503
12efee
fff5ff
fd320b
02e811
01f30f
f5e9ff
0df812
0206f4
091902
f8f60e
030dea
f1fc02
0b0cfa
f519ff
f5f900
fcf905
021b12
1a03c1
d8fa24
1a0107
1003f7
fe0304
010317
0bfddb
e3fef3
0000eb
2209fd
2304f2
10fe04
1bff02
1f0804
0c0300
1007e7
162424
291c04
180228
fb0105
0b081e
```

```
f5f41b
fc0404
0ef205
150f03
f4f3fa
d6bdc1
d0e5b5
ed0ffc
14edea
1d02f5
1c18fb
00ef24
f9f001
50430d
151f53
393611
04282a
1f251c
481715
152e46
0f5516
1008ff
241509
f90817
20ff0d
07040d
0f1a0a
f30303
070e07
f3fdfa
b3b8e0
d1bcd4
05d1ca
0b0dfe
1e05d1
181610
fdfdf9
09fee8
201307
1d001e
cc0723
f40ee3
d8f808
0eebea
f0d9f5
0a1438
fefd22
ef12fc
ed122a
08f2d7
0002e1
f6fc00
e7e6eb
000e10
0903fe
ebe5f4
dcece8
c1acc6
```

```
f2d8be
01d3ce
cff405
0716df
16fe12
f805e5
111af6
f70df9
0a0f06
09ec0b
f21602
f6f2f9
fd0e04
bff3eb
efddb2
ffd7b3
17e3f1
f31413
f90b0b
091b1e
0df500
00e505
1f2307
092315
1302fa
fa0f0a
fc06fa
fb0501
f6190c
f5f509
09ea04
06ffee
f5f904
03f2e7
eee6f8
f8f3ee
0ffd02
0700f0
fefe05
081319
0f160a
07060b
f61f10
1b1417
0ef717
181a25
fff51b
1f00f2
0bf826
fb0713
f80e18
080306
f1faf9
0d06f8
f912fb
c3cdf4
c016ea
ddfbe1
```

```
1dfd02
2dfc08
2529f2
f1f212
09fd01
000006
f6ffef
0bf506
fff9fc
0bf301
fb0b07
fe04f3
0bf6f1
05eeec
f2f9fc
f104ec
08ed00
0d050c
00f305
02ffed
0bff00
f6fdf5
070000
f1fb04
fbf1ef
f8f500
faf5fe
f40009
e91004
f2f302
f908ea
0ceaf7
e7f9e6
e514f4
f7f409
ea0509
fb0be3
000a03
ecf3ff
eefef6
fdedf1
fbed00
0305f2
fcebf5
0002fc
fe07fb
f3f7fd
fdfaf1
efe707
00060c
ed03f3
f3ef00
f40004
fa090d
000af7
0bf905
02ee04
06feeb
```

```
eff5f1
f6ef07
08eff8
f4edf4
fc0afb
e80d0a
08f0fc
01ff04
eff1ec
f6edf3
00ef08
06f909
ffec00
f50eef
f506e4
ff0a03
ed02ff
f0f507
f400fd
f6fae5
fcf5f4
f50605
08f703
f9eeee
08f903
f0edfd
0d01f1
0d0c03
e90208
f0fdfc
00ecf3
07f2ed
06ea06
02e307
ff0202
010f0b
faff08
ebf8f2
0006f3
0a0601
06ee06
fcf6f3
01f2fe
0e0bf4
f600f6
06f201
fd07f4
06f4f6
f5ed00
f2fa0c
00100c
fa01f1
f302ff
09f2fc
f9eef9
000603
0df1f7
0aeef1
```

```
f80af2
00faf7
e90400
fcec07
0805e6
07ecf7
f2f805
f30500
ec07ee
f4ef0a
f5fcfd
ff05ef
f4fa0a
0001f8
00f3e9
f3f8f8
05f2ec
eaef12
f102f0
f6f003
02fe00
010df4
020bea
0000e5
ebdb05
d7121c
00fafb
ede60a
ecf6ee
180022
0ff2e6
ff0105
e6fdff
221405
03e2cb
cf0aed
f5fde9
172008
ecf622
1e070c
260cfe
cf0a35
ea12a3
de06fd
fb02e2
f9f2fc
d201f6
fef2f0
fbe9f0
1ff3da
e8e91c
03d1ed
e4f923
0aeecf
c5deed
12fbed
e7b7e2
eafaff
```

```
f603fa
000a03
e82512
07e700
03f51c
191e0b
fe0009
420d01
d10930
eb20c4
e0ed00
090bf5
e9101a
f3ef03
faf50a
f5000c
e7e2fe
27fee8
f3d81e
03f1ea
02fef6
fd00f9
0be90e
110317
edf900
e7e9ff
012909
fbfdfa
270d04
ee150a
0b1712
041200
26080b
f21910
1d30f1
dfe006
05050a
080c1c
241a16
060df9
0508f7
f002ec
e70cfb
0ef4ea
f5e8f2
02e810
e801fc
1d04f4
010a13
3a390f
dfec17
fb00fc
0006f5
e9f6fa
e60907
05e3f7
fb17fe
f800f4
```

```
1406ff
e4ee28
1c00fb
0503ff
19f406
000509
071211
ff0818
1301f8
ea0e03
e9fde4
e9f8fc
f2f2f7
0e170e
03edf0
07150c
d4e30e
0bef06
e7caf4
0613ed
e50ef4
ea07f2
0a06fa
eaf720
0101f3
04fff5
10ef1c
060ff7
e2f8e1
030506
1707fb
f8fc0e
15fef5
fe2e36
02efe3
08ebee
f2e506
03ebe2
f7f2fd
ee0700
f504eb
0000f1
000015
0000e8
0000f1
000007
0000fe
000010
0000e4
000034
00002d
00002d
000007
000012
00007f
000044
0000ff
00000f
```

```
000012
0000df
00000b
0000f6
000009
000050
0000d1
000067
0000e0
000028
0000e3
0000cb
000072
0000de
0000be
000055
000006
00001a
000029
000051
000030
000053
0000dd
0000d5
000014
0000b4
000047
00003f
0000ca
0000dd
000035
0000ac
000039
000072
0000cd
000056
000029
0000b8
000022
0000bc
000002
00001f
000036
0000d9
000013
000050
000072
0000cb
000020
0000ec
000001
0000dd
0000ca
0000df
0000fa
0000e1
000040
000015
```

```
00001a
0000e1
000003
0000a5
0000db
000028
000035
0000d6
000021
00001c
000039
0000f5
0000e1
0000da
000015
00001a
0000cc
000020
000039
000023
00000f
0000f2
000098
000024
0000eb
00002a
00001d
00003b
000012
0000c4
000028
0000dd
0000cf
0000cc
00001e
00000b
000011
00005b
00000c
000057
000009
00003d
00003a
000049
0000e4
0000d7
```

Listing 24: Packed FC2 weights and biases for three-lane FPGA ROM loading.

```
e9f9e3
fc152d
121bd6
f0f500
fa1a07
fd4de7
280806
2d1304
00d918
293b28
e9d9ec
db30eb
dff9f5
0f2a0a
36ec00
c703c9
d7cd1b
fafbf3
dc14e6
f31c0d
c21d03
0521de
07ed00
ede2da
2e1fd5
24f7d5
dfde08
f85aa0
e52fe8
0d09f5
e530d7
d9f93e
e4d11d
e01f26
23ef30
0bdacf
14f01f
3910fa
00eef5
e6e095
1630d4
e7e3d4
1c22ea
051819
f3d3dc
fcd2f0
1e25b9
13de14
0ddb02
e42413
2fd4dd
100bfa
3e1b02
26f6fd
e92b05
071dfe
```

```
f2e412
e027e3
e91304
e50012
1ddbd2
3df7fa
0235e1
0ffbfd
faefe8
fe0e27
33ee0a
eef7e3
0f0efd
141006
edf10d
05b724
f80b01
f7b21d
2cd603
0fd52e
e2efe9
ef1a04
e2ea22
321604
def9ea
e5fa02
020d07
e4dd0c
f6ef14
040d17
271c17
e52fe3
0f1747
eb0cf0
e9f422
f329e7
fcdf17
fb19da
122b4e
24e538
fb351f
c2ec2a
e3f618
e60935
0acdda
da12fa
0fe9ee
e1fc15
fdd0f2
213c17
efe3ed
0f71cb
ecda02
1610f8
fe2af1
0723e8
ead805
dd48e9
```

```
010a00
d0fea0
e24b17
dfc90a
020ad6
d517d8
eb1c1f
ea24f1
18c134
f8dbfb
021d12
f1cdfb
f9f804
fa0d01
ee28f1
e1eb13
f21619
08222a
fa2e25
e817e3
0cf3da
1a22ed
fad5d1
fd2122
1537ff
fac8e2
253202
f0ea0f
ff40de
ee12f5
183819
eec6fd
10ef1c
d10ee4
e7e1fe
cc19f4
10f0e1
f5f3de
2be705
f6e7fe
19ec44
42e93c
05b326
121c43
d01b24
0af328
27dc08
0fe534
fd1407
19db00
2a1431
0f21fb
f6efd2
05f4d8
26d5e7
d9fffc
e210da
f11ddf
```

```
df3b05
e23cd1
26130c
fdd705
da0918
f62537
1717fa
e6262e
2b18f3
fb0a32
f9fe06
02f9d3
18a6eb
e636eb
0b1cd5
f35194
092510
f5f6fe
0be80a
f03048
d00000
e7e721
0f1d0c
fd1cd0
0b302d
4521eb
431b17
e2e31b
d2ff22
e3ebec
fb1ae9
0c04cb
c62714
393838
21f41e
07003c
002602
001c0b
e207e2
12e6f7
2339ec
ea20ef
d4f520
1bcce3
fefee5
d13724
0e24c5
d93130
21dd1f
fdd9ee
2a2810
d8291a
47c7f4
d609e4
dfe128
1f3784
ffdc1d
fa1ef4
```

```
1d11f7
dc0b1c
e01031
e71cf5
df14f0
e9f805
d8ed15
493ff2
420924
e7fc24
00baf1
e6e61d
00e9ec
06021d
15c82b
0e44d9
1ffe1d
df0f1f
0debf7
e61301
e8b014
2c2af4
3de41e
020009
e2d39e
f4da0c
d31a08
c03a3f
0ed9c5
fe4832
ccd01f
3f0932
1cd605
13ddde
1ac2cf
f60a04
0b39f3
2960d4
1024ed
f20101
171204
110952
44f6cc
12e34d
15ea35
0130f0
ed0819
34c4dc
1ff5ef
2318d1
2ad726
07e830
e00435
1f2bde
221d3e
25d623
fc293a
0800ff
```

```
f02910
f4e400
ffdf21
dfef23
c82ef0
17cc04
f32d39
f3ebd7
f0081a
e01230
dfc725
07e5e5
ef3600
c5e5da
0be147
180923
1a06e0
fb0ed6
f83805
36e25c
e703ef
012c31
0bfbec
15e233
1fed06
db2832
0b21e8
b71240
12092c
3b0f31
3b36ce
161459
e1f0f1
1b0dfa
1513dd
0600f4
e3f01e
e11cc2
d63e3b
c82ad3
ed1312
220a14
d63724
b5f413
e20a05
02e222
3f270a
1ce71d
4635eb
0b2df4
f5c205
ca09de
e5000a
2ceae2
3de1ff
09f9ed
f43417
06fc1e
```

```
0c16dd
3ec105
2be9f0
2c2438
1efb24
f5e4c5
0119f8
ee3820
ededc9
def2e2
e43bdb
d6b9cd
dc09d5
d2fae9
181613
eaec05
3d11db
21dee6
f2d0f9
e6f531
4831e8
3228da
fadbfe
f821e0
fcef0c
d5cb30
ef1419
ec3001
f809e3
0bee3a
f7dd00
ce3d50
05cfe5
c30ae9
19caca
14f132
101c59
ea0f01
edbdf0
20e9fe
33d1ea
fde010
05dc2d
421a02
033df1
1ce539
fc211f
e92bd8
e7ee24
fc0027
e52f22
15cf13
1731f9
1ded16
160ee4
06391e
41e003
0be2d1
```

```
f9e5cc
cbe604
ea0638
080701
ee1cbb
1b03e3
eb1bf8
c03036
f6e6e1
0bc923
c4ec10
d618ea
08ddff
fdef26
eef300
f4250c
0706fa
3816cd
2f4402
fd230d
d70009
1efd37
1f2cde
1ff324
0ddc04
dc17ff
2ac8ff
2514cc
0feff1
fbfad5
0e1ae4
1429db
db0ade
211704
f9e10e
f00527
07331a
202ffb
1fe410
fb0108
310af6
16ca0f
00fde4
120d33
f7e2d7
212d1b
f7dd1c
dc181f
07ddf3
f1d747
0329fb
1c16f7
1ecd10
f526db
250c46
e4f0e4
30052a
15e726
```

```
e233b1
fdde24
0730d3
1ef834
f1431d
ebb62d
f6e6f6
f73b01
d9231d
f3edcc
17efed
06f6f4
0525f8
1ded29
eedc2a
d1f72f
d4100d
35141f
ce140d
14fb19
f3222b
e41241
03394e
1b00da
e7090a
2ae9e4
1d1723
2714f1
ddfcf3
4b0ae0
c33fdb
17cdd5
0adf13
2f0008
2b3821
edeb11
0117e0
32e30e
25f6c6
a1fc2c
dde33a
ea22df
052e12
fb3400
ddd8e4
f61ef6
0feb1c
03ce0c
f1d1df
e654e1
1815e5
e35a2b
e0dd51
00d611
ec0614
013123
3733e7
e21525
```

```
00e2eb
fbd6fe
e5f615
0f0f14
f93f36
003fd7
1e19e5
e92929
e2e919
17e9e9
19ceeb
3800db
e02ad0
30e6f1
e4d520
11c0ec
d649e9
d1e3d7
f314ea
071c30
f3eb10
0df517
13cf0b
e8e6f2
1500fe
f04705
0cfcea
2240ac
19f9f7
001f03
f224e9
ea3412
e4172c
12ed45
08dbeb
da0900
0f2f35
3c14fa
140009
cc002f
161c10
1d06e9
ff1b38
f10ef2
e83d3d
dcd11f
fad81a
35112c
1b2309
f42708
d32424
fdd7f8
e40bce
240317
fff207
0af2e4
e81a07
08df15
```

```
cb3332
dddbdc
37f8d6
e610e3
2adbe9
06fd0d
2a430b
ebfbca
11000d
dbe0c6
27151d
d7ecff
e60b27
e01c2e
00e900
254c2f
1be110
df221d
d8060f
39fc11
e71bfe
2ceb41
202a1a
fafb1d
05004a
e7f5e6
d32238
f5ef2c
00f04b
282361
e1f0cb
09f6ed
122c24
42385c
463e2c
1fff12
d224ec
0b4400
4b1319
51e635
b21cec
eaf900
190128
d60721
28e8de
e1e12d
0f1d0a
e6c917
d225f8
ec03b2
20fbff
ae00e4
f017f0
e02025
490411
1fd90b
d4e915
fbfbdc
```

```
27cbeb
ede824
fd0bf2
e5da01
2c3704
2403fd
f2e4e2
15e0ff
f002e4
e31fe8
f334f1
de15ef
02d1f4
1b2205
0843e8
06c727
2dfcd7
1fe3f1
042c0d
e228c8
f4fede
441c02
f7ea40
2318cd
2e3211
392c06
13461c
16dfeb
f4fc04
eac913
0aebf1
09f16b
e2ee00
001627
e617db
e3d419
e2dee4
cf0adf
dbe3e5
de162b
fb0f1e
f4f0e7
d1f834
e9db10
1233de
39eb09
daef35
e91125
24df2a
1623fd
e2f1e1
d2102b
f2ea19
f7e4f7
e34cfe
13270e
3dfdf7
0c28f9
```

```
2bfa17
0e48ed
f0cb27
fafae0
3125e9
1dd107
28e9fa
2e11fe
d209db
d323d8
dd64cb
0ce9fa
ebdb1a
d5f7cc
ef1cf6
fef7d3
19f2f4
ed1ecc
d81e34
28b7e3
08f30f
570326
f0efe3
06201e
f8f1ec
e6fce9
2d1403
d4faef
0bf4cd
0d200f
23d8b7
fddf4b
2c09f9
0de501
19e416
2428ed
00d203
00e90c
18fa1f
00ffcb
dc151d
e513fb
99b6ff
1d1820
fdf61f
e50824
4b15fe
c6edf6
1b180b
e209f9
ecb0c6
ddb3f8
db461c
fd9d5c
e0e721
2d292a
e1f722
13de24
```

```
30e8fc
c7fad3
180cf9
26dc08
07d026
2e15e6
e73209
200d21
ed23e6
1c1c10
332d08
d61003
e2fd17
f2d8d9
f21bf6
193c14
2afe37
31e61a
0914f0
fbe436
effff9
19e0fc
f609de
343933
f91005
31f422
2a311d
0309ca
1af3f4
1ef233
ecf9c6
f9dce7
e31531
df0904
2bea1a
0d2d06
fcfde0
16f618
1405e8
f400ec
e720d3
2f12d0
e30003
212bf6
e80ed3
ccb308
13f2bb
0bd2c4
37fb07
240209
3af819
f0e706
d5ffde
e61e22
0fd023
e4e02b
e3f0bf
242f1d
```

```
0003e4
e00001
552adf
0bf118
2023f1
e2fbe3
0906f0
30ee27
f60c38
2b11ef
0b1925
0404fd
0e20df
ffd8e4
00f012
dbceee
e302d8
17addd
18ff0f
cff002
0eee26
00060e
fb09fc
39e0bd
faeabd
13dbfe
2adacc
c9e021
edf258
1c02c3
172217
e62e00
eae607
1a0000
2fe9e9
e04ab0
d034f4
e305db
1b08e4
ff2501
f4d2fc
17f4a9
072112
251748
421dc7
e6220f
42c7ec
13311b
feed14
f223f6
d80e1b
3821d8
112a2e
e9070a
dc39cb
f2ec32
e1da1f
4d00fd
```

```
1adf14
1a140d
0b24ff
0aca26
27ea2a
bfc735
13081d
f3ea14
ed4bfd
2fe304
f5e626
f91da2
e3eafa
0df812
08301c
eaedf1
c90eeb
001a04
d83218
e30cf6
e6dbcf
f61114
fd4000
cbd352
af41f8
fcc604
13fc00
0fbbc4
1b0916
26e2e2
f0f0cf
2220f3
d3fa0e
2cf6d0
f9f844
1a071a
de2bd4
e40ee0
e50bdc
1a2d3a
19fc1b
040527
00f023
d20db9
d9fddd
ff37e8
ef03ef
f8261a
f1350e
281c1e
18fb1f
d2e75b
0401d7
2ef3f3
343bfe
141ed8
fdc8ff
0f0426
```

```
1df628
d1232f
fce115
e9f11a
ec1601
c4280d
290903
f832f0
3a00a0
030847
09cdf9
e60603
363145
ef1ad2
f11128
4f0e21
0b3be8
dbdeeb
00ec04
fb1a32
09f7e1
ecfd00
0ff1e3
f3092a
12fa14
dbe85b
1c01ec
0ab6ec
f127c9
fc2908
ef0605
ffa6ed
1b0e04
f20feb
2afa12
e5042b
2f10dd
1319bd
0d04f9
f1280d
12f4b8
18dbf1
0ad6ed
eb2416
e32f25
0e08f1
1a33ce
f10a1e
d8f4fd
030bc8
f92110
1b2639
09ffd9
faf008
f334dd
0b1aec
ee4f4f
0edf00
```

```
2eece8
ddae0e
031012
16c632
dee81d
10022a
25f9f0
d6f83f
d81404
e2f4d4
e7d831
23e4e0
f6e4ef
276208
122661
e5de0a
effc1c
ee1aeb
0d0f04
e8e9cf
fedef5
d906cf
f01fe7
f207ea
1a1bdc
d4f4e7
e7e3e9
09060a
fbf3f0
eaf91c
e70f21
e30c0a
f5e0ec
e9e901
06f9e8
001211
05edd8
d1e3ef
ff10f0
f0081d
e31600
2b18ea
12d4f3
05fecd
ef26e8
e21613
d9d4f7
f6e9f9
0feb2b
ddd9cc
f211e1
19e4c9
dd11d7
c8eae8
fd1ddd
cef5ef
0006d9
21d70f
```

```
46020b
0b13c5
ed3512
050f0e
2624fa
d302c9
e818e0
0fdf00
183014
0209de
1f31d2
e0121c
fc003a
f0d7ff
2d150e
2f1b0b
183728
db190c
08e81b
3710be
012d35
ebf0d3
1916ea
d9f21c
f90ae5
3a9ae8
12e443
f216fc
ebdae6
da02f0
f90711
031805
f51bc3
17ffe7
240fff
0835e4
d7fb39
ce2d26
2adc07
f70d07
1025fe
e0f0c8
d9110b
e304fb
1dfef2
fede27
15ebe9
e1f8e9
fcf5db
f80a10
10210b
da03d2
db1eb3
e6f90f
1709e7
de1c1d
083114
090820
```

```
ed1515
32f222
19f042
f5d804
2fffcd
e8f7eb
db1213
3b28e5
ea273b
d01ffe
e8150d
e01c19
3aec01
dedc22
272c07
3713f7
08012b
26f9d1
1a2142
dd0e0b
32310c
294214
01f8e6
e1e2fb
3bf304
ebe9ce
ed2d2a
fbeec2
0b1a08
ee1ad3
2a00f1
ccd701
e607f7
22000b
251aee
3907c5
2c0502
df0ef6
24df04
05f9f4
0723d2
3824d4
093b2d
f21cd7
d026fc
0b062e
11f508
18cd37
24fb0d
183c10
1feff2
dab715
131fea
bd2801
e621aa
f60107
0728db
c81b11
```

```
fe06df
401bd6
cbc609
10dbd9
10db0d
dfdedf
e6100e
d4e614
fcd717
ebeadb
dc05f7
06e11f
ead9e5
d22cdb
0aecf9
f7e71a
ebf00f
f206e8
15db2b
02d10f
ffd8fe
0009fc
f01b1c
f90fd7
df10e2
def2f4
f2f21a
28d7f9
fde4f0
d20307
e417db
e40ef2
e1101e
0d1cdd
d90ce3
19f7f7
1821f4
d01512
1ce914
190cdd
03c4ea
1d23c5
1af11a
dd3134
133201
f5f6e4
035815
1c0700
d9e6ed
f932e3
f7e3e6
0b0801
2c2e13
16c529
f0f620
032913
112557
e43300
```

```
1d1d2f
54f00b
e6e91c
d4e3d8
0ce905
f0090a
14183e
f0eaef
db1e53
ea2402
1602fb
2cc711
10d92a
030315
18ed19
2323cd
f521f7
14eef0
ead8c4
07d813
faef01
0724fe
f106e4
e7da35
1beef0
dd3536
ef3e12
d5d60f
08f2e6
2f00f6
d82758
1b1e0e
0409e4
31250f
0d11ed
e22560
ebdfde
29f6f8
073dd6
e319de
4bef15
2d06e5
f3dee7
11130e
1cfe1f
0bf9fe
fb010b
de2000
dae407
d90b36
27d6ef
342025
0ce9ef
fc1cee
e6c911
ebd208
ee06f6
09f6fb
```

```
fce5df
ea1dca
efe709
eebff3
fa0dc0
00f80f
f9ede9
0ff217
12d3f0
e6d8de
06f9e9
fb0bdd
1fdc22
e800d4
e804db
e2e42b
23ed1d
de2308
0bcee7
24e61e
edc612
0e14da
0d17d5
f726ee
0e25d0
171917
11dbe9
e1daf3
e3f1f2
290afc
14f808
2708e3
faede3
10e2e9
0c23e4
fafbf6
d4e2f8
1b071f
e3282b
0ad620
0711d3
e924e7
110c09
01100c
d3dc1a
07e919
18f2db
0beffc
2603e3
e4f8e0
22eae5
0ef0d1
0025f5
1b34fb
090309
e813fa
1e3700
06e334
```

```
02d0e8
eb0e1e
e02203
103132
370a28
47fe19
45fb08
1ee4dd
dfd7eb
2a1425
1fe408
09dd11
da2c1d
ca2e2a
ebe925
2cf71f
2014ef
ed06e1
1ecbf5
de17eb
0431f8
2fc801
04f1cf
1717c2
0f292b
d8d3c4
2bddbe
00020a
031526
07f511
0529fc
fad8f5
0a07e5
f724dc
1121d2
9cdee4
dd00db
f501fa
2936ee
d1c514
21e5ee
fbe909
061a1f
fb201f
20d722
4fd700
2054fe
d120e0
d5da2c
3116f3
e0e00f
1311e6
bb1326
1ffc01
2717ef
093a18
21df20
e0efda
```

```
2c0e1b
2015c0
c5f8ff
2714dc
0327e9
f503f7
191eb3
d1d6da
f7020a
070010
f4e606
07c4f6
fc24d0
28e2e2
bf00f5
dbf220
d2d71b
1310e7
fd25de
ff0527
0fe4fa
e2c20e
e41ad7
f71ee7
e809d9
02c5fe
fc14e2
f917f4
b7fcec
f61cc0
1808de
ed00d5
f2e0e3
0127f0
1ac7dd
ee13e7
f203f7
e61311
fbe9f3
f4e514
ebfde9
12131f
e029db
06ebff
e4dada
cbeccd
0bd21a
d5d804
f9e0ca
0cd7d5
122417
ea0722
d8082c
051637
f50ddf
280639
2ddb0e
262ae9
```

```
16e417
2e27d7
f011e8
1b0b46
f205f5
2808fb
d7f9f5
2b14fa
252121
19d915
3a17fc
3215e8
1bc41f
dc254c
dcd0ea
02fb0b
000436
dd32f4
0f0f1b
ca0b14
e2ebe4
decb11
1ed8ec
e1ee20
140b04
01fdd5
f7d238
f01210
dad1f0
e1f3f2
1133aa
fd0c06
c1df00
f60f0a
1218dd
ed1317
d2d70f
2705f4
d3d600
07f414
0edae2
eae7f1
27f6f9
2602e4
f7eef6
d7e3e4
12d8e1
edf7e5
cf1518
e8eff1
e723e2
d50828
15f4e6
f02ae5
de000e
27e6fe
db14ae
04e0f7
```

```
d4f424
1d0ec9
2a0001
e2dd11
d61626
f4232a
fdd4d8
d6c7de
f8e208
08221e
2adb00
f2db02
fd0215
bc07b7
0f23fd
253201
2c00f9
20ec19
17cced
0102bc
2a0120
e5eacf
edde1e
23fded
d70107
1a09b8
e6dafd
0518fb
16f2e7
09faf2
16d725
be0ee9
fae92a
e820e5
ef2ae8
eb0ed4
040231
d8f81e
cb0cd6
e312e0
2b1c27
41e159
e40233
fb340f
f72625
01ccea
29e70d
02ea4c
1705e9
e1d5ed
e52e09
1ec614
dd25ed
f4f7e8
fb3ad4
142e30
26f901
d6fb05
```

```
330dd2
ff1233
0907eb
fe1800
f9edc0
2af600
fc34ed
e0fb20
3c0028
e62ef4
d91dea
1f2b19
f10503
12f51b
3c2cdb
070a09
1afd23
e2f003
fe423d
0bfafc
2417c2
04f4ef
ddfd1c
05e847
1cfd22
e92106
e4f9e9
f22d3f
f9f2f6
e4e8de
19ec00
22e0fb
e4f1e7
f929d6
e824fd
05001f
0a051a
114efa
271134
e312f8
160ce3
ee2cd9
ff0d02
d7fdf3
22dbf9
fb15d9
e746db
f30e4f
db3600
eac601
df1500
cf3bb3
2aec05
2ed1f7
e6eeee
00feea
e10602
12f2cd
```

```
14ef1c
eb1ee7
1344dd
22ec3a
020f08
1aee30
fff000
02f40e
ecb8db
e4f01c
33f9da
db1844
f6dd08
3b14f2
1907ef
f40728
11f5ea
f3d61a
0ff406
dd17e5
221822
1ce6f9
03ebdc
e0e700
0800d9
ce1832
08e8d8
ea08f8
1fecf6
c311e8
26ee0b
fd3400
ec24dd
df040b
1516e1
ddfd09
cb19f7
c12bd9
23111d
3be421
f82804
faee20
2ef004
2925d3
fdf2e5
1d0fcd
d6f535
e810d8
d82e2a
12e70e
ebec19
2f0011
152fc0
2211cc
0030e3
f826fd
f7d1f2
f01ae4
```

```
ddbee4
ed1af6
00010b
22ee08
d2d423
1d13f0
b93526
e02b21
e9e4f7
f70d13
ff392a
ecf44f
b72627
05f9ef
bdf836
dee6ed
30ec26
0602eb
ec01f4
0b00e0
c42f0c
e9e31b
16e414
01e6e4
0d3c18
153d1e
ecf41a
e42f4e
09212f
db1727
14ecd7
2414d1
e8e500
fb31e2
1a01d4
d2fc0c
d4fdf9
0e3eff
d7d10e
abce23
1ffb17
113814
20eaf0
ece423
300eea
fe07cf
20f2dc
23fada
0fff38
3805f2
0920f3
fc0fdc
311119
040d0c
fb1826
0352fd
34fb14
011ee2
```

```
f1152d
28010b
19dbf9
cef21d
fa17b9
29e5f0
132f2b
14ebf6
051ffe
d82d2f
e7ddeb
01d8ed
17261a
e2e5d8
00eeed
2ce7ec
210b39
0004fe
04f30c
d4ffe8
22fb26
17f4fe
f31803
f21b31
3646e2
ede5f2
f009f8
fa3cd7
f62efd
eaedfd
0613ed
16fde0
ee2af9
000418
1632e0
071218
01351c
ea1043
fd2e11
1d27e5
04200d
0f32ea
f5c627
f5ec20
2e0408
fbec17
23fafb
ec1d22
44ffd7
18e5e9
e2f52c
34ed0c
e6fd00
0213f5
e91308
f92dd5
1adcc4
cfd8cd
```

```
d92fe1
e9d810
08fcfd
fbe2e5
f50b14
f1d53a
ef2312
040944
26fecf
1729cd
2329fa
da46e8
00f917
3ff812
271915
1ac5ea
3edfe7
163709
1b47d5
060f50
202626
1207d8
340e1c
f718d4
03a924
0df9fb
1a0d08
fe1c00
bafc23
00163e
360113
382115
f90808
00222f
2117e8
fe0fd3
17fcdf
f631ea
f7dcec
180bc4
fc0e1f
cddaea
f1ebe0
f918f3
dd09f2
f34dd2
040c06
ff13fe
08d7e3
0401c0
1426d4
32efd0
e72cef
0bfc01
cf2a1c
ea40bc
0f30e0
0a1111
```

```
04021e
113aba
20e20c
0dfad2
2700fe
d3e724
fe34f9
01f918
ea2008
c92711
ed0aff
daec16
0206ea
f5251b
f000f4
2726e9
fefbd4
0cee09
ee1e2f
ead001
0b0032
2d06fb
15fbf4
351f2b
ce1204
ece5bf
43281b
063912
1421d4
ed0fe2
0ccce5
f00e36
2a21d0
19e4f1
e7d3dd
fc01e0
15d515
b71c36
e9060c
e0d5e9
0c18ee
162ece
c2d912
b72019
10c32c
f433f3
f9171f
f1e9de
19411f
e2db17
e70019
0812fd
3039e3
ed9b2b
db1a46
ff3f0b
f7fae5
dbb823
```

```
25dd12
1b03e1
00fa1f
fedd41
ee152a
dbebe8
eb35f7
2000f2
01cd19
ff0ae3
db14f7
11ed65
fdea26
09c8fd
183c1e
132d38
df1e08
d4f402
fb30f5
3be83c
cb2df0
09d233
da152d
281b02
673d0c
0fc1d9
0a0deb
34e9e7
f31c30
19dc1e
21de3f
e6f4f8
c91b1d
004f1a
2fc808
004f4a
15dbed
faaceb
22f6d6
1248cc
03250d
3c3f04
ebe00c
042b1d
db07db
d64720
20e7e1
af2d55
d7dd27
1d1fe1
eb1600
f6ece4
e4ddd6
d811d7
eaf2d9
0e07ef
edece0
e50fef
```

```
edf80c
e9f525
de19d9
10dbf2
e7fa01
d81700
16da26
16f6db
22f3d3
0dfbea
de23d5
f2f1dd
1a2217
dadbe5
d11efc
d6d6f6
14000a
d8e010
1405ea
0de1d9
080005
e31df3
1203d7
130af5
e10df7
18ea04
1fedd8
dd0ce6
22e0e1
181ee4
fbe0e9
05ef1a
dde3da
f9ded9
03d207
001be6
1604ff
23eddc
25d6fa
fce816
01e5ea
d1ebf9
f8f011
07d203
e80bf0
21dbe1
e60b19
2cd905
efed00
22e7e7
e402f5
2208fa
0a061a
f80223
eb00dd
1422ef
23ec06
0aebfe
```

```
2712e9
def6f1
221f18
07ff08
e62403
0b0810
fa05ed
e92514
05f6e9
030d26
01eae5
202617
d1301c
d4fa11
b8f719
d4e142
fc2a45
27e8eb
ee30e9
22d2d1
11e927
25382b
1b0ae3
4cdea3
e138e7
17370a
fce222
04ea1b
1f34ec
f0f4d7
e6f3f1
1de71f
0c37f3
ee121b
22ddd7
f5fffd
1af04e
fa1a1f
f30edb
140bbd
e1e2f1
0dd1e3
2b2827
0f0ddc
f71fee
f20b00
f1224e
2ee51c
022422
311c46
341727
ea05f8
f141f0
2831f5
361003
ff1bd8
23470c
2318de
```

```
fd2dee
f1e1af
0a1b05
0a2ade
fbe901
d1269e
f5220a
ed3ded
41f543
c8d9d1
d9ef19
11010e
060d42
1d120c
13fde2
d41cd5
0b2be8
f21500
171216
2efbff
301fea
21ced2
e6293a
421500
0d0e15
1f2606
c41409
dbe637
efde19
001a2f
f8e1ed
fa5524
f22757
2714d3
2eb0df
23ccec
0e09e4
e0ddcc
1abbf2
f7ff09
40ea1c
cbfdfd
1c04ff
f1ec14
e600f2
d4e722
1ed811
110bdf
ee0fbd
2a1118
13dee3
fb1b1a
00e8ea
f73de6
06c23d
2ff52f
0feb20
f1dade
```

```
cad61a
e14b26
0b2a25
ed21d7
17ea1b
092112
db2a00
24e8f6
2bf330
2bc72e
08e604
1efb10
f43113
05f7b1
1fdd1d
ff1923
250cf9
263603
0cf923
fd0406
efdbd9
19fff8
d2d624
18ea2b
00f4f7
f91bfd
00f8e0
123220
37de25
e0ec00
fcf5be
2b24cb
bcecec
c20b37
c11c04
0b1123
e61220
e1e810
e5e7f7
f700e2
1cf5aa
03f8f1
db110c
273f09
f5150b
e708ff
272826
23d22f
122e0d
cf011c
152e37
1cf2f6
00cf16
e9f1f1
323115
1c0008
31132f
d4c000
```

```
c62cf2
25d238
e7e609
04fde2
130732
262d29
12e74d
0716f6
dfddd7
df1dee
e60efb
f129d4
13d5db
ee1c4a
e5130f
cbfe2a
07e5fe
e405e5
19ca0a
f4020e
d3e4f9
d32423
1aca1f
f249de
151517
f944e7
1a08d9
000f36
251b03
e92331
3aeed8
2e131d
df04ed
f7e9e1
da2a31
20d6e1
00f1db
f71f2b
f0ab29
faed07
19e617
ff1dc5
030a3a
ddee1d
ea2326
e72062
0aefe2
dacee6
02e120
172b12
d1d220
0620f9
d0dac0
d8df39
0d1824
580d33
0b00de
0af830
```

```
d7e456
fcff12
25a8e1
e4e0ee
371e00
e6f6d3
c10021
1a27dc
08210c
c6152d
f611ff
ff2a01
2be621
ec0022
de020c
3b12dd
f6e703
270bf6
200f1e
e1ff05
113712
1d2e1e
0cbe0e
f6e75c
293333
22fbeb
d2e9ee
e4e104
e514c5
0728ec
f946e6
33e41d
0d3927
1d062c
1901f4
f230f9
3332e6
1aca20
463508
0ac117
1cfbdd
f2e831
0c11d0
07e3df
d308fc
03f127
2a08d8
c10d5f
e4dbbd
ded2eb
d1e0da
2b04c4
0affe0
440ace
e0032c
4827fe
10e3cf
e8352e
```

```
dc35e2
f51930
bac41e
0beb0c
ccf12b
05250b
35faec
0f00f5
2e092e
3204f2
09f516
1533d8
dde6ff
0034cc
e4ead8
e92ce3
15acf5
f931f5
dffeee
b2cbcc
12f2ef
0bf6eb
f70916
f7daf3
01f5d7
d9f808
0113cd
20463a
ed33e6
e113f3
fc1cf6
dccef9
e51409
241205
23eaf6
fe0044
f2db32
fde043
123436
f00329
f028c2
2a0509
e6100a
0ff207
d00918
db0507
f7e61d
e0e30e
0cc4e0
e8dcf5
def414
0af5d5
d70ce2
0e0afd
effe24
27da09
14d8ec
2bd500
```

```
01f6e9
251cd7
0208f3
f6fff5
07e2cf
d90af5
fef30c
10152a
dbf2cf
de1d05
c50dd5
f015db
1bf3dc
00d3f1
efe90a
14c815
e8f31b
06fe17
f319d5
111f15
1101d8
f6dfe3
101603
de1f1d
242ef4
231be3
3be7ed
f325fd
e9e2d0
072501
2438ee
ebcd06
22e8d2
2c1a29
1125db
0700dd
f9e00f
e0df04
f316c2
e1fcd9
ead000
07f428
2bfcf2
18df26
1102f2
330000
28f7fc
0414f5
10120a
eec1ec
f82efd
12e96d
00fe2d
0e16e8
09eb18
d60c32
e91ddc
d6efec
```

```
2cf21d
11f332
29ee14
00ef21
f02801
f813f0
e7de01
0be027
f20c07
2efbdb
1944fc
fa19df
0bdef2
08d81a
2718f8
f60025
0aceda
1311f2
fa1c0f
fb16f5
16f7ea
cec42e
0b1c00
2415e5
160de7
d20035
e61dd9
23d7f5
300a0f
1e0ffc
3b030a
2b0408
d61120
2f139e
e20cea
df01de
1c391f
f903fd
e52e08
07002b
e9f0ec
28ef27
07121e
18170e
2ace28
bdf6f3
f3112d
03f1f4
ca2d2e
ef05d6
171f33
35f514
ddf121
ef27fc
060e30
f40506
03f222
0c00e4
```

```
0719fa
0a04c3
0c002a
0ddcf8
f537e7
f52c00
badfe8
d9f03e
261e0a
ddfe31
06e1d4
000421
32c90a
e40e10
dcc4d1
0f00c4
10e41a
280c1f
d12cff
f71f3a
23ecee
e700fd
142e2c
062900
f6403c
1ee914
33e314
181be4
1bfee1
24e61d
d2f3dd
f21715
0421f2
e3d628
1bc508
cbebe5
08231e
efcd00
d71cea
04d314
e6eaf3
ddfb08
f10bf7
2a06d7
ebc9ff
24de21
00dfec
12d6fe
e2f224
1edcd1
12d4ee
2a0008
e10adb
d6e21a
0016ed
290a14
d511d4
d501f2
```

```
fa11ff
26d4ce
c7072d
cad8da
f9fd07
cbfb00
19dffd
f3ef12
f503d3
e0fed6
232a23
13060e
28f2e3
0e0eda
fbf2ee
1d18f6
01ffd0
2de0a9
131a30
25271d
0d0207
c8e497
052522
e3fd2f
222b14
03d70d
0c1f23
0600fa
d9d21c
180acc
09ec12
24def2
203521
1e1001
0cdbd7
1efcf5
2a2a22
0ab7bc
eeda0b
283ef2
e9360d
29e32a
c6e612
fc191e
1705f6
d817e6
022e11
0b3704
0eff1a
30deb4
05cc03
05f3c9
d5da00
e01000
2a02f3
f005d1
f12c3a
ec1929
```

```
132834
e8d0f7
d8d9ee
0eca06
1403f8
f8f228
25b8f0
070e26
fb15f5
fe10d7
351ae9
0608f1
ebdefe
27dffa
fd01f4
28ed11
c70500
004b04
102af5
31f217
e40d09
eb0f26
eed60f
e4df05
faff36
ec123b
dadbf4
e327ff
1f0e22
ebfae9
d4f2da
110c26
c3ffdd
0914e7
c10005
27ff2e
2c482e
f0fee7
d915c4
f417d0
02ed0e
2d3407
db2cde
f4038b
c30000
262a0b
32001e
001ce3
1cebf3
15d0be
33e105
260ab7
dd0718
fb01e7
0dd2b5
ece81c
ed7f15
0be5e5
```

```
2bc91b
ddd90f
1415c6
fa11c4
dcf805
220930
d42313
fc2b12
fb0e3a
0e1129
23e6dc
d9f609
aec6c8
d00322
c1d40a
1b15fc
d1b914
d61d2c
e02610
0ccbd7
ac1e13
030ee7
22e908
f1f7ee
ebdded
ee2ec8
a3e32e
0f0d2f
2bcd1c
0301f1
08e9da
ebdccd
11220b
d6bef0
ff30ef
ce2402
26db2a
d6dd19
1f3c1e
ed00f4
ea1815
f22332
efff0d
19e0ed
0b02e8
fa2c22
0122ff
261948
eb231b
ea000d
fbc82c
293522
f3e8da
fcfc16
d105da
ece01b
e7d422
ec271c
```

```
2922e5
150d28
272804
dc04f0
fe1239
e3e4e5
19d6f7
0fd6ea
07dae2
051406
36f0f7
140900
1cfa04
e9240d
e71f46
d804ef
db00f5
3729f4
2f0009
0cd7f6
e53916
ebf30c
252eda
f1ced0
03f016
002632
122fff
14f6dc
f6f23a
f1e117
011cde
23f1f7
00ec07
e01b31
16dc3e
01dde9
ffe3e1
2dd014
34fb19
eb1ff9
be15f5
f41041
f20cf9
1efa2a
09e6d0
d31b1c
0012e3
e30740
d6f602
062eed
fb1b3d
e30fed
072b27
4209ea
cf3e2e
211af4
e8e634
181bf8
```

```
001d25
00e31e
21393e
ecf7f0
02bffb
24feaa
0eebff
dc0df3
1d2fde
202e26
d8ebfa
1b35f7
2209af
d12418
133826
461fdc
15f335
f109d8
fff1fd
0e33ed
b5dc29
1e1145
e23640
e500c3
cf25f0
2ef2e2
18f9f7
f1e11e
14281e
2e00f3
e0f6fe
07f00f
0d26ee
154538
3f2bde
24c6d7
2af618
f9e0e6
fde1e2
c4021d
f404c5
170a30
ea1b20
e21934
0be6f9
d34921
e7dba8
2deadc
d7d617
052ed3
e3dfd3
443411
d71b32
0200e7
d8e611
0023f0
f322aa
cce6e0
```

```
c519e6
05f1c8
c2e23c
e113e9
002100
d1c5d5
d023f6
e3dcf3
0cf410
18d71c
1ade04
1a09ad
bee221
0acbf2
02d942
1ffa12
320308
4997c7
15eb1f
21c3df
d61ffb
eff1fe
28f8db
e2f51d
f9132a
0a1f1f
233a19
fdeae9
d505cd
0edf08
fd212b
1018ce
10f3f8
e8db50
d6fbe5
1b0062
f2acf8
3f2e0b
0904da
20e738
30e1d8
16e60b
dbf2f7
231411
1cfded
060af2
e6e31b
e9d207
1becd0
091dfc
1fea28
d6f3d7
d6db1f
ffcce4
cbef12
db13e7
170b1d
0ce7eb
```

```
c9e3d7
11f7ec
fcf3da
e10d08
17d1d0
29042b
eb1cd0
1f0adf
ff0d17
e00b1c
d12fd7
dc08eb
04f2e3
08edfd
161d12
fefdf2
21e6fe
04100c
1406fa
f9f71f
fceadf
f1efdf
363711
2e4ef4
3b110a
e32b1b
1f2a18
14fabe
ce36dd
082a0f
1709cb
1f090f
ff35e0
dbd118
f1360e
00f123
efff27
f820e1
4ae916
33d7d2
460b04
ecf2e6
fb0f32
241c0a
24ec3a
0100fe
dd6b18
d50438
040617
f0cf76
18d831
0df32f
1fc4d9
f7d2c7
e2db22
f6fec9
2d20cb
26ead9
```

```
eee6dd
e03dfb
0405f7
d6f9f2
e0c81d
e847d5
f3e6f5
ecd225
0615e8
ed22dc
e01f1f
ff28f2
0d12ef
ecf627
17ebe3
ee3b37
bc36f5
d9dede
d9d736
e5edf3
f2ffe0
e4f01a
4fd8ed
1afd1e
12071d
341d1b
322954
fd09f4
cef1e7
173d30
fde410
09fc4a
fd1110
31e72d
d7ef12
0b1ef9
ddf238
ce41e6
281308
360617
fadfe6
ee03e5
0720d5
0b0b57
cb13e2
d6e1f7
ded600
e4d8d9
d3cd16
00d9f2
00ced2
f0dadb
d8e9d1
e9df26
20fd06
d801e3
fe2029
101c06
```

```
cf07e8
e21f0a
cb0ff6
e9efe6
0ce518
e90c01
f7ddd9
e42102
16cce1
23e81c
0015e7
ea1cf3
ee04f8
d7e921
e5eafe
e7cbe5
d6031e
120f27
f116fd
051b19
2efbe9
1af2ef
ff03d0
14f900
fae5cc
d4dafe
112ef1
d8da36
2a2de3
04d4e7
04ee0e
32e54a
2200e8
3bdbed
14fd07
1f31da
0afed5
3036fd
060cf7
0b1e18
e4000e
f5dd38
af221e
e63933
16dcee
f93d3e
37dc18
c9e3f9
f4feda
fdd7f0
1fa8fa
11e0f9
1910d8
121add
eff412
0ae814
22fa00
d83930
```

```
e120e8
26fec8
fcea0f
f7ecd9
fc152c
ebfe19
2b3939
e22fdc
f9ede0
2721f4
2d1317
fcd2dd
f1ed24
1921f6
1bfc06
d8dd06
1afefd
0a19f3
2a3a05
10d212
f11601
dde9fe
40c827
0008f0
3a22f7
55f5db
380036
ef05fe
de19f6
572be1
45e5fc
ee2312
ff3929
d7f328
2e00da
051af2
1b0507
0c2015
f2d921
1630b8
cbf623
fbc8f8
dfd3e6
25faff
22032c
1205e7
fdca48
1e190e
eb05d2
f301cf
160616
f40fdd
05ed07
12f34e
0eea0d
ce2106
15edb8
d4b6f1
```

```
220d0b
d5134e
39e914
2017e4
c627fc
112ff5
200600
11142c
ce57ec
2c201d
0fcff7
0cda34
e71715
eb002b
1af7dc
29eff6
e139df
e4dbff
f1f2f7
afeb30
f50b2b
080c07
42ec22
ffedd1
f60cd0
3edd2e
00ebe7
10ffe1
152bf6
24de2f
1cef1c
defcdc
ea41de
20e516
efd8f0
30e9c0
db2bd7
1e1af8
0b2402
3fef26
dfe60d
cfb8d4
dde2f9
262500
270047
0b001a
ed2f10
02fafa
15021c
de35e0
021dd0
eadd19
1a0617
ef06e3
25c120
f24de1
3e26e2
3a0144
```

```
1906fd
1a031c
fff728
1cd2fb
0f0310
c547e9
d523ae
c1dd27
1e26e1
11ecfc
150cf4
ffed0e
1e00fd
eed9e6
1d1ae4
f8edd5
e300f3
0d0ffe
d8dbd3
fff0de
12ecd6
20e41d
1ee214
e7de1d
d9df0c
25f6d5
fceef5
08ed05
0c1411
fd0ad1
e601e2
dde5d6
e400de
00e30c
ef1105
d8e820
e3e7d8
f8f7f2
fa000c
0301f3
01d2f0
0b29e7
1904fa
d8e4f5
eb13f7
ee2200
d51a17
e42b19
03f013
f2def5
e00b29
210803
3910dc
28db39
17232d
f40745
2a2631
03f2f5
```

```
0dd21b
f23216
d81119
00eb1c
dd2e20
17cc17
421725
f8def8
2f330f
fc083b
2320eb
2a1d17
fa160c
fc04da
48d701
08d640
fbe7a9
efea28
f2e534
eb2a06
46231c
0529fc
d7d0bf
e5fe21
210d07
e32ff6
01d613
3323ce
ecf609
d5dd0f
1a0cf8
e1d511
fef8eb
f4113c
282507
ccd334
1e22ff
14e528
200a06
153d0b
2946ce
0417b0
f52c42
2e13fb
df1b1b
30b7d7
e10df5
1a2101
42306d
27e70f
e548e4
f4ecdb
260bfd
1f1bd9
f41d00
da2429
1ad634
1be008
```

```
d6c940
cb0030
31e50d
e7dc35
00f1e9
0ffefb
f82601
13f5ff
e002e6
d25a00
1104ea
e804d8
f3fd1d
ee4334
ea0fe9
0edc05
fb321d
21d939
dafa12
eaf5fc
cb031b
1a17f2
26e90a
133017
de0f06
0930e1
e2f91a
2a16f2
eee4f0
101c21
0b26e8
e6feea
d7dcdc
091839
e4d3e4
fb3d1f
e7080c
08edf1
fce3e6
fad200
f8cfd6
e6db1c
0d0e09
1712f7
0df40a
270817
1b3227
28f332
f0dd0c
000a11
e90d0e
c20b16
eae535
00fff6
e91601
27e4d8
000000
0000a5
```

```
00003a
000045
000007
000005
0000fe
00007e
000032
00005f
0000ca
000011
000077
000001
0000e2
000042
00002c
0000d4
000019
00000f
0000c9
000051
00007f
0000c7
000067
0000a5
0000e1
000057
00001d
0000af
000049
000047
0000ef
0000f7
000096
0000af
000059
000098
0000c9
00005b
000039
0000c5
0000c5
00002b
000012
0000cc
0000c8
0000f4
000008
0000ed
0000f3
0000ee
0000e0
000006
000002
000003
00001a
000053
000056
00009e
```

```
0000ad
000044
000038
0000fa
00001a
00001a
0000c2
0000fd
0000fa
000065
000005
0000b7
000002
00001e
000018
00000b
00001a
000030
000002
000036
000001
000046
00004a
0000d7
```

Listing 25: Packed FC3 weights and biases for three-lane FPGA ROM loading.

```
e0229a
2b033a
a2422a
391d2f
49c001
ee0524
05cf48
340f9d
d21e1b
0fc837
da17d0
18afb4
d21616
e5383f
d7ea00
47b893
f037ac
21beb1
e71f21
d51800
12d92b
21b3ed
f6070f
d5cac9
1ee4e9
993bc7
ce9c39
de9b10
d2f59b
f7ded9
424cf6
```

```
e74b1d
e4f00f
f5220a
a7bbfa
dc6519
44f515
f24616
ec3d17
101313
27bfba
bb6329
29a26e
1fe8da
d8f248
1be4ba
e38da8
16d3e2
d3ce2e
36dfd0
0818be
f39506
5947c3
3d97f6
e7e3c6
ad05ba
ca4326
fdf41d
9eaacc
442a1a
b1afe5
26ffc1
31da58
0bc241
101658
19f30f
d6fad3
f7d9f3
090adc
e70d4f
f228c5
3332de
fc1703
f64c13
a6b4f1
f82114
ddf551
f61110
023552
d72ab5
f21add
1a1e33
fcac0e
effcbf
042cc5
b1ccaf
3c290e
30fa1d
e3c4ec
```

```
ac1ced
0e23fb
368b23
df0fb6
c90de4
fb27f3
fdf9ef
62d6e3
f689e4
281a8c
319e3e
2ac5b9
fdf9f6
ce0cf9
f610ab
08cd17
4140e4
26f399
e1b8a1
b4e300
04f3fe
f82fa7
1c1a16
f6f3c2
d0c53b
fd22c1
3be381
3b223f
1ed209
bf000a
8ceb8e
0ad8ad
f1e9d1
fde501
dce7d6
ed0608
0003aa
f60f02
dd02cf
323338
bb0553
e3eb00
0a4535
00fded
3ead0e
356c40
5b26f1
1307f3
36cb1c
ebecca
c81018
27181d
253cc4
02ca54
c7182b
bf3ffe
13a9bf
bf05a8
```

```
34fa1c
0328a5
24912a
0d12da
bf1c14
dc0aed
0ff0f8
10ea24
b70023
e91133
53b6f0
1f3eb8
1b3daa
1116fd
3cd1cb
4c01d1
17d6df
29a1f2
2b1104
0ddafb
d4f3fd
15e53c
553df7
e24361
b1cebc
0b313f
f2aa3a
2ce1b3
b31c4a
b2bdef
ebbc4f
ea1a85
e205a8
1335de
d25ad6
d9e12f
c91093
2ac6b7
dfd714
40d742
e417c6
4c0af7
dfd601
b64fe2
dd0005
cfb21f
ef14d2
c39bfb
18bbc5
b713dc
f792f9
3199b5
4525af
fabbc0
24174f
544aae
37ac3f
20f2d0
```

```
e62304
e34436
c02d0d
00c1f1
1d390f
2813b1
dc1321
fe1bf6
ebe6d1
4baac1
c2991b
d5fc15
080bd2
2f0c1f
30f427
8447f7
3fb3f2
24b84b
cec128
28060c
4b4cdc
f211bd
dbb82c
31251c
2dfcf0
00d9e1
dc08d8
51c52d
ead903
35c512
2bf0d4
dbc5db
05a70d
3fdeb3
d8c5f7
25dcca
b74a15
cbd1a1
0cc8dd
431fb9
b953dd
2ae7af
d5aa44
c4e1e9
2f3abd
2bfadc
f43d1d
39e3e7
d40ce4
fbec28
f7ce2c
d8dd33
1eea1a
d34231
ee1e87
befcae
20c0ac
ea3cfe
```

```
e7280a
b5053c
d0c003
342526
a7a949
d9283f
b3a214
bc0031
f8eb45
26b3d0
eac0fd
d78dca
9240ba
0f1111
f53512
23efc6
143c46
000021
0000b6
000098
000095
0000d4
000081
00006a
0000b2
000053
00000a
```