Kanto Player
CSEE W4840 Final Report

Kavita Jain-Cocks
kj2264@columbia.edu

Zhehao Mao
zm2169@columbia.edu

Amrita Mazumdar
am3210@columbia.edu

Darien Nurse
don2102@columbia.edu

Jonathan Yu
jy2432@columbia.edu

May 15, 2013

1 Introduction

This project presents an audio player with frequency visualization, implemented on an Altera DE2 Cyclone FPGA. The user is able to play audio files in a custom encoding from an SD card and view a nice visualization of the audio frequencies on a VGA display, similar to the visualization on classic music players. Our implementation uses hardware to handle audio output and frequency visualization, and software to handle user interaction and system initialization. The user can interact with the system using switches for color mixing and keyboard for fast forward, rewind, and track selection.

2 System Architecture

2.1 High-Level Overview

The following is a high-level overview for playing music and displaying visualizations using an Altera DE2 Cyclone FPGA.

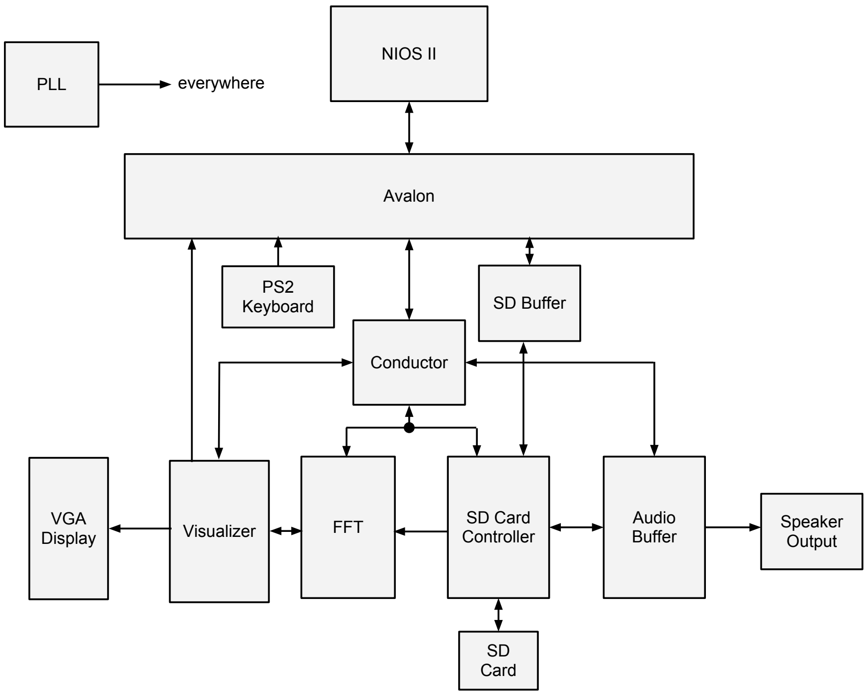

Figure 1: High-level block diagram of the full `Kanto` system.

- Data for the music is stored on the SD card in a time-domain format. When the FPGA starts up, `SD Controller` peripheral performs a series of initialization steps to prepare the SD card for processing. Once the SD card is ready, a signal is sent to the `Conductor` so that some cool stuff can start to unfold.
- The `Conductor` peripheral, along with the `NIOS` peripheral, controls how and when data flows between every other peripheral. When the `Conductor` receives the ready signal from the `SD Controller`, the `Conductor` transfers control of the system to the `NIOS` so that software initialization can occur.
- Once the processor gets control, it reads in the first block, which is a track table containing the starting block address and title of each song on the `SD card`. The track table is drawn on the VGA display for user control, and sets up the selection and current track indicators. The processor then seeks to the first track, reads in the first block of audio data, and then returns control to the `Conductor`.

- The `Audio Buffer` operates on two blocks of memory in the block RAM. At any given time, the `Audio Buffer` is reading data from one block while data is being written to the other. The data being read is pushed to the audio codec at the audio sampling rate. The next block of data is already written by the time the current audio data is finished being played. The buffer then reads from the newly completed block and new data is written over the old data that was read in the previous block.
- The `FFT` takes the same blocks of data that the `Audio Buffer` uses and performs a 256-point `FFT`. The `FFT` converts the time-domain data to frequency-domain data and stores it in block RAM.
- The `Visualizer` peripheral is used to display the frequencies calculated by the `FFT` visually. Since humans can only hear certain frequencies, the `Visualizer` isolates and only displays the frequencies detectable to the human ear, in this case the first 32 frequencies of the 256 calculated by the `FFT`.
- When the `Audio Buffer` is done with one block, the `Conductor` triggers the SD card to read in another block. On every fourth block read, the `Conductor` triggers the `FFT` unit to recompute the frequency values. When the `FFT` unit is done, the conductor triggers the visualizer to refresh the display.

3 Design Implementation

3.1 SD Card Controller

The SD card controller is responsible for initializing the SD card's own built-in controller, sending read commands to the SD card as necessary, and receiving the data and writing it to the audio buffer. It takes as inputs the global clock and a 32-bit SD block address to read, and it sends write data, write address, and write enable to the audio buffer.

SPI Bus Protocol

SD cards support three communication protocols: SPI bus mode, 1-bit SD bus mode, and 4-bit SD bus mode. Our implementation uses the SPI bus, which is the simplest; the controller acts as the master device, and the SD card acts as a single slave. The signals on the SPI bus are:

- `SCLK` - clock, controlled by the master
- `MOSI` - master out, slave in
- `MISO` - master in, slave out
- `CS` - chip select (active low)

The controller communicates with the SD card by sending commands over SPI. The controller must first send a sequence of commands to initialize the SD card. Once the SD card is initialized and ready, the controller waits for a signal telling it to perform a read. Once the signal is sent, the controller sends the read command with the SD card block address as argument and then reads the response from the SD card.

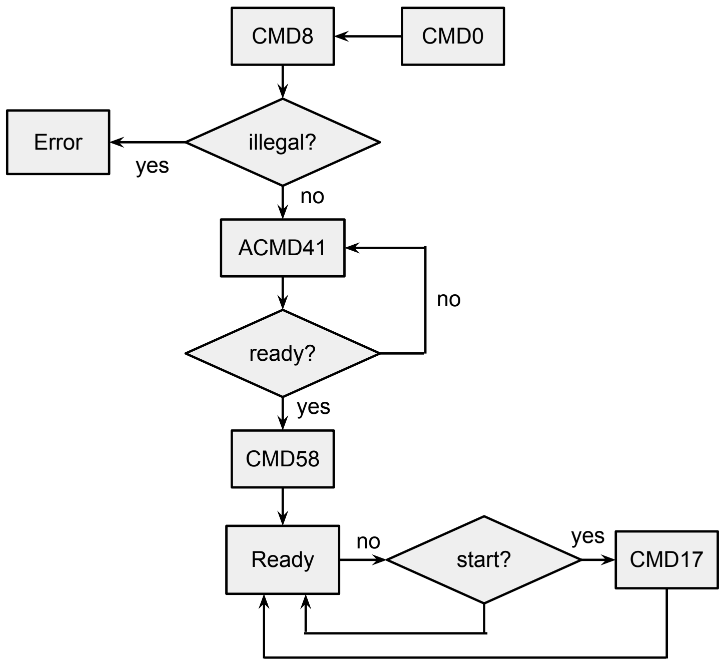

Figure 2: Flowchart of the commands sent by the `SD Card Controller`.

SD Card Data Formatting

Support for standard filesystems is not implemented. However, there is support for multiple song files. SD cards are addressed as 512-byte blocks. As such, the first 512 bytes of the SD card is dedicated to metadata; it contains the start addresses for each song on the SD card.

The songs themselves are in 16-bit raw PCM format, written to the SD card end to end and aligned to 512 byte boundaries. The start addresses of each song, as discussed earlier, are contained in the metadata block. A script was developed, `mkauimg.py`, for converting audio files to the appropriate format to be written to an SD card and is included in the appendix.

3.2 Audio Buffer

The audio buffer controls playback of the audio data. It contains a RAM holding 512 16-bit values and a unit that speaks to the Wolfson WM8731 audio codec on the DE2 board.

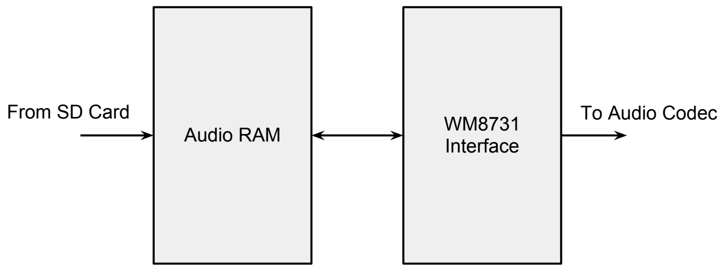

Figure 3: High-level block diagram of the `Audio Buffer` unit.

The audio codec interface reads in a 16-bit sample from the audio RAM and transmits the bits serially to the audio codec. Once the last bit has been transmitted, the next sample is requested from the audio RAM.

The audio RAM is effectively split in two so that the audio codec interface plays samples from one half while the SD card writes to the other. This is the reason why the RAM is chosen to hold 512 samples, since one SD card block is 512 bytes, or 256 samples. Once the audio codec interface plays sample 255 or 511 (indexed from 0), it sends a signal indicating that the SD card controller should read in another block.

3.3 FFT Unit

The `FFT` unit is used to compute the discrete frequency transform of a set of audio samples to be visualized.

We use the basic Cooley-Tukey FFT algorithm with a radix of 16 to compute the frequency transform of a given sample. The number of frequencies N computed by the FFT was chosen to be 256. The radix size and number of frequencies were chosen to optimize for space and time. The basic DFT is defined by the equation:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi j}{N} nk} \tag{1}$$

The index k is an integer from 0 to $N-1$. Thus, the result of the transform is a sequence of N complex numbers.

According to the Cooley-Tukey algorithm, we split our original input into 16 different parts and perform a DFT on each individual component. We can then recombine the individual DFT outputs in 4 recombination stages, using the following equation for each stage:

$$X_k = \begin{cases} E_k + e^{-\frac{2\pi j}{N} k} O_k & \text{if } k < N/2 \\ E_{k-N/2} - e^{-\frac{2\pi j}{N}(k-N/2)} O_{k-N/2} & \text{if } k \geq N/2. \end{cases} \tag{2}$$

The FFT hardware consists of two types of pipelines, one for the DFT, and another for the recombination.

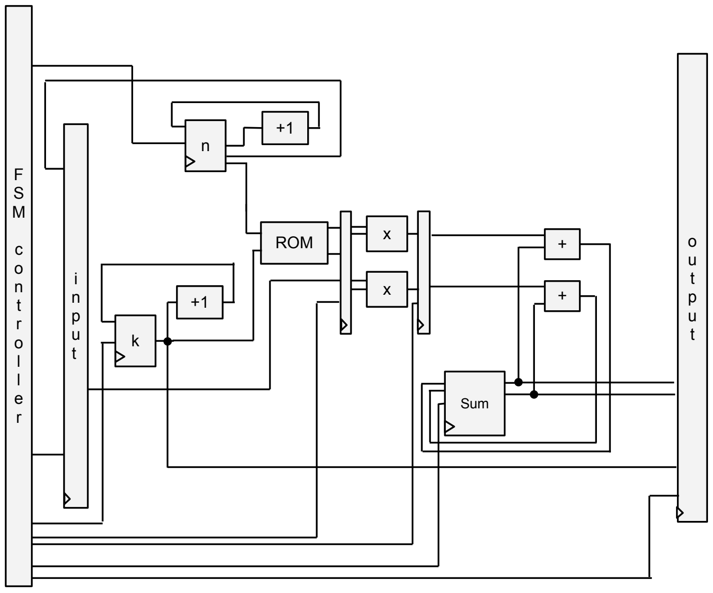

Figure 4: Block diagram of the `DFT Unit` pipeline

The DFT pipeline computes a 16-point DFT according to equation 1.

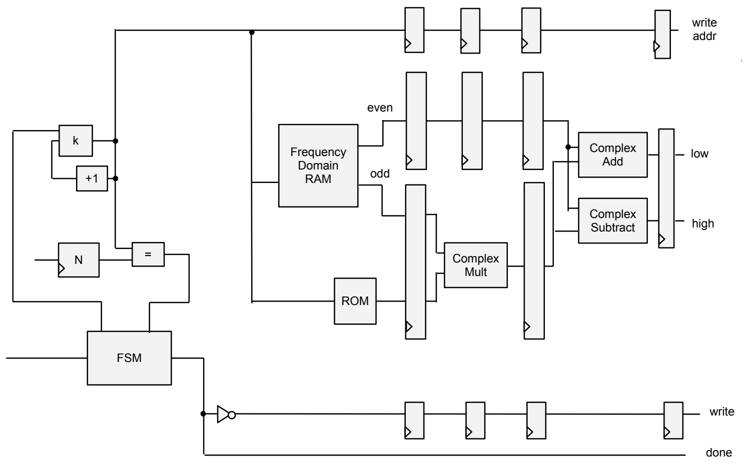

Figure 5: Block diagram of the texttttRecombination Unit pipeline

The recombination unit computes 32 parts of the recombinational step according to equation 2. The upper and lower parts, X_k and $X_{k+N/2}$, are computed in parallel. This allows us to re-use the odd term $e^{-\frac{2\pi j}{N}k}O_k$.

The complex multiplier in the recombination unit performs its computation in two pipelined steps, as follows.

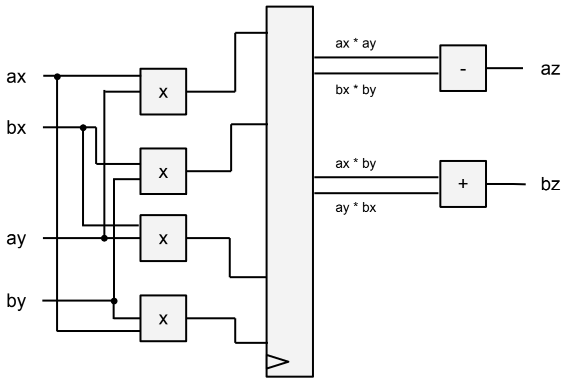

Figure 6: Block diagram of the `Complex Multiplier` unit used for recombination.

Our top-level FFT block uses two DFT units, a recombination unit, two RAMs (one for time domain data and one for frequency domain data), four ROMs for the recombination, one ROM for the DFT coefficients, and a control unit to set all the multiplexers and control the flow of computation.

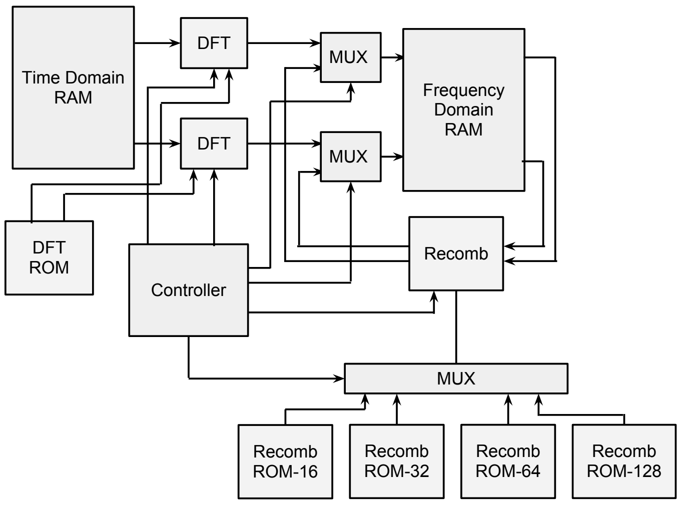

Figure 7: Top-level block diagram of the `FFT` unit.

The DFT ROM holds 256 32-bit values, each one of which represents a complex number (higher 16 bits for real part and lower 16 bits for imaginary part). These values represent the constant coefficients in the DFT equation $e^{-\frac{2\pi j}{N}nk}$ as 16-bit fixed-point precision numbers. Each value is addressed by an 8-bit address, where the highest four bits represent the value of k and the lowest four bits represent the value n.

Similarly, the 4 recombination ROMs have 16, 32, 64, or 128 fixed point imaginary values. These correspond to the constant coefficients $e^{-\frac{2\pi j}{N}k}$ from equation 2. The ROMs have 4-bit addresses, so only 16 values can be accessed during a single recombination step. For the larger ROMs, a select input set by the controller control which chunk of 16 values can be addressed.

The controller, in response to an external start signal, triggers 16 DFT computations, with 2 computations running in parallel at a time. This is followed by 4 stages of recombination. Each recombination stage uses a different ROM and consists of 8 steps (32 out of 256 outputs are computed on each step).

The multipliers used in this design all use the dedicated multiplier circuitry on the Cyclone II. All RAMs and ROMs use the dual-port M4K block RAM.

3.4 Visualizer

There are two main tasks that the vizualizer needs to accomplish. The first is sequentially reading in the data produced by the FFT and the other is displaying that data on the VGA. Originally all 256 different frequencies were being displayed however after initial designs the decision made was to include data for the first 32 frequencies on the display since these are the hearable frequencies. The reading process requires two states, a holding state and a reading state. The transition to reading happens when the FFT sends a "done" signal which means that the data is in place to be read. For display purposes, the 32 frequencies are placed into 16 bins, 2 per bin, each of which corresponds to one of sixteen bars located horizontally across the screen. The height of these bars is decided by summing the amplitude of the two frequencies contained in the bin and then scaling this value to the necessary height for the screen.

It was necessary to use two different clocks since the VGA display requires a 25.1 MHz clock. In this case, a 25 MHz clock as used. There was no need to read in data on the slower clock and therefore the 50 MHz clock was used there so as to read the data as quickly as possible.

An additional feature that was added was the ability to change the color of the bars appearing on the screen. Three switches corresponding to red, green, and blue allow the user to mix and match to create different colors. The switches are active low so the default color when all switches are "off" is white, so as to be seen on the black background. In order to improve the accuracy of the display, adjustments were made so that new data is only read in when data is not being drawn to the screen.

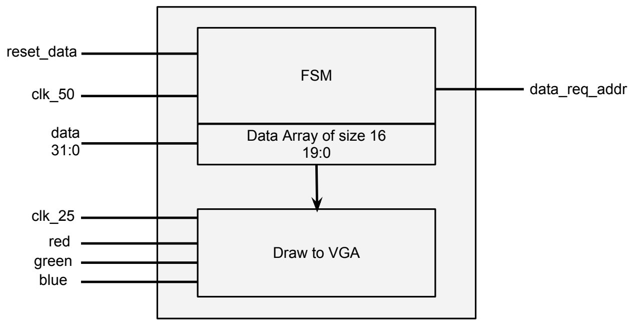

Figure 8: Block diagram of `Visualizer` unit.

3.5 Software User Interface

This project was initially designed using only hardware components. The NIOS entity was added later in order to give the system additional functionality. At startup, the software reads in the first block of data from the SD card. This first block is divided into 64 byte chunks. The first four bytes in each chunk is a 32-bit number which is the starting block address of that song. The next 60 bytes are a null-terminated string with the track title. The very last 64 byte chunk simply has the block address of the end of the last song. If there are fewer than 7 songs on the SD card, the remaining bytes in the block will all be zero. Once initialization is finished, the NIOS software hands control over the system to the hardware conductor. It then goes into a loop, during which it polls the current block address and the state of the FPGA push buttons. If the current block address goes over the first block address of the next track, the track number is updated internally as well as written to a set of seven-segment displays. The software also controls a frame-buffer used to fill in the top part of the VGA display. It uses this frame-buffer, along with the PS/2 keyboard, to provide a track-selection user interface display. This interface shows the different track titles and allows you to select from them using the keyboard. Keyboard-controlled operations are navigating through available songs using the `j` and `k` keys in conjunction with the `Enter` key, skipping forward and backwards through the available tracks using `n` and `p`, fast-forward and rewinding using `f` and `b`, and pausing and resuming playback using the spacebar. If a new track is selected using the keyboard controls, the program takes control away from the conductor, changes the block address to the start of the selected track, reads in that block, and then returns control to the conductor.

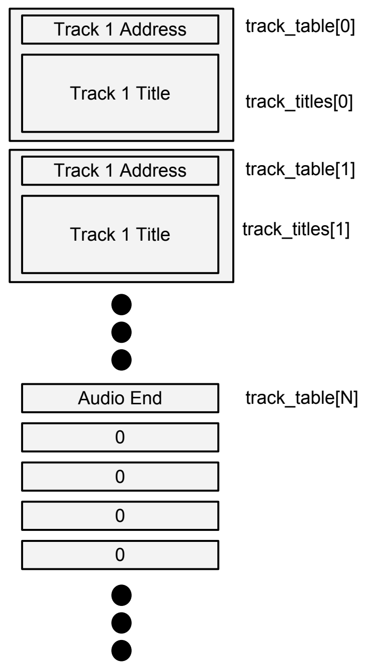

Figure 9: Visual representation of the track table used by the `NIOS` component.

3.6 Miscellaneous Controller Components

The Conductor

The conductor unit handles system coordination and communication between modules, and is the primary controller unit for the system. Typical system operation and data flow within the module is as follows:

- `initial` — When the system starts up, the conductor begins in the `initial` state. It waits for the SD card to finish intialization (sd_ready is high) and then hands off control to the NIOS system for system initialization and playback control in the `cpuctrl` state.
- `cpuctrl` — In this state, the NIOS system is handling user interaction. If the signal `nios_readblock` is high, the conductor will trigger read of a single block of data from the SD card by entering the trigger_sd state. Otherwise, if the `nios_play` signal is high, the system can begin or resume audio playback by transistioning into the resume state. In this condition, the block address to be read from is incremented.
- `trigger_sd` — This state is a transitional state to set sd_start high for one clock cycle. The conductor immediately transistions from this state to wait_sd
- `wait_sd` — In this state, the module waits until the `sd_ready` flag is set high, indicating that a block has been read and control can be returned to the CPU.
- `resume` — This state resets the `fft_counter` signal to 0 before audio playback begins.
- `playing` — This state is the default state when playback is occurring. Depending on the status signals, the conductor could switch into block_end or fft_end. If nios_play is set low, the conductor switches to the `initial` state to return the SD card to CPU control.
- `block_end` — The conductor enters this state for one clock cycle after the audio buffer switches buffers. It triggers a read from the SD card and, on every fourth block, triggers the FFT. The conductor also increments the SD card block address.

- `fft_end` — The conductor enters this state when the FFT unit finishes computing (fft_done changes from low to high). In this state, the visualizer reset is triggered.

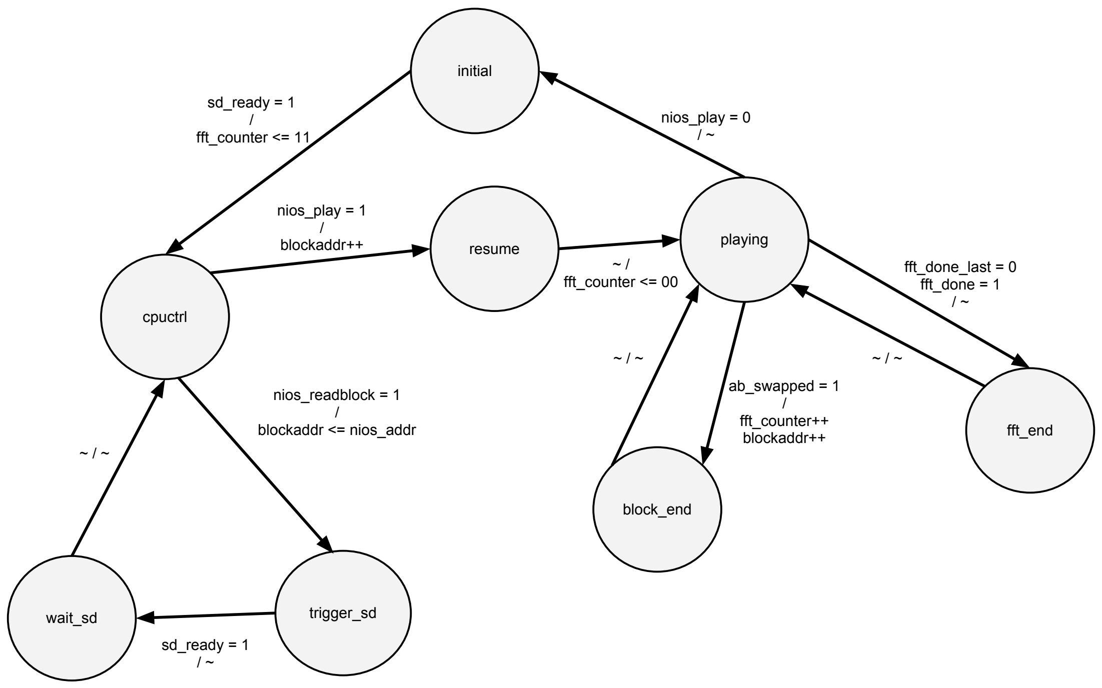
initial
sd_ready = 1 / fft_counter <= 11
nios_play = 0 / ~
nios_play = 1 / blockaddr++
resume
playing
~ / fft_counter <= 00
fft_done_last = 0 fft_done = 1 / ~
cpuctrl
~ / ~
~ / ~
nios_readblock = 1 / blockaddr <= nios_addr
ab_swapped = 1 / fft_counter++ blockaddr++
fft_end
~ / ~
block_end
wait_sd
trigger_sd
sd_ready = 1 / ~

Figure 10: Simplified State Diagram for the `Conductor` unit: Mealy machine, using abstract transition descriptions and omitting unused signals for compactness. ~denotes a lack of outputs

Phase-Locked Loop for Multiple Clocks

The visualizer unit and audio playback required different clocks to drive their respective peripherals, but also required clocks to synchronize communication with other modules in the kanto system. To most easily configure these clocks, a Phase-Locked Loop (PLL) was generated from an Altera Megafunction. The PLL was used to generate a 50MHz clock for general system synchronization, a 25MHz clock to drive the VGA display, and an 11.29MHz clock for the audio output.

4 Design Changes

4.1 Removal of SRAM

In our initial design, we planned on using the SRAM to store the values for the audio buffer and the output of the FFT unit. Separate components would communicate by reading and writing values to the SRAM. We had implemented an SRAM controller with four-phase handshaking to multiplex among the different components. We eventually found, however, that our memory usage did not justify the extra complexity that using SRAM imposed on our design, so we ended up using on-chip block RAM instead.

4.2 Adding Software Control

Initially, our design was implemented entirely in hardware, as we found our control scheme wasn't complex enough to require software. However, in order to satisfy the requirements for the class, we had to add a software component. We decided to use software to coordinate the initialization of our system, as well as track switching, and have the hardware controller (the conductor) take over when the audio is actually playing.

4.3 Display Changes

After connecting the FFT and visualizer units together, we determined the visualizer would be more aesthetically pleasing if we isolated and displayed only the frequencies detectable to the human ear, in this case the first 32 frequencies of the 256 calculated by the FFT. Another aesthetic change made was to provide the user with a color "channel mixer" via switches on the FPGA board, so the user can interactively adjust the color of the visualization.

5 Testing Methodology

Testing proved to be invaluable in isolating and fixing bugs before integrating modules into the system. For computational units and controllers, testing was done through simulations and assertions; for peripherals, testing was done by using test drivers and observing the peripherals behavior. The following sections provide a brief overview of our testing procedures for major components.

5.1 FFT Controller

The FFT algorithm was implemented in C early on. A sample input was run through the software FFT to generate a corresponding output, and we tested by comparing this output to the output generated in hardware given the same input. As we modified the FFT (for instance, to reduce parallelism and use block ram), we continually tested against this sample to make sure the hardware was still mathematically correct.

5.2 Visualizer

The two major tests of correctness for the visualizer were 1) displaying each of the bars at the correct height; and 2) quickly refreshing the screen when the registers were updated. To test this, we wrote a hardware driver that set the bars at two different levels and flipped back and forth between them while we observed the changes on the VGA monitor.

5.3 Audio Buffer

The audio buffer needed to produce a clean sound with no crackling. We tested this by writing in a sine wave and listening to the output. Additionally, a simulation test-bench was used to ensure that swapping between the two buffers was properly synchronized.

5.4 SD Card Controller

We tested the SD card controller by writing known data onto an SD card and ensuring that the same data was read back. Because there are numerous intermediate steps in communicating with an SD card, various LEDs and 7-segment displays indicate the current state of the state machine, allowing us to see where commands have failed.

5.5 Conductor

The conductor is the glue that holds all the components together. We emulated the system components in a test-bench and simulated the signals they would output. Using assertions, we confirmed that the conductor was correctly timing the interactions between components.

6 Timeline & Milestone Progress

Milestone	Date	Goal	Accomplished
Milestone 1	Apr 2	RTL design and block diagrams of all peripherals.	*Completed.*
Milestone 2	Apr 16	Individual peripherals written in VHDL and test benched.	*Completed.*
Milestone 3	Apr 30	Build interfaces between all peripherals and finish synchronization software.	*Completed*
Deadline	May 15	System complete and presentation finished.	*Completed.*

7 Distribution of Work

- **Kavita Jain-Cocks**
  1. Visualizer
  2. SD Controller
- **Zhehao Mao**
  1. FFT
  2. Audio Buffer
  3. SD Controller
  4. Conductor
  5. Software
- **Amrita Mazumdar**
  1. Visualizer
  2. FFT
  3. Software
- **Darien Nurse**
  1. Visualizer
  2. Software
- **Jonathan Yu**
  1. SD Controller
  2. Software

8 Challenges & Lessons Learned

8.1 Interfacing to External Hardware

Interfacing to external hardware, such as the SD card, audio codec, and VGA display, was by far the most challenging parts of the assignment, mainly because it was difficult to debug. Since proper functionality depended on what the external hardware was doing, we couldn't reliably use Quartus test-benches like we did for the rest of the components, since our test-benches couldn't exactly replicate the behavior of the external hardware. We found that the best way to get interfaces to external hardware working was to base our implementation off of successful existing implementations. For instance, for the audio buffer, we used the components provided for Lab 3 to interface with the audio codec, but modified the files to achieve the correct sampling frequency. For the SD card controller, we based a lot of our code off of the SPI controller from Prof. Edward's Apple II FPGA project, as well as the SD card driver code in the Linux Kernel. For the visualizer, we initially based our work on components from Lab 3, but quickly found we needed more advanced functionality than the examples provided. For debugging the visualizer, we resorted to driving the LEDs and 7-segment displays for feedback on states and data.

8.2 Changing Levels of Parallelism

Because of the extensive parallelism in the FFT module, it became difficult to fit the entire system on the FPGA, and we had to reduce the amount of parallelism in the FFT unit to decrease the number of logic gates used by the board. We found that reducing the level of parallelism is often just as hard, if not harder, than adding parallelism, and that both actions require extensive changes to the design of a system. It would have made things much easier if we had decided on the proper level of parallelism at the outset. If we had done some quick calculations at the outset, we would have discovered that could have used a very low level of parallelism and still been well within the deadlines imposed by the timing of our system.

8.3 Timing Issues

Timing issues can be a big problem in a complex design, especially if external hardware is involved, as different hardware peripherals require different clocks. For our design, the audio codec used a 11.29 MHz clock, the VGA used a 25 MHz clock, and the SD card used a 6.25 MHz clock. We used a PLL for the first two and clock enables for the last one (since the SD card clock wasn' always running). But passing status and control signals between units running on different clocks ended up being a minor challenge as well. We ended up solving this problem by either stretching the signals or detecting the rising edge of signals.

Because the operations of reading in VGA data and refreshing the VGA display both took more than one cycle meant that we needed to ensure that the data wasn't being changed while the display was in the middle of refreshing. This was accomplished by reading data into a separate set of registers and only transferring it to the registers used by the display processes once the refresh had reached the edge of the screen. This made it possible to reduce flickering that was being seen on the display.

8.4 Reduction of Hardware Usage

At its peak, our design used up 89% of the logic elements on the FPGA. We found out that this was because we were accidentally using logic elements instead of block RAM for all of our memory components. After switching over the memory components to block RAM, our logic element usage went down to only 10%, while our block RAM usage went from 1% to only 5%. This showed us the importance of actually using memory elements for memory.

Shifting our memory components to block RAM did require a lot of other changes to our design. For instance, since the block RAM only had two ports, we had to dramatically reduce the amount of parallelism in the FFT unit. We also had to duplicate some memory. For instance, the FFT unit and audio buffer initially shared the same memory for the time domain samples, but we eventually separated this into two different RAMs which the SD card controller wrote to simultaneously. Although these practices seemed counterintuitive, this duplication did not add that much to our memory requirements, as noted above.

Having a modular design, both at the highest level and at the level of each component, made these changes go relatively smoothly.

9 Reflections & Prospectus

Our group is very satisfied with the final version of our project. We accomplished all major project goals, and added improved functionality in certain modules, such as color changes in the visualizer and track selection. Although certain elements did not conform to the originally-delineated system design, such as replacing the SRAM with block RAM and changing the role of software in the high-level design, we feel the project exceeds our original goals for the project. There were a few areas under which our group could have improved, namely in scheduling and deliverables, and further design optimizations.

9.1 Scheduling & Deliverables

Because of the modular nature of our system design, our plan for project completion involved developing and optimizing each component fully in isolation, and then linking them together afterwards. Although this plan was successful for our group, a more productive strategy could have been a more sequential approach to system design, starting with critical components, such as the SD card, and moving linearly through the different components, ending with the visualizer. This method would have allowed us to have more concrete deliverables during milestones and helped us detect system-wide issues earlier on, such as the heavy use of logical elements in our memory and FFT designs.

Another consequence of our scheduling was that, although we remained fairly on track with our predetermined milestones and ended up completing the design ahead of schedule, our description of milestones and deliverables was extremely vague and hard to verify during the biweekly milestone check-ins. This could have been remedied in multiple ways. One option would have been to choose more specific, tangible deliverables at each milestone to demonstrate our progress. Another would have been to review our milestones with Professor Edwards or our adviser before deciding upon them.

One situation that needed to be contended with was that we had a larger group size to begin with and also lost a group member early in the process. The large group size was challenging because we needed to ensure the project goals were split evenly and progress was continuously being made, while grappling with the schedules of many different team members. Losing a group member only compounded the problem, because we had to reorganize some of the work distribution early on in the initial design phase. To address this, our team worked hard to maintain continued lines of communication via e-mail checkups and twice-weekly meetings. We felt that by staying organized and updating the entire group regularly on our progress, we were able to stay informed about the status of the project and what our immediate next tasks were, as well as reorganize tasks to compensate for the lost group member. Some invaluable tools used for group management were `Google Groups` and `Google Drive` for group email and documentation organization, and `Github Organizations` to manage our repositories of project code.

9.2 Optimizations & Improvements

Although all the initial project goals have been accomplished, there are always more improvements and new features that could be implemented:

1. The **visualization could be made more complex** than simply displaying bars for each frequency. Having a more interesting visualization would vastly improve the ”coolness” factor of our project.

2. It would be convenient if we changed the SD card format from our custom format to a **standard FAT filesystem**. This would make it easier for the user to add songs to the SD card, as they could simply copy audio files to the filesystem instead of having to use custom tools for converting the data and writing it to the block device.

3. A few areas of our code still contain sub-optimal hardware implementations (for instance, long if-else statements which may have been better written as state diagrams). Cleaning up our hardware implementation for a **more optimized use of logical elements** is critical for future work with this system.

9.3 Acknowledgements

The team would like to thank Professor Stephen Edwards and our TA Stephen Pratt for all their guidance and support in our Embedded Systems journey. Additional thanks go out to Luis Peña, for being cool, man.

A Source Code

A.1 VHDL

kanto.vhd

```vhdl
--
-- DE2 top-level module that includes the simple audio component
--
-- Stephen A. Edwards, Columbia University, sedwards@cs.columbia.edu
--
-- From an original by Terasic Technology, Inc.
-- (DE2_TOP.v, part of the DE2 system board CD supplied by Altera)
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity kanto is
    port(
        -- Clocks

        CLOCK_27,                                      -- 27 MHz
        CLOCK_50,                                      -- 50 MHz
        EXT_CLOCK : in std_logic;                      -- External Clock

        -- Buttons and switches

        KEY : in std_logic_vector(3 downto 0);         -- Push buttons
        SW : in std_logic_vector(17 downto 0);         -- DPDT switches

        -- LED displays

        HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment displays
             : out std_logic_vector(6 downto 0);
        LEDG : out std_logic_vector(7 downto 0);       -- Green LEDs
        LEDR : out std_logic_vector(17 downto 0);      -- Red LEDs

        -- RS-232 interface

        UART_TXD : out std_logic;                      -- UART transmitter
        UART_RXD : in std_logic;                       -- UART receiver

        -- IRDA interface

--        IRDA_TXD : out std_logic;                    -- IRDA Transmitter
        IRDA_RXD : in std_logic;                       -- IRDA Receiver

        -- SDRAM

        DRAM_DQ : inout std_logic_vector(15 downto 0); -- Data Bus
        DRAM_ADDR : out std_logic_vector(11 downto 0); -- Address Bus
        DRAM_LDQM,                                     -- Low-byte Data Mask
        DRAM_UDQM,                                     -- High-byte Data Mask
        DRAM_WE_N,                                     -- Write Enable
        DRAM_CAS_N,                                    -- Column Address Strobe
        DRAM_RAS_N,                                    -- Row Address Strobe
        DRAM_CS_N,                                     -- Chip Select
        DRAM_BA_0,                                     -- Bank Address 0
        DRAM_BA_1,                                     -- Bank Address 0
        DRAM_CLK,                                      -- Clock
        DRAM_CKE : out std_logic;                      -- Clock Enable

        -- FLASH

        FL_DQ : inout std_logic_vector(7 downto 0);    -- Data bus
```

```
FL_ADDR : out std_logic_vector(21 downto 0);    -- Address bus
FL_WE_N,                                        -- Write Enable
FL_RST_N,                                       -- Reset
FL_OE_N,                                        -- Output Enable
FL_CE_N : out std_logic;                        -- Chip Enable

-- SRAM

SRAM_DQ : inout std_logic_vector(15 downto 0); -- Data bus 16 Bits
SRAM_ADDR : out std_logic_vector(17 downto 0); -- Address bus 18 Bits
SRAM_UB_N,                                      -- High-byte Data Mask
SRAM_LB_N,                                      -- Low-byte Data Mask
SRAM_WE_N,                                      -- Write Enable
SRAM_CE_N,                                      -- Chip Enable
SRAM_OE_N : out std_logic;                      -- Output Enable

-- USB controller

OTG_DATA : inout std_logic_vector(15 downto 0); -- Data bus
OTG_ADDR : out std_logic_vector(1 downto 0);     -- Address
OTG_CS_N,                                        -- Chip Select
OTG_RD_N,                                        -- Write
OTG_WR_N,                                        -- Read
OTG_RST_N,                                       -- Reset
OTG_FSPEED,                                      -- USB Full Speed, 0 = Enable, Z =
    Disable
OTG_LSPEED : out std_logic;                      -- USB Low Speed, 0 = Enable, Z =
    Disable
OTG_INT0,                                        -- Intfaultupt 0
OTG_INT1,                                        -- Intfaultupt 1
OTG_DREQ0,                                       -- DMA Request 0
OTG_DREQ1 : in std_logic;                        -- DMA Request 1
OTG_DACK0_N,                                     -- DMA Acknowledge 0
OTG_DACK1_N : out std_logic;                     -- DMA Acknowledge 1

-- 16 X 2 LCD Module

LCD_ON,                                          -- Power ON/OFF
LCD_BLON,                                        -- Back Light ON/OFF
LCD_RW,                                          -- Read/Write Select, 0 = Write, 1 =
    Read
LCD_EN,                                          -- Enable
LCD_RS : out std_logic;                          -- Command/Data Select, 0 = Command, 1
     = Data
LCD_DATA : inout std_logic_vector(7 downto 0);  -- Data bus 8 bits

-- SD card interface

SD_DAT,                                          -- SD Card Data
SD_DAT3,                                         -- SD Card Data 3
SD_CMD : inout std_logic;                        -- SD Card Command Signal
SD_CLK : out std_logic;                          -- SD Card Clock

-- USB JTAG link

TDI,                                             -- CPLD -> FPGA (data in)
TCK,                                             -- CPLD -> FPGA (clk)
TCS : in std_logic;                              -- CPLD -> FPGA (CS)
TDO : out std_logic;                             -- FPGA -> CPLD (data out)

-- I2C bus

I2C_SDAT : inout std_logic; -- I2C Data
I2C_SCLK : out std_logic;   -- I2C Clock

-- PS/2 port

PS2_DAT,                        -- Data
```

```
            PS2_CLK : in std_logic;         -- Clock

            -- VGA output

            VGA_CLK,                                        -- Clock
            VGA_HS,                                         -- H_SYNC
            VGA_VS,                                         -- V_SYNC
            VGA_BLANK,                                      -- BLANK
            VGA_SYNC : out std_logic;                       -- SYNC
            VGA_R,                                          -- Red[9:0]
            VGA_G,                                          -- Green[9:0]
            VGA_B : out std_logic_vector(9 downto 0);       -- Blue[9:0]

            --     Ethernet Interface

            ENET_DATA : inout std_logic_vector(15 downto 0);        -- DATA bus 16Bits
            ENET_CMD,                     -- Command/Data Select, 0 = Command, 1 = Data
            ENET_CS_N,                                              -- Chip Select
            ENET_WR_N,                                              -- Write
            ENET_RD_N,                                              -- Read
            ENET_RST_N,                                             -- Reset
            ENET_CLK : out std_logic;                               -- Clock 25 MHz
            ENET_INT : in std_logic;                                -- Intfaultupt

            -- Audio CODEC

            AUD_ADCLRCK : inout std_logic;                         -- ADC LR Clock
            AUD_ADCDAT : in std_logic;                             -- ADC Data
            AUD_DACLRCK : inout std_logic;                         -- DAC LR Clock
            AUD_DACDAT : out std_logic;                            -- DAC Data
            AUD_BCLK : inout std_logic;                            -- Bit-Stream Clock
            AUD_XCK : out std_logic;                               -- Chip Clock

            -- Video Decoder

            TD_DATA : in std_logic_vector(7 downto 0);     -- Data bus 8 bits
            TD_HS,                                         -- H_SYNC
            TD_VS : in std_logic;                          -- V_SYNC
            TD_RESET : out std_logic;                      -- Reset

            -- General-purpose I/O

            GPIO_0,                                       -- GPIO Connection 0
            GPIO_1 : inout std_logic_vector(35 downto 0) -- GPIO Connection 1
            );

    end kanto;

    architecture datapath of kanto is
        signal ab_play : std_logic;
        signal ab_audio_ok : std_logic;
        signal ab_swapped : std_logic;
        signal ab_force_swap : std_logic;

      signal fft_req : std_logic;
      signal fft_ack : std_logic;
      signal fft_addr : std_logic_vector(17 downto 0);
      signal fft_readdata : std_logic_vector(15 downto 0);
      signal fft_start : std_logic;
        signal fft_allow_write : std_logic;
        signal fft_tdom_write : std_logic;
        signal fft_fdom_addr : unsigned(7 downto 0);
        signal fft_fdom_data : signed(31 downto 0);
      signal fft_done      : std_logic;

      signal main_clk : std_logic;
      signal aud_clk : std_logic;
      signal start : std_logic;
```

```
  -- inserted for SDC testing
  signal sd_start : std_logic;
  signal sd_ready : std_logic;
    signal sd_err : std_logic;
    signal sd_waiting : std_logic;
    signal sd_resp_debug : std_logic_vector(7 downto 0);
    signal sd_state_debug : std_logic_vector(7 downto 0);
    signal sd_ccs : std_logic;
    signal sd_writedata : signed(15 downto 0);
    signal sd_writeaddr : unsigned(7 downto 0);
    signal sd_blockaddr : unsigned(31 downto 0);
    signal sd_write_en : std_logic;

    signal clk25 : std_logic := '0';
    signal viz_reset : std_logic;
    signal cond_err : std_logic;
    signal audio_track : std_logic_vector(7 downto 0);

    -- control and status signals for NIOS
    signal nios_addr : unsigned(31 downto 0);
    signal nios_readblock : std_logic;
    signal nios_play : std_logic;
    signal nios_done : std_logic;

    signal sdbuf_rden : std_logic;
    signal sdbuf_addr : std_logic_vector(7 downto 0);
    signal sdbuf_data : std_logic_vector(15 downto 0);

    signal vga_display_x : std_logic_vector(9 downto 0);
    signal vga_display_y : std_logic_vector(6 downto 0);
    signal vga_display_pixel_on : std_logic;

    component de2_i2c_av_config is
        port (iclk : in std_logic;
              irst_n : in std_logic;
              i2c_sclk : out std_logic;
              i2c_sdat : inout std_logic);
    end component;
begin

    LEDG(0) <= sd_ready;
    LEDG(1) <= ab_audio_ok;
    LEDG(2) <= ab_play;
    LEDR(0) <= sd_err;
    LEDR(1) <= cond_err;
    ab_play <= SW(17) and ab_audio_ok;

    NIOS : entity work.nios_system port map (
        clk_0 => main_clk,
        reset_n => '1',

        SRAM_ADDR_from_the_sram => SRAM_ADDR,
        SRAM_CE_N_from_the_sram => SRAM_CE_N,
        SRAM_DQ_to_and_from_the_sram => SRAM_DQ,
        SRAM_LB_N_from_the_sram => SRAM_LB_N,
        SRAM_OE_N_from_the_sram => SRAM_OE_N,
        SRAM_UB_N_from_the_sram => SRAM_UB_N,
        SRAM_WE_N_from_the_sram => SRAM_WE_N,

        unsigned(nios_addr_from_the_kanto_ctrl) => nios_addr,
        nios_done_to_the_kanto_ctrl => nios_done,
        nios_play_from_the_kanto_ctrl => nios_play,
        nios_readblock_from_the_kanto_ctrl => nios_readblock,
        sd_blockaddr_to_the_kanto_ctrl => std_logic_vector(sd_blockaddr),
        audio_track_from_the_kanto_ctrl => audio_track,
        keys_to_the_kanto_ctrl => not KEY,
        sd_ccs_to_the_kanto_ctrl => sd_ccs,
        sdbuf_addr_from_the_sdbuf => sdbuf_addr,
        sdbuf_data_to_the_sdbuf => sdbuf_data,
        sdbuf_rden_from_the_sdbuf => sdbuf_rden,

        PS2_Clk_to_the_ps2 => PS2_CLK,
        PS2_Data_to_the_ps2 => PS2_DAT,

        display_x_to_the_vga => vga_display_x,
        display_y_to_the_vga => vga_display_y,
        display_pixel_on_from_the_vga => vga_display_pixel_on,
        display_clk_to_the_vga => clk25
    );

    PLL : entity work.audpll port map (
        inclk0 => CLOCK_50,
        c0 => aud_clk,
        c1 => main_clk,
        c2 => clk25
    );

    AUD_XCK <= aud_clk;

    I2C_CONF : de2_i2c_av_config port map (
        iclk => main_clk,
        irst_n => '1',
        i2c_sdat => i2c_sdat,
        i2c_sclk => i2c_sclk
    );

    CDTR : entity work.conductor port map (
        clk => main_clk,
        ab_audio_ok => ab_audio_ok,
        ab_swapped => ab_swapped,
        ab_force_swap => ab_force_swap,
        sd_start => sd_start,
        sd_ready => sd_ready,
        sd_ccs => sd_ccs,
        sd_addr => sd_blockaddr,
        fft_start => fft_start,
        fft_done => fft_done,
        fft_allow_write => fft_allow_write,
        cond_err => cond_err,
        viz_reset => viz_reset,
        nios_addr => nios_addr,
        nios_readblock => nios_readblock,
        nios_play => nios_play,
        nios_done => nios_done
    );

    AB : entity work.audio_buffer port map (
        clk => main_clk,
        aud_clk => aud_clk,
        play => ab_play,
        swapped => ab_swapped,
        force_swap => ab_force_swap,

        aud_adclrck => aud_adclrck,
        aud_adcdat => aud_adcdat,
        aud_daclrck => aud_daclrck,
        aud_dacdat => aud_dacdat,
        aud_bclk => aud_bclk,

        writeaddr => sd_writeaddr,
        writedata => sd_writedata,
        write_en => sd_write_en
    );
```

```
        SDC : entity work.sd_controller port map (
            clk50 => main_clk,

            cs   => SD_DAT3,
            mosi => SD_CMD,
            miso => SD_DAT,
            sclk => SD_CLK,

            start => sd_start,
            ready => sd_ready,
            err => sd_err,
            waiting => sd_waiting,
            ccs => sd_ccs,

            resp_debug => sd_resp_debug,
            state_debug => sd_state_debug,

            blockaddr => sd_blockaddr,
            writedata => sd_writedata,
            writeaddr => sd_writeaddr,
            write_en => sd_write_en
        );

        fft_tdom_write <= fft_allow_write and sd_write_en;

        FFT : entity work.fft_controller port map (
            clk => main_clk,
            start => fft_start,
            done => fft_done,

            tdom_addr_in => sd_writeaddr,
            tdom_data_in => sd_writedata,
            tdom_write => fft_tdom_write,

            fdom_addr_out => fft_fdom_addr,
            fdom_data_out => fft_fdom_data
        );

       VISUALIZER : entity work.visualizer port map(
        clk25 => clk25,
            clk50 => main_clk,
            reset_data      => viz_reset,
        fft_fdom_addr   => fft_fdom_addr,
        fft_fdom_data   => fft_fdom_data,
        VGA_CLK         => VGA_CLK,
        VGA_HS          => VGA_HS,
        VGA_VS          => VGA_VS,
        VGA_BLANK       => VGA_BLANK,
        VGA_SYNC    => VGA_SYNC,
        VGA_R         => VGA_R,
        VGA_G          => VGA_G,
        VGA_B       => VGA_B,
        ledr17        => LEDR(17),
        ledr16        => LEDR(16),
        ledr15        => LEDR(15),
            sw_r            => SW(2),
            sw_g            => SW(1),
            sw_b            => SW(0),

            vga_text_buffer_x => vga_display_x,
            vga_text_buffer_y => vga_display_y,
            vga_text_buffer_pixel => vga_display_pixel_on
       );

        SDBUF_RAM : entity work.tdom_full_ram port map (
            clock => main_clk,
            data => std_logic_vector(sd_writedata),
            wraddress => std_logic_vector(sd_writeaddr),
            wren => fft_tdom_write,
            rdaddress => sdbuf_addr,
            q => sdbuf_data,
            rden => sdbuf_rden
        );

        SS0 : entity work.sevenseg port map (
            number => sd_resp_debug(3 downto 0),
            display => HEX0
        );

        SS1 : entity work.sevenseg port map (
            number => sd_resp_debug(7 downto 4),
            display => HEX1
        );

        SS2 : entity work.sevenseg port map (
            number => sd_state_debug(3 downto 0),
            display => HEX2
        );

        SS3 : entity work.sevenseg port map (
            number => sd_state_debug(7 downto 4),
            display => HEX3
        );

        SS4 : entity work.sevenseg port map (
            number => audio_track(3 downto 0),
            display => HEX4
        );

        SS5 : entity work.sevenseg port map (
            number => audio_track(7 downto 4),
            display => HEX5
        );

        HEX7 <= (others => '1');
        HEX6 <= (others => '1');

        LEDG(7 downto 3) <= (others => '0');
        LEDR(13 downto 2) <= (others => '0');

        LCD_ON   <= '1';
        LCD_BLON <= '1';
        LCD_RW   <= '1';
        LCD_EN   <= '0';
        LCD_RS   <= '0';

        --SD_DAT3 <= '1';
        --SD_CMD <= '1';
        --SD_CLK <= '1';

        UART_TXD <= '0';
        DRAM_ADDR <= (others => '0');
        DRAM_LDQM <= '0';
        DRAM_UDQM <= '0';
        DRAM_WE_N <= '1';
        DRAM_CAS_N <= '1';
        DRAM_RAS_N <= '1';
        DRAM_CS_N <= '1';
        DRAM_BA_0 <= '0';
        DRAM_BA_1 <= '0';
        DRAM_CLK <= '0';
        DRAM_CKE <= '0';

    --    SRAM_DQ <= (others => 'Z');
    --    SRAM_ADDR <= (others => '0');
    --    SRAM_UB_N <= '1';
    --    SRAM_LB_N <= '1';
    --    SRAM_CE_N <= '1';
    --    SRAM_WE_N <= '1';
    --    SRAM_OE_N <= '1';

        FL_ADDR <= (others => '0');
        FL_WE_N <= '1';
        FL_RST_N <= '0';
        FL_OE_N <= '1';
        FL_CE_N <= '1';

        OTG_ADDR <= (others => '0');
        OTG_CS_N <= '1';
        OTG_RD_N <= '1';
        OTG_RD_N <= '1';
        OTG_WR_N <= '1';
        OTG_RST_N <= '1';
        OTG_FSPEED <= '1';
        OTG_LSPEED <= '1';
        OTG_DACK0_N <= '1';
        OTG_DACK1_N <= '1';

        TDO <= '0';

        ENET_CMD <= '0';
        ENET_CS_N <= '1';
        ENET_WR_N <= '1';
        ENET_RD_N <= '1';
        ENET_RST_N <= '1';
        ENET_CLK <= '0';

        TD_RESET <= '0';

        -- Set all bidirectional ports to tri-state
        DRAM_DQ         <= (others => 'Z');
        FL_DQ             <= (others => 'Z');
        OTG_DATA        <= (others => 'Z');
        LCD_DATA        <= (others => 'Z');
        SD_DAT            <= 'Z';
        ENET_DATA     <= (others => 'Z');
        GPIO_0            <= (others => 'Z');
        GPIO_1            <= (others => 'Z');

    end datapath;
```

audio_buffer.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity audio_buffer is
    port (clk : in std_logic;
          play : in std_logic;
          aud_clk : in std_logic;
          swapped : out std_logic;
          force_swap : in std_logic;

          aud_adcdat : in std_logic;
          aud_adclrck : inout std_logic;
          aud_daclrck : inout std_logic;
          aud_dacdat : out std_logic;
          aud_bclk : inout std_logic;

          writeaddr : in unsigned(7 downto 0);
          writedata : in signed(15 downto 0);
          write_en : in std_logic);
end audio_buffer;

architecture rtl of audio_buffer is
    signal audio_addr : unsigned(8 downto 0) := (others => '0');
    signal audio_data : signed(15 downto 0);
    signal audio_request : std_logic;
    signal last_audio_request : std_logic;

    signal wlr : std_logic := '0'; -- writes leading reads
    signal wfulladdr : unsigned(8 downto 0);
begin
    wfulladdr <= wlr & writeaddr;

    AUDIO_RAM : entity work.tdom_double_ram port map (
        clock => clk,
        rdaddress => std_logic_vector(audio_addr),
        signed(q) => audio_data,
        rden => play,
        wraddress => std_logic_vector(wfulladdr),
        data => std_logic_vector(writedata),
        wren => write_en
    );

    process (clk)
        variable counter_en : std_logic := '0';
    begin
        if rising_edge(clk) then
            swapped <= '0';
            last_audio_request <= audio_request;
            -- detect rising edge of audio_request
            counter_en := audio_request and (not last_audio_request);

            if counter_en = '1' then
                -- swap when audio address reaches 255 or 511
                if audio_addr(7 downto 0) = x"ff" then
                    wlr <= audio_addr(8);
                    swapped <= '1';
                end if;
                audio_addr <= audio_addr + 1;
            elsif force_swap = '1' then
                audio_addr <= wlr & x"00";
                wlr <= not wlr;
            end if;
        end if;
    end process;

    CODEC : entity work.de2_wm8731_audio port map (
        clk => aud_clk,
        reset_n => play,
        test_mode => '0',

        aud_adclrck => aud_adclrck,
        aud_adcdat => aud_adcdat,
        aud_daclrck => aud_daclrck,
        aud_dacdat => aud_dacdat,
        aud_bclk => aud_bclk,

        data => std_logic_vector(audio_data),
        audio_request => audio_request
    );
end rtl;
```

complex_mult.vhd

```
library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity complex_mult is
    port (clk : in std_logic;
          realx : in signed(15 downto 0);
          imagx : in signed(15 downto 0);
          realy : in signed(15 downto 0);
          imagy : in signed(15 downto 0);
          realz : out signed(31 downto 0);
          imagz : out signed(31 downto 0));
end complex_mult;

architecture rtl of complex_mult is
    signal xryr : signed(31 downto 0);
    signal xryi : signed(31 downto 0);
    signal xiyr : signed(31 downto 0);
    signal xiyi : signed(31 downto 0);
begin
    MXRYR : entity work.mult port map (
        dataa => std_logic_vector(realx),
        datab => std_logic_vector(realy),
        signed(result) => xryr
    );

    MXRYI : entity work.mult port map (
        dataa => std_logic_vector(realx),
        datab => std_logic_vector(imagy),
        signed(result) => xryi
    );

    MXIYR : entity work.mult port map (
        dataa => std_logic_vector(imagx),
        datab => std_logic_vector(realy),
        signed(result) => xiyr
    );

    MXIYI : entity work.mult port map (
        dataa => std_logic_vector(imagx),
        datab => std_logic_vector(imagy),
        signed(result) => xiyi
    );

    process (clk)
    begin
        if rising_edge(clk) then
            realz <= xryr - xiyi;
            imagz <= xryi + xiyr;
        end if;
    end process;
end rtl;
```

conductor.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity conductor is
    port (clk : in std_logic;
          ab_audio_ok : out std_logic;
          ab_swapped : in std_logic;
          ab_force_swap : out std_logic;
          fft_allow_write : out std_logic;

          sd_start : out std_logic;
          sd_ready : in std_logic;
          cond_err : out std_logic;
          sd_addr : out unsigned(31 downto 0);
          sd_ccs : in std_logic;

          nios_addr : in unsigned(31 downto 0);
          nios_readblock : in std_logic;
          nios_play : in std_logic;
          nios_done : out std_logic;

          fft_start : out std_logic;
          fft_done : in std_logic;
          viz_reset : out std_logic);
end conductor;

architecture rtl of conductor is
    type conductor_state is (initial, cpuctrl, trigger_sd, wait_sd,
                            resume, playing, fft_end, block_end);
    signal state : conductor_state := initial;
    signal fft_done_last : std_logic;
    signal fft_counter : unsigned(1 downto 0);
    signal blockaddr : unsigned(31 downto 0);
begin
    sd_addr <= blockaddr;

    process (clk)
    begin
        if rising_edge(clk) then
            fft_done_last <= fft_done;

            case state is
                when initial =>
                    if sd_ready = '1' then
                        -- once SD card is initialized,
                        -- let the CPU take control
                        state <= cpuctrl;
                        fft_counter <= "11";
                    end if;
                when cpuctrl =>
                    if nios_readblock = '1' then
                        state <= trigger_sd;
                        blockaddr <= nios_addr;
                    elsif nios_play = '1' then
                        state <= resume;
                        if sd_ccs = '1' then
                            blockaddr <= blockaddr + 1;
                        else
                            blockaddr <= blockaddr + 512;
                        end if;
                    end if;
                when trigger_sd =>
                    state <= wait_sd;
                when wait_sd =>
                    if sd_ready = '1' then
                        -- once block is read, return control to CPU
                        state <= cpuctrl;
                    end if;
                when resume =>
                    -- once swapped, we can allow audio to play
                    state <= playing;
                    fft_counter <= "00";
                when playing =>
                    if nios_play = '0' then
                        state <= initial;
                    elsif ab_swapped = '1' then
                        -- if we've outrun the SD card
                        -- indicate that an error has occurred
                        if sd_ready = '0' then
```

```
                            cond_err <= '1';
                        else
                            cond_err <= '0';
                        end if;
                        fft_counter <= fft_counter + 1;
                        state <= block_end;

                        if sd_ccs = '1' then
                            blockaddr <= blockaddr + 1;
                        else
                            blockaddr <= blockaddr + 512;
                        end if;
                    elsif fft_done_last = '0' and fft_done = '1' then
                        state <= fft_end;
                    end if;
                when fft_end =>
                    state <= playing;
                when block_end =>
                    -- once the audio buffer has switched blocks
                    -- tell SD card to read another block and,
                    -- on every fourth block, start FFT
                    state <= playing;
            end case;
        end if;
    end process;

    -- can play audio once initialization is done
    ab_audio_ok <= '1' when state = playing or state = block_end or
                        state = fft_end else '0';
    ab_force_swap <= '1' when state = resume else '0';
    sd_start <= '1' when state = trigger_sd or state = block_end or
                     state = resume else '0';
    -- only compute fft and refresh visualizer every fourth block
    fft_start <= '1' when state = block_end and fft_counter = 0 else '0';
    viz_reset <= '1' when state = fft_end else '0';

    -- only let sd card write to FFT unit the block before FFT is computed
    fft_allow_write <= '1' when fft_counter = "11" else '0';

    nios_done <= '1' when state = cpuctrl else '0';
end rtl;
```

de2_kanto_ctrl.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity de2_kanto_ctrl is
    port (clk : in std_logic;
          reset_n : in std_logic;
          read : in std_logic;
          write : in std_logic;
          chipselect : in std_logic;
          address : in std_logic_vector(2 downto 0);
          readdata : out std_logic_vector(31 downto 0);
          writedata : in std_logic_vector(31 downto 0);

          nios_addr : out std_logic_vector(31 downto 0);
          nios_readblock : out std_logic;
          nios_play : out std_logic;
          nios_done : in std_logic;

          keys : in std_logic_vector(3 downto 0);

          sd_ccs : in std_logic;
          audio_track : out std_logic_vector(7 downto 0);
          sd_blockaddr : in std_logic_vector(31 downto 0));
end de2_kanto_ctrl;

-- 0 is block address
-- 1 is readblock
-- 2 is play
-- 3 is done
-- 4 is audio_track
-- 5 is keys

architecture rtl of de2_kanto_ctrl is
begin
    process (clk)
    begin
        if rising_edge(clk) then
            if reset_n = '0' then
                nios_addr <= (others => '0');
                nios_play <= '0';
            elsif chipselect = '1' then
                case address is
                    when "000" =>
                        if read = '1' then
                            readdata <= sd_blockaddr;
                        elsif write = '1' then
                            nios_addr <= writedata;
                        end if;
                    when "001" =>
                        if write = '1' then
                            nios_readblock <= writedata(0);
                        end if;
                    when "010" =>
                        if write = '1' then
                            nios_play <= writedata(0);
                        end if;
                    when "011" =>
                        if read = '1' then
                            readdata <= (31 downto 1 => '0') & nios_done;
                        end if;
                    when "100" =>
                        if write = '1' then
                            audio_track <= writedata(7 downto 0);
                        end if;
                    when "101" =>
                        if read = '1' then
                            readdata <= (31 downto 4 => '0') & keys;
                        end if;
                    when "110" =>
                        if read = '1' then
                            readdata <= (31 downto 1 => '0') & sd_ccs;
                        end if;
                    when others =>
                        if read = '1' then
                            readdata <= (others => '0');
                        end if;
                end case;
            end if;
        end if;
    end process;
end rtl;
```

de2_sd_buffer.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity de2_sd_buffer is
```

```
      port (clk : in std_logic;
            reset_n : in std_logic;
            read : in std_logic;
            chipselect : in std_logic;
            address : in std_logic_vector(7 downto 0);
            readdata : out std_logic_vector(15 downto 0);

            sdbuf_rden : out std_logic;
            sdbuf_addr : out std_logic_vector(7 downto 0);
            sdbuf_data : in std_logic_vector(15 downto 0));
end de2_sd_buffer;

architecture rtl of de2_sd_buffer is
begin
    sdbuf_addr <= address;
    readdata <= sdbuf_data;
    sdbuf_rden <= read and chipselect and reset_n;
end rtl;
```

de2_sram_controller.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity de2_sram_controller is
port (
    signal chipselect : in std_logic;
    signal write, read : in std_logic;
    signal address : in std_logic_vector(17 downto 0);
    signal readdata : out std_logic_vector(15 downto 0);
    signal writedata : in std_logic_vector(15 downto 0);
    signal byteenable : in std_logic_vector(1 downto 0);
    signal SRAM_DQ : inout std_logic_vector(15 downto 0);
    signal SRAM_ADDR : out std_logic_vector(17 downto 0);
    signal SRAM_UB_N, SRAM_LB_N : out std_logic;
    signal SRAM_WE_N, SRAM_CE_N : out std_logic;
    signal SRAM_OE_N : out std_logic
);
end de2_sram_controller;

architecture dp of de2_sram_controller is
begin
    SRAM_DQ <= writedata when write = '1'
        else (others => 'Z');
    readdata <= SRAM_DQ;
    SRAM_ADDR <= address;
    SRAM_UB_N <= not byteenable(1);
    SRAM_LB_N <= not byteenable(0);
    SRAM_WE_N <= not write;
    SRAM_CE_N <= not chipselect;
    SRAM_OE_N <= not read;
end dp;
```

de2_vga_text_buffer.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_vga_text_buffer is
port(
    vga_clk : in std_logic;
    vga_reset : in std_logic;
    vga_read : in std_logic;
    vga_write : in std_logic;
    vga_chipselect : in std_logic;

    -- there are 80 x 5 = 400 characters
    -- 400 * 4 parts per character = 1600 parts
    -- so 11 bit addressing
    vga_address : in std_logic_vector(10 downto 0);

    -- each character part is 32 bits
    vga_readdata : out std_logic_vector(31 downto 0);
    vga_writedata : in std_logic_vector(31 downto 0);

    -- the following are accessed by a vga driver
    -- give me a pixel coordinate (on a 640x80 plane)
    -- and I'll tell you if it's on or off

    -- it's 40 pixels high because we have 5 lines,
    -- each 16 pixels high
    display_pixel_on : out std_logic;
    display_x : in std_logic_vector(9 downto 0);
    display_y : in std_logic_vector(6 downto 0);
    display_clk : in std_logic
);
end de2_vga_text_buffer;

architecture rtl of de2_vga_text_buffer is

    -- there are 16 lines in each character font.
    -- first line of ever character goes in one array, second line in
    -- another array, etc. -- this lets us write and read in parallel

    type addr_array is array(0 to 15) of std_logic_vector(8 downto 0);
    type data_array is array(0 to 15) of std_logic_vector(7 downto 0);
    type backwards_data_array is array(0 to 15) of std_logic_vector(0 to 7);

    signal row_writedata : data_array;
    signal row_write_en : std_logic_vector(0 to 15);
    signal row_readdata : backwards_data_array;

    signal x : integer range 0 to 399;
    signal y : integer range 0 to 15;
    signal inner_x : integer range 0 to 7;

begin

    RAM_GENERATE : for i in 0 to 15 generate
        RAM : entity work.vga_row_ram port map (
            rdaddress => std_logic_vector(to_unsigned(x, 9)),
            rdclock => display_clk,
            q => row_readdata(i),

            -- the top 9 bits tell us what character we are in
            -- the bottom 2 bits tell us which rows of the character
            -- we are in (since we have split each character into
            -- 4 groups of 4 rows)
            wraddress => vga_address(10 downto 2),
            -- so the bottom two bits tell us whether to enable the write
            -- for this row
            wren => row_write_en(i),
            wrclock => vga_clk,
            data => row_writedata(i)
        );
    end generate RAM_GENERATE;

    -- since we can only write one character part at a time, and
    -- each character is composed of four character parts, we look at the
    -- bottom two bits of the address to determine which lines this
    -- character part belongs on
    row_write_en(0) <= vga_write when vga_address(1 downto 0) = "00" else '0';
    row_write_en(1) <= vga_write when vga_address(1 downto 0) = "00" else '0';
    row_write_en(2) <= vga_write when vga_address(1 downto 0) = "00" else '0';
    row_write_en(3) <= vga_write when vga_address(1 downto 0) = "00" else '0';
    row_write_en(4) <= vga_write when vga_address(1 downto 0) = "01" else '0';
    row_write_en(5) <= vga_write when vga_address(1 downto 0) = "01" else '0';
    row_write_en(6) <= vga_write when vga_address(1 downto 0) = "01" else '0';
    row_write_en(7) <= vga_write when vga_address(1 downto 0) = "01" else '0';
    row_write_en(8) <= vga_write when vga_address(1 downto 0) = "10" else '0';
    row_write_en(9) <= vga_write when vga_address(1 downto 0) = "10" else '0';
    row_write_en(10) <= vga_write when vga_address(1 downto 0) = "10" else '0';
    row_write_en(11) <= vga_write when vga_address(1 downto 0) = "10" else '0';
    row_write_en(12) <= vga_write when vga_address(1 downto 0) = "11" else '0';
    row_write_en(13) <= vga_write when vga_address(1 downto 0) = "11" else '0';
    row_write_en(14) <= vga_write when vga_address(1 downto 0) = "11" else '0';
    row_write_en(15) <= vga_write when vga_address(1 downto 0) = "11" else '0';

    -- we have 8 bit wide data ports, so we split it into 4 parallel
    -- writes. we use the write enables to determine which actually
    -- get written to
    MAPPING2_GENERATE : for i in 0 to 3 generate
        row_writedata(i * 4 + 0) <= vga_writedata(31 downto 24);
        row_writedata(i * 4 + 1) <= vga_writedata(23 downto 16);
        row_writedata(i * 4 + 2) <= vga_writedata(15 downto 8);
        row_writedata(i * 4 + 3) <= vga_writedata(7 downto 0);
    end generate MAPPING2_GENERATE;


    -- there are 16 lines in each font character, so we can
    -- look at the bottom 4 bits of y to know which line we're on
    y <= to_integer(unsigned(display_y(3 downto 0)));

    -- each character is 8 pixels wide, so to find which character
    -- we're in, ignore the bottom 3 bits. But there's a catch - we
    -- store all the rows end to end, so we have to add y * row_length
    -- where row_length = 80

    -- we can implement this as y * 80 == y << 6 + y << 4
    x <= to_integer(unsigned(display_x(9 downto 3))
      + unsigned(unsigned(display_y(6 downto 4)) & "000000")
      + unsigned(unsigned(display_y(6 downto 4)) & "0000"));
--    x <= 0;

    -- we still need to find the x position within that particular char.
    -- this -2 is a horrible horrible hack and I have no idea why it works
    -- probably related to timing
    inner_x <= to_integer(unsigned(display_x(2 downto 0))) - 2;

    display_pixel_on <= row_readdata(y)(inner_x);

end rtl;
```

de2_wm8731_audio.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_wm8731_audio is
port (
    clk : in std_logic;       --  Audio CODEC Chip Clock AUD_XCK (11.29 MHz)
    reset_n : in std_logic;
    test_mode : in std_logic;       --    Audio CODEC controller test mode
    audio_request : out std_logic;  --    Audio controller request new data
    data : in std_logic_vector(15 downto 0);
    -- Audio interface signals
    AUD_ADCLRCK  : out  std_logic;   --    Audio CODEC ADC LR Clock
    AUD_ADCDAT   : in   std_logic;   --    Audio CODEC ADC Data
    AUD_DACLRCK  : out  std_logic;   --    Audio CODEC DAC LR Clock
    AUD_DACDAT   : out  std_logic;   --    Audio CODEC DAC Data
    AUD_BCLK     : inout std_logic  --    Audio CODEC Bit-Stream Clock
  );
end  de2_wm8731_audio;

architecture rtl of de2_wm8731_audio is

    signal lrck : std_logic;
    signal bclk : std_logic;

    signal lrck_divider : unsigned(7 downto 0);
    signal bclk_divider : unsigned(2 downto 0);

    signal set_lrck : std_logic;
    signal clr_bclk : std_logic;
    signal lrck_lat : std_logic;

    signal shift_out : std_logic_vector(15 downto 0);

    signal sin_out     : std_logic_vector(15 downto 0);
    signal sin_counter : unsigned(5 downto 0);

begin

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        lrck_divider <= (others => '1');
      else
        lrck_divider <= lrck_divider + 1;
      end if;
    end if;
  end process;

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' or set_lrck = '1' then
        bclk_divider <= (others => '0');
      else
        bclk_divider <= bclk_divider + 1;
      end if;
    end if;
  end process;

  set_lrck <= '1' when lrck_divider = x"7F" or lrck_divider = x"FF" else '0';
  lrck <= not lrck_divider(7); -- high first, then low

  -- BCLK divider
  bclk <= bclk_divider(2);
  clr_bclk <= '1' when bclk_divider = "111" else '0';

  -- Audio data shift output
  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        shift_out <= (others => '0');
      elsif set_lrck = '1' then
        if test_mode = '1' then
          shift_out <= sin_out;
        else
```

```
          shift_out <= data;
        end if;
      elsif clr_bclk = '1' then
        shift_out <= shift_out (14 downto 0) & '0';
      end if;
    end if;
  end process;

    -- Audio outputs

    AUD_ADCLRCK  <= lrck;
    AUD_DACLRCK  <= lrck;
    AUD_DACDAT   <= shift_out(15);
    AUD_BCLK     <= bclk;

    -- Self test with Sin wave

    process(clk)
    begin
      if rising_edge(clk) then
        if reset_n = '0' then
            sin_counter <= (others => '0');
        elsif lrck_divider = x"ff"  then
          if sin_counter = "101111" then
            sin_counter <= "000000";
          else
            sin_counter <= sin_counter + 1;
          end if;
        end if;
      end if;
    end process;

    audio_request <= '1' when lrck_divider = x"fe" else '0';

  with sin_counter select sin_out <=
    X"0000" when "000000",
    X"10b4" when "000001",
    X"2120" when "000010",
    X"30fb" when "000011",
    X"3fff" when "000100",
    X"4deb" when "000101",
    X"5a81" when "000110",
    X"658b" when "000111",
    X"6ed9" when "001000",
    X"7640" when "001001",
    X"7ba2" when "001010",
    X"7ee6" when "001011",
    X"7fff" when "001100",
    X"7ee6" when "001101",
    X"7ba2" when "001110",
    X"7640" when "001111",
    X"6ed9" when "010000",
    X"658b" when "010001",
    X"5a81" when "010010",
    X"4deb" when "010011",
    X"3fff" when "010100",
    X"30fb" when "010101",
    X"2120" when "010110",
    X"10b4" when "010111",
    X"0000" when "011000",
    X"ef4b" when "011001",
    X"dee0" when "011010",
    X"cf05" when "011011",
    X"c001" when "011100",
    X"b215" when "011101",
    X"a57e" when "011110",
    X"9a74" when "011111",
    X"9127" when "100000",
    X"89bf" when "100001",
    X"845d" when "100010",
    X"8119" when "100011",
    X"8000" when "100100",
    X"8119" when "100101",
    X"845d" when "100110",
    X"89bf" when "100111",
    X"9127" when "101000",
    X"9a74" when "101001",
    X"a57e" when "101010",
    X"b215" when "101011",
    X"c000" when "101100",
    X"cf05" when "101101",
    X"dee0" when "101110",
    X"ef4b" when "101111",
    X"0000" when others;

end architecture;
```

dft_stage1.vhd

```
    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

    entity dft_stage1 is
        port (tdom_data : in signed(15 downto 0);
              tdom_addr : out unsigned(3 downto 0);

              clk : in std_logic;
              reset : in std_logic;

              rom_data : in signed(31 downto 0);
              rom_addr : out unsigned(7 downto 0);

              rom_real : out signed(15 downto 0);
              rom_imag : out signed(15 downto 0);
              tdom_real : out signed(15 downto 0);
              outk : out unsigned(3 downto 0);
              write : out std_logic;
              done : out std_logic);
    end dft_stage1;

    architecture rtl of dft_stage1 is
        signal n : unsigned(3 downto 0) := x"0";
        signal k : unsigned(3 downto 0) := x"0";
        signal prevk : unsigned(3 downto 0) := x"0";
        signal done_intern : std_logic := '0';
        signal write_intern : std_logic := '0';
    begin
        rom_addr <= k & n;
        tdom_addr <= n;

        rom_real <= rom_data(31 downto 16);
        rom_imag <= rom_data(15 downto 0);
        tdom_real <= tdom_data;

        process (clk)
        begin
            if rising_edge(clk) then
                done <= done_intern;
                write <= write_intern;
                outk <= prevk;
                prevk <= k;
            end if;
        end process;
        process (clk)
        begin
            if rising_edge(clk) then
                if reset = '1' then
                    k <= x"0";
                    n <= x"0";
                    write_intern <= '1';
                    done_intern <= '0';
                elsif done_intern = '1' then
                    write_intern <= '0';
                    k <= x"0";
                    n <= x"0";
                elsif k = x"f" and n = x"f" then
                    write_intern <= '1';
                    done_intern <= '1';
                elsif n = x"f" then
                    k <= k + x"1";
                    n <= x"0";
                    write_intern <= '1';
                else
                    n <= n + x"1";
                    write_intern <= '0';
                end if;
            end if;
        end process;
    end rtl;
```

dft_stage2.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dft_stage2 is
    port (rom_real : in signed(15 downto 0);
          rom_imag : in signed(15 downto 0);
          tdom_real : in signed(15 downto 0);

          clk : std_logic;
          reset : std_logic;

          res_real : out signed(31 downto 0);
          res_imag : out signed(31 downto 0);

          ink : in unsigned(3 downto 0);
          outk : out unsigned(3 downto 0);

          inwrite : in std_logic;
          outwrite : out std_logic;

          indone : in std_logic;
          outdone : out std_logic);
end dft_stage2;

architecture rtl of dft_stage2 is
    signal mult_real : signed(31 downto 0);
    signal mult_imag : signed(31 downto 0);
    signal real_copy_bit : std_logic;
    signal imag_copy_bit : std_logic;
begin
    real_copy_bit <= mult_real(31);
    imag_copy_bit <= mult_imag(31);

    REALM : entity work.mult port map (
        dataa => std_logic_vector(rom_real),
        datab => std_logic_vector(tdom_real),
        signed(result) => mult_real
    );

    IMAGM : entity work.mult port map (
        dataa => std_logic_vector(rom_imag),
        datab => std_logic_vector(tdom_real),
        signed(result) => mult_imag
    );

    process (clk)
    begin
        if rising_edge(clk) then
            if reset = '1' then
                outwrite <= '0';
                outdone <= '0';
                outk <= x"0";
                res_real <= (others => '0');
                res_imag <= (others => '0');
            else
                outwrite <= inwrite;
                outdone <= indone;
                outk <= ink;
                res_real <= (3 downto 0 => real_copy_bit) & mult_real(31 downto 4);
                res_imag <= (3 downto 0 => imag_copy_bit) & mult_imag(31 downto 4);
            end if;
        end if;
    end process;
end rtl;
```

dft_stage3.vhd

```
    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

    entity dft_stage3 is
        port (mult_real : in signed(31 downto 0);
              mult_imag : in signed(31 downto 0);

              clk : in std_logic;
              reset : in std_logic;

              indone : in std_logic;
              inwrite : in std_logic;
              k : in unsigned(3 downto 0);

              outdone : out std_logic;
              outwrite : out std_logic;

              fdom_data : out signed(31 downto 0);
              fdom_addr : out unsigned(3 downto 0));
    end dft_stage3;

    architecture rtl of dft_stage3 is
        signal sum_real : signed(31 downto 0);
        signal sum_imag : signed(31 downto 0);
    begin
        process (clk)
        begin
            if rising_edge(clk) then
                outdone <= indone;
                outwrite <= inwrite;
                if reset = '1' then
                    sum_real <= (others => '0');
                    sum_imag <= (others => '0');
                    outdone <= '0';
                    outwrite <= '0';
                    fdom_data <= (others => '0');
                    fdom_addr <= (others => '0');
                elsif inwrite = '1' then
                    fdom_addr <= k;
                    fdom_data <= sum_real(31 downto 16) & sum_imag(31 downto 16);
                    sum_real <= mult_real;
                    sum_imag <= mult_imag;
                else
                    sum_real <= sum_real + mult_real;
                    sum_imag <= sum_imag + mult_imag;
                end if;
            end if;
        end process;
    end rtl;
```

dft_tb.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dft_tb is
end dft_tb;

architecture sim of dft_tb is
    signal clk : std_logic := '1';
    signal reset : std_logic;
    signal done : std_logic;
    signal read_addr : unsigned(3 downto 0);
    signal read_data : signed(31 downto 0);
    signal rom_addr : unsigned(7 downto 0);
    signal rom_data : signed(31 downto 0);
    signal tdom_addr : unsigned(3 downto 0);
    signal tdom_data : signed(15 downto 0);
    signal fdom_addr : unsigned(3 downto 0);
    signal fdom_data : signed(31 downto 0);
    signal fdom_write : std_logic;

    type rom_type is array(0 to 15) of signed(15 downto 0);
    constant tdom_rom : rom_type := (x"7fff", x"12a0", x"856d", x"c9b4",
                                     x"6ac5", x"555f", x"ae14", x"92c9",
                                     x"3223", x"7bcf", x"f1e6", x"8016",
                                     x"e8e1", x"792f", x"3a64", x"97d0");

    type expected_type is array(0 to 15) of signed(31 downto 0);
    constant expected : expected_type :=
        (x"01b70000", x"018dfe55", x"00d8fbbc", x"fd5ff2cf",
         x"0d2f1744", x"061605ec", x"050e02ce", x"04b80136",
         x"04a10000", x"04b8fec9", x"050efd31", x"0616fa13",
         x"0d2fe8bb", x"fd5f0d30", x"00d80443", x"018d01aa");
begin
    clk <= not clk after 10 ns;

    process (clk)
    begin
        if rising_edge(clk) then
            tdom_data <= tdom_rom(to_integer(tdom_addr));
        end if;
    end process;

    DFT : entity work.dft_top port map (
        clk => clk,
        reset => reset,
        done => done,
        tdom_data => tdom_data,
        tdom_addr => tdom_addr,
        rom_data => rom_data,
        rom_addr => rom_addr,
        fdom_data => fdom_data,
        fdom_addr => fdom_addr,
        fdom_write => fdom_write
    );

    COEFF_ROM : entity work.dft_coeff_rom port map (
        clk => clk,
        data_low => rom_data,
        addr_low => rom_addr,
        addr_high => (others => '0')
    );

    FDOM_RAM : entity work.fft_fdom_ram port map (
        clk => clk,
        reset => reset,

        writedata_low => fdom_data,
        writeaddr_low => fdom_addr,
        write_en_low => fdom_write,
        readaddr_low => read_addr,
        readdata_low => read_data,

        writedata_high => (others => '0'),
        writeaddr_high => (others => '0'),
        write_en_high => '0',
        readaddr_high => (others => '0'),

        stage => "00",
        step => "000"
    );

    process
        variable i : integer range 0 to 16;
    begin
        reset <= '1';
        wait for 20 ns;
        reset <= '0';
        read_addr <= x"0";
        wait for 5200 ns; -- 5220 ns
        assert done = '1';

        i := 0;
        while i < 16 loop
            read_addr <= to_unsigned(i, 4);
            wait for 40 ns;
            assert read_data = expected(i);
            i := i + 1;
        end loop; -- 5860

        reset <= '1';
        wait for 20 ns; -- 5880
        reset <= '0';
        wait for 20 ns; -- 5900
        assert done = '0';
        wait;
    end process;
end sim;
```

dft_top.vhd

```
    library ieee;
    use ieee.std_logic_1164.all;
```

```
use ieee.numeric_std.all;

entity dft_top is
    port (tdom_data : in signed(15 downto 0);
          tdom_addr : out unsigned(3 downto 0);
          clk : in std_logic;
          reset : in std_logic;
          rom_data : in signed(31 downto 0);
          rom_addr : out unsigned(7 downto 0);
          fdom_data : out signed(31 downto 0);
          fdom_addr : out unsigned(3 downto 0);
          fdom_write : out std_logic;
          done : out std_logic);
end dft_top;

architecture rtl of dft_top is
    signal s1_rom_real : signed(15 downto 0);
    signal s1_rom_imag : signed(15 downto 0);
    signal s1_tdom_real : signed(15 downto 0);
    signal s1_k : unsigned(3 downto 0);
    signal s1_write : std_logic;
    signal s1_done : std_logic;

    signal s2_k : unsigned(3 downto 0);
    signal s2_write : std_logic;
    signal s2_done : std_logic;
    signal s2_res_real : signed(31 downto 0);
    signal s2_res_imag : signed(31 downto 0);

    signal s3_done : std_logic;
    signal done_delay : unsigned(2 downto 0);
begin
    -- make sure done goes low right after reset
    -- hold it there until first input propagates through pipeline
    done <= '1' when s3_done = '1' and done_delay = "111" else '0';

    process (clk)
    begin
        if rising_edge(clk) then
            if reset = '1' then
                done_delay <= "000";
            elsif done_delay /= "111" then
                done_delay <= done_delay + "1";
            end if;
        end if;
    end process;

    S1 : entity work.dft_stage1 port map (
        tdom_data => tdom_data,
        tdom_addr => tdom_addr,

        clk => clk,
        reset => reset,

        rom_data => rom_data,
        rom_addr => rom_addr,

        rom_real => s1_rom_real,
        rom_imag => s1_rom_imag,
        tdom_real => s1_tdom_real,
        outk => s1_k,
        write => s1_write,
        done => s1_done
    );

    S2 : entity work.dft_stage2 port map (
        rom_real => s1_rom_real,
        rom_imag => s1_rom_imag,
        tdom_real => s1_tdom_real,

        clk => clk,
        reset => reset,

        res_real => s2_res_real,
        res_imag => s2_res_imag,

        ink => s1_k,
        outk => s2_k,
        inwrite => s1_write,
        outwrite => s2_write,
        indone => s1_done,
        outdone => s2_done
    );

    S3 : entity work.dft_stage3 port map (
        mult_real => s2_res_real,
        mult_imag => s2_res_imag,

        clk => clk,
        reset => reset,

        indone => s2_done,
        outdone => s3_done,
        inwrite => s2_write,
        outwrite => fdom_write,
        k => s2_k,

        fdom_addr => fdom_addr,
        fdom_data => fdom_data
    );
end rtl;
```

fft_controller.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fft_controller is
    port (clk : in std_logic;
          start : in std_logic;
          done : out std_logic;

          tdom_data_in : in signed(15 downto 0);
          tdom_addr_in : in unsigned(7 downto 0);
          tdom_write   : in std_logic;

          fdom_data_out : out signed(31 downto 0);
          fdom_addr_out : in unsigned(7 downto 0));
end fft_controller;

architecture rtl of fft_controller is
    type control_state_type is (idle, dftsetup, dftcomp,
                                recomb_setup, recomb_comp);
    signal tdom_addr_even : unsigned(3 downto 0);
    signal tdom_data_even : signed(15 downto 0);
    signal tdom_addr_odd : unsigned(3 downto 0);
    signal tdom_data_odd : signed(15 downto 0);
    signal tdom_sel : unsigned(2 downto 0);
    signal control_state : control_state_type;
    signal last_state : control_state_type;
    signal fdom_writedata_low : signed(31 downto 0);
    signal fdom_readdata_low : signed(31 downto 0);
    signal fdom_readaddr_low : unsigned(3 downto 0);
    signal fdom_writeaddr_low : unsigned(3 downto 0);
    signal fdom_write_en_low : std_logic;
    signal fdom_writedata_high : signed(31 downto 0);
    signal fdom_readdata_high : signed(31 downto 0);
    signal fdom_readaddr_high : unsigned(3 downto 0);
    signal fdom_writeaddr_high : unsigned(3 downto 0);
    signal fdom_write_en_high : std_logic;
    signal recomb_stage : unsigned(1 downto 0);
    signal comp_step : unsigned(2 downto 0);
    signal fdom_step : unsigned(2 downto 0);
    signal dft_rom_data_low : signed(31 downto 0);
    signal dft_rom_addr_low : unsigned(7 downto 0);
    signal dft_out_data_low : signed(31 downto 0);
    signal dft_out_addr_low : unsigned(3 downto 0);
    signal dft_out_write_low : std_logic;
    signal dft_rom_data_high : signed(31 downto 0);
    signal dft_rom_addr_high : unsigned(7 downto 0);
    signal dft_out_data_high : signed(31 downto 0);
    signal dft_out_addr_high : unsigned(3 downto 0);
    signal dft_out_write_high : std_logic;
    signal dft_done : std_logic_vector(1 downto 0);
    signal dft_reset : std_logic;
    signal recomb_reset : std_logic;
    -- This helps us map the DFT inputs to the DFT outputs
    -- The mappings are (0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15)
    -- We do the even and odd mappings simultaneously
    -- Thus, mapping(i) = 2 * fft_reorder(i) for even
    -- and mapping(i) = 2 * fft_reorder(i) + 1 for odd
    type fft_reorder_type is array(0 to 7) of unsigned(2 downto 0);
    constant fft_reorder : fft_reorder_type := ("000", "100", "010", "110",
                                                "001", "101", "011", "111");
    signal rcrom16_data : signed(31 downto 0);
    signal rcrom32_data : signed(31 downto 0);
    signal rcrom64_data : signed(31 downto 0);
    signal rcrom128_data : signed(31 downto 0);
    signal rcromcur_addr : unsigned(3 downto 0);
    signal rcromcur_data : signed(31 downto 0);
    signal rcrom32_addr : unsigned(4 downto 0);
    signal rcrom64_addr : unsigned(5 downto 0);
    signal rcrom128_addr : unsigned(6 downto 0);

    signal recomb_writeaddr_low : unsigned(3 downto 0);
    signal recomb_writedata_low : signed(31 downto 0);
    signal recomb_readaddr_low : unsigned(3 downto 0);
    signal recomb_readdata_low : signed(31 downto 0);
    signal recomb_write_low : std_logic;
    signal recomb_writeaddr_high : unsigned(3 downto 0);
    signal recomb_writedata_high : signed(31 downto 0);
    signal recomb_readaddr_high : unsigned(3 downto 0);
    signal recomb_readdata_high : signed(31 downto 0);
    signal recomb_write_high : std_logic;
    signal recomb_done : std_logic;
begin

    TDOM_RAM : entity work.fft_tdom_ram port map (
        clk => clk,

        readaddr_even => tdom_addr_even,
        readdata_even => tdom_data_even,
        readaddr_odd => tdom_addr_odd,
        readdata_odd => tdom_data_odd,
        readsel => tdom_sel,

        writeaddr => tdom_addr_in,
        writedata => tdom_data_in,
        write_en => tdom_write
    );
```

```vhdl
FDOM_RAM : entity work.fft_fdom_ram port map (
    readdata_low => fdom_readdata_low,
    readaddr_low => fdom_readaddr_low,
    writedata_low => fdom_writedata_low,
    writeaddr_low => fdom_writeaddr_low,
    write_en_low => fdom_write_en_low,
    readdata_high => fdom_readdata_high,
    readaddr_high => fdom_readaddr_high,
    writedata_high => fdom_writedata_high,
    writeaddr_high => fdom_writeaddr_high,
    write_en_high => fdom_write_en_high,
    stage => recomb_stage,
    step => fdom_step,
    clk => clk
);

COEFF_ROM : entity work.dft_lut port map (
    clock => clk,
    address_a => std_logic_vector(dft_rom_addr_low),
    address_b => std_logic_vector(dft_rom_addr_high),
    signed(q_a) => dft_rom_data_low,
    signed(q_b) => dft_rom_data_high
);

tdom_sel <= fft_reorder(to_integer(comp_step));

DFT_EVEN : entity work.dft_top port map (
    tdom_data => tdom_data_even,
    tdom_addr => tdom_addr_even,

    clk => clk,
    reset => dft_reset,

    rom_data => dft_rom_data_low,
    rom_addr => dft_rom_addr_low,

    fdom_data => dft_out_data_low,
    fdom_addr => dft_out_addr_low,
    fdom_write => dft_out_write_low,
    done => dft_done(0)
);

DFT_ODD : entity work.dft_top port map (
    tdom_data => tdom_data_odd,
    tdom_addr => tdom_addr_odd,

    clk => clk,
    reset => dft_reset,

    rom_data => dft_rom_data_high,
    rom_addr => dft_rom_addr_high,

    fdom_data => dft_out_data_high,
    fdom_addr => dft_out_addr_high,
    fdom_write => dft_out_write_high,
    done => dft_done(1)
);

-- Multiplex between dft and recomb units for the fdom ram
with control_state select fdom_writedata_low <=
    dft_out_data_low when dftsetup | dftcomp,
    recomb_writedata_low when recomb_setup | recomb_comp,
    (others => '0') when others;

with control_state select fdom_writeaddr_low <=
    dft_out_addr_low when dftsetup | dftcomp,
    recomb_writeaddr_low when recomb_setup | recomb_comp,
    (others => '0') when others;
```

```
with control_state select fdom_write_en_low <=
    dft_out_write_low when dftsetup | dftcomp,
    recomb_write_low when recomb_setup | recomb_comp,
    '0' when others;

with control_state select fdom_readaddr_low <=
    recomb_readaddr_low when recomb_setup | recomb_comp,
    fdom_addr_out(3 downto 0) when idle,
    (others => '0') when others;

recomb_readdata_low <= fdom_readdata_low;

with control_state select fdom_writedata_high <=
    dft_out_data_high when dftsetup | dftcomp,
    recomb_writedata_high when recomb_setup | recomb_comp,
    (others => '0') when others;

with control_state select fdom_writeaddr_high <=
    dft_out_addr_high when dftsetup | dftcomp,
    recomb_writeaddr_high when recomb_setup | recomb_comp,
    (others => '0') when others;

with control_state select fdom_write_en_high <=
    dft_out_write_high when dftsetup | dftcomp,
    recomb_write_high when recomb_setup | recomb_comp,
    '0' when others;

with control_state select fdom_readaddr_high <=
    recomb_readaddr_high when recomb_setup | recomb_comp,
    fdom_addr_out(3 downto 0) when idle,
    (others => '0') when others;

recomb_readdata_high <= fdom_readdata_high;

-- highest bit of fdom address determines which half of the RAM
-- the data is pulled out of
fdom_data_out <= fdom_readdata_high when fdom_addr_out(7) = '1' else
                 fdom_readdata_low;
-- when idle, allow the fdom read address to select the step
-- when computing, let comp_step determine it
fdom_step <= fdom_addr_out(6 downto 4) when
                control_state = idle else comp_step;

RECOMB : entity work.fft_recomb port map (
    clk => clk,
    reset => recomb_reset,
    rom_addr => rcromcur_addr,
    rom_data => rcromcur_data,
    low_readaddr => recomb_readaddr_low,
    low_writeaddr => recomb_writeaddr_low,
    low_readdata => recomb_readdata_low,
    low_writedata => recomb_writedata_low,
    low_write_en => recomb_write_low,
    high_readaddr => recomb_readaddr_high,
    high_writeaddr => recomb_writeaddr_high,
    high_readdata => recomb_readdata_high,
    high_writedata => recomb_writedata_high,
    high_write_en => recomb_write_high,
    done => recomb_done
);

-- select which recomb rom to use based on recomb_stage
with recomb_stage select rcromcur_data <=
    rcrom16_data when "00",
    rcrom32_data when "01",
    rcrom64_data when "10",
    rcrom128_data when others;
```

```
RCR16 : entity work.rcrom16 port map (
    address => std_logic_vector(rcromcur_addr),
    signed(q) => rcrom16_data,
    clock => clk
);

rcrom32_addr <= comp_step(0) & rcromcur_addr;
RCR32 : entity work.rcrom32 port map (
    address => std_logic_vector(rcrom32_addr),
    signed(q) => rcrom32_data,
    clock => clk
);

rcrom64_addr <= comp_step(1 downto 0) & rcromcur_addr;
RCR64 : entity work.rcrom64 port map (
    address => std_logic_vector(rcrom64_addr),
    signed(q) => rcrom64_data,
    clock => clk
);

rcrom128_addr <= comp_step & rcromcur_addr;
RCR128 : entity work.rcrom128 port map (
    address => std_logic_vector(rcrom128_addr),
    signed(q) => rcrom128_data,
    clock => clk
);

done <= '1' when control_state = idle else '0';
dft_reset <= '1' when control_state = dftsetup else '0';
recomb_reset <= '1' when control_state = recomb_setup else '0';

process (clk)
begin
    if rising_edge(clk) then
        last_state <= control_state;
        case control_state is
            when idle =>
                -- reads in the idle state must be done with
                -- recomb_stage at "11" so that low and high
                -- split the ram into two contiguous chunks
                recomb_stage <= "11";
                comp_step <= "111";
                if start = '1' then
                    control_state <= dftsetup;
                end if;
            when dftsetup =>
                -- same thing goes for dft
                -- recomb_stage set to "11"
                recomb_stage <= "11";
                -- first time in dftcomp will see comp_step = 0
                comp_step <= comp_step + 1;
                control_state <= dftcomp;
            when dftcomp =>
                if dft_done = "11" then
                    if comp_step = "111" then
                        control_state <= recomb_setup;
                    else
                        control_state <= dftsetup;
                    end if;
                end if;
            when recomb_setup =>
                -- go through steps and then through stages
                if comp_step = "111" then
                    recomb_stage <= recomb_stage + 1;
                    comp_step <= "000";
                else
                    comp_step <= comp_step + 1;
                    end if;
                    control_state <= recomb_comp;
                when recomb_comp =>
                    if recomb_done = '1' then
                        if comp_step = "111" and recomb_stage = "11" then
                            control_state <= idle;
                        else
                            control_state <= recomb_setup;
                        end if;
                    end if;
            end case;
        end if;
    end process;
end rtl;
```

fft_tb.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fft_tb is
end fft_tb;

architecture sim of fft_tb is
    signal tdom_addr : unsigned(7 downto 0);
    signal tdom_data : signed(15 downto 0);
    signal tdom_write : std_logic;
    signal fdom_addr : unsigned(7 downto 0);
    signal fdom_data : signed(31 downto 0);
    signal clk : std_logic := '1';
    signal start : std_logic;
    signal done : std_logic;
    type rom_type is array(0 to 255) of signed(15 downto 0);
    constant rom_data : rom_type := (x"7fff", x"7eb2", x"7ad5", x"747b", x"6bc4", x"60de", x"
        5401", x"4571", x"3579", x"246b", x"12a0", x"0074", x"ee47", x"dc75", x"cb5c", x"bb53"
        , x"acaf", x"9fbb", x"94bb", x"8be7", x"856d", x"816f", x"8002", x"812d", x"84ea", x"8
        b25", x"93bf", x"9e8a", x"ab4f", x"b9cb", x"c9b4", x"dab6", x"ec79", x"fea2", x"10d1",
         x"22aa", x"33cf", x"43e7", x"529f", x"5faa", x"6ac5", x"73b6", x"7a4f", x"7e6d", x"7
        ffb", x"7ef1", x"7b55", x"7539", x"6cbd", x"620c", x"555f", x"46f7", x"371f", x"2629",
         x"146d", x"0247", x"f016", x"de37", x"cd07", x"bcdf", x"ae14", x"a0f1", x"95bc", x"8
        cae", x"85f7", x"81b8", x"8009", x"80f1", x"846d", x"8a6a", x"92c9", x"9d5e", x"a9f3",
         x"b847", x"c80e", x"d8f8", x"eaad", x"fcd0", x"0f02", x"20e8", x"3223", x"425a", x"
        5138", x"5e72", x"69c1", x"72ec", x"79c2", x"7e21", x"7ff1", x"7f2a", x"7bcf", x"75f2"
        , x"6db0", x"6336", x"56b9", x"487a", x"38c3", x"27e6", x"1639", x"041a", x"f1e6", x"
        dffa", x"ceb4", x"be6e", x"af7d", x"a22d", x"96c3", x"8d7c", x"8687", x"8208", x"8016"
        , x"80bc", x"83f7", x"89b5", x"91d8", x"9c37", x"a89c", x"b6c6", x"c66c", x"d73d", x"
        e8e1", x"fafd", x"0d32", x"1f24", x"3074", x"40c9", x"4fcd", x"5d34", x"68b7", x"721b"
        , x"792f", x"7dce", x"7fe1", x"7f5c", x"7c42", x"76a4", x"6e9e", x"645a", x"580e", x"
        49f9", x"3a64", x"29a0", x"1805", x"05ec", x"f3b6", x"e1bf", x"d064", x"c001", x"b0ea"
        , x"a36d", x"97d0", x"8e50", x"871d", x"825e", x"802a", x"808e", x"8387", x"8905", x"
        90ed", x"9b15", x"a749", x"b549", x"c4cd", x"d583", x"e716", x"f92b", x"0b61", x"1d5e"
        , x"2ec3", x"3f34", x"4e5e", x"5bf2", x"67a8", x"7145", x"7895", x"7d75", x"7fc9", x"7
        f87", x"7caf", x"7750", x"6f86", x"657a", x"595f", x"4b74", x"3c02", x"2b59", x"19cf",
         x"07be", x"f587", x"e385", x"d217", x"c197", x"b25b", x"a4b1", x"98e2", x"8f29", x"87
        ba", x"82ba", x"8045", x"8066", x"831d", x"885c", x"9008", x"99f9", x"a5fb", x"b3d0",
        x"c330", x"d3cc", x"e54d", x"f759", x"0990", x"1b97", x"2d0f", x"3d9d", x"4ceb", x"5
        aaa", x"6693", x"7068", x"77f6", x"7d15", x"7fab", x"7fab", x"7d15", x"77f6", x"7068",
         x"6693", x"5aaa", x"4ceb", x"3d9d", x"2d0f", x"1b97", x"0990", x"f759", x"e54d", x"
        d3cc", x"c330", x"b3d0", x"a5fb", x"99f9", x"9008", x"885c", x"831d", x"8066", x"8045"
        , x"82ba", x"87ba", x"8f29", x"98e2", x"a4b1", x"b25b", x"c197", x"d217", x"e385", x"
        f587", x"07be", x"19cf");
    type expected_type is array(0 to 255) of signed(31 downto 0);
    constant expected : expected_type := (x"ffac0000", x"ffaa0000", x"ffa20002", x"ff920001",
        x"ff62fff9", x"fe9cffda", x"065900df", x"01470022", x"00f60000", x"011bffc4", x"0444
        fdae", x"ff7200b1", x"fff10053", x"00120031", x"0020001c", x"0026000c", x"00260000", x
        "0024fff1", x"0020ffe5", x"0018ffd5", x"0009ffbd", x"ffd5ff75", x"01b101ce", x"0074002
        e", x"00630000", x"0078ffd4", x"01e0fe94", x"ffb9005e", x"fff10028", x"ffff0016", x"
        0005000b", x"00070005", x"00090000", x"0009fff9", x"0006fff3", x"0002ffea", x"fff7ffdb
        ", x"ffd2ffb1", x"01380116", x"004a001e", x"003a0000", x"003fffdf", x"00c6fee0", x"
        fff6004f", x"000b0023", x"00100015", x"0013000c", x"00150004", x"00140000", x"0014fff9
        ", x"0012fff2", x"000fffea", x"0007ffda", x"ffeaffb0", x"010a0111", x"004e001c", x"
        00450000", x"005affe6", x"0191ff30", x"ffb70034", x"ffe70015", x"fff3000b", x"fff80006
        ", x"fffa0003", x"fffc0000", x"fffbfffd", x"fff9fff9", x"fff5fff4", x"ffebffeb", x"
        ffc0ffcf", x"015700b9", x"004b0015", x"00380000", x"003dffe7", x"00c9ff22", x"fff1003d
        ", x"0007001b", x"000c0010", x"000e0009", x"000f0004", x"000e0000", x"000efffc", x"000
        dfff7", x"000cfff1", x"0009ffe8", x"fffdffcc", x"006800b1", x"00220012", x"001f0000",
        x"0028ffed", x"009bff59", x"ffec002d", x"fffe0014", x"0003000b", x"00050006", x"
        00060002", x"00060000", x"0006fffc", x"0005fff9", x"0003fff3", x"fffdffe9", x"ffe7ffcf
        ", x"00ba00b4", x"002f0014", x"00270000", x"002cffe9", x"0091ff2f", x"fff7003c", x"
        0008001c", x"000c0011", x"0010000a", x"00110005", x"00120000", x"0012fff9", x"0010fff2
        ", x"000dffea", x"0003ffdb", x"ffdcffb2", x"0159010b", x"0060001b", x"00560000", x"
        0072ffea", x"0220ff53", x"ff910028", x"ffd4000f", x"ffe40006", x"ffea0003", x"ffed0001
        ", x"ffee0000", x"ffeefffe", x"ffeafffc", x"ffe4fff9", x"ffd4fff1", x"ff92ffd8", x"
        022100ab", x"00730016", x"00560000", x"0061ffe4", x"015afef4", x"ffdc004d", x"00030023
        ", x"000c0015", x"0010000c", x"00120006", x"00120000", x"0012fffb", x"0010fff5", x"000
        effef", x"0009ffe3", x"fff7ffc3", x"009300d0", x"002c0016", x"00270000", x"0030ffec",
        x"00baff4c", x"ffe90030", x"fffd0016", x"0003000c", x"00050007", x"00070003", x"
        00070000", x"0007fffd", x"0006fff9", x"0004fff4", x"ffffffeb", x"ffecffd3", x"009c00a6
        ", x"00280012", x"00200000", x"0023ffed", x"0068ff4e", x"fffe0033", x"00090017", x"000
        c000f", x"000f0008", x"000f0004", x"00100000", x"0010fffb", x"000efff6", x"000dfff0",
        x"0007ffe4", x"fff2ffc2", x"00ca00dd", x"003e0018", x"00390000", x"004cffea", x"0159
        ff46", x"ffc10030", x"ffeb0015", x"fff5000b", x"fffa0006", x"fffc0003", x"fffc0000", x
        "fffbfffd", x"fff9fff9", x"fff5fff4", x"ffe9ffeb", x"ffb8ffcb", x"019300cf", x"005
        b0019", x"00460000", x"004fffe3", x"010bfeee", x"ffeb004f", x"00090025", x"00100016",
        x"0014000d", x"00150006", x"00160000", x"0016fffa", x"0015fff3", x"0012ffeb", x"000
        dffdc", x"fff7ffb0", x"00c8011f", x"00400020", x"003b0000", x"004cffe1", x"0139fee9",
        x"ffd4004d", x"fff80024", x"00030015", x"0007000c", x"000a0006", x"000a0000", x"000
        afffa", x"0007fff3", x"0001ffe9", x"fff3ffd7", x"ffbbffa1", x"01e2016a", x"0079002a",
        x"00650000", x"0078ffd1", x"01b3fe31", x"ffd7008a", x"000a0042", x"001a002b", x"
        0022001a", x"0027000d", x"00280000", x"0028fff3", x"0022ffe2", x"0015ffce", x"fff3ffab
        ", x"ff74ff4e", x"04470251", x"011e003b", x"00f80000", x"014affdc", x"065cff1f", x"
        fe9f0024", x"ff640005", x"ff94fffe", x"ffa6fffd", x"ffadffff");
begin
    FFT : entity work.fft_controller port map (
        tdom_data_in => tdom_data,
        tdom_addr_in => tdom_addr,
        tdom_write => tdom_write,

        fdom_data_out => fdom_data,
        fdom_addr_out => fdom_addr,

        clk => clk,
        start => start,
        done => done
    );

    clk <= not clk after 10 ns;

    process
        variable i : integer range 0 to 256;
    begin
        tdom_write <= '1';
        start <= '0';
        i := 0;

        while i < 256 loop
            tdom_data <= rom_data(i);
            tdom_addr <= to_unsigned(i, 8);
            wait for 20 ns;
            i := i + 1;
        end loop; -- 5120 ns

        tdom_write <= '0';
```

```
            start <= '1';
            fdom_addr <= x"00";
            wait for 20 ns; -- 5140 ns
            start <= '0';

            wait for 55860 ns; -- 61000 ns

            assert done = '1';

            i := 0;
            while i < 256 loop
                fdom_addr <= to_unsigned(i, 8);
                wait for 40 ns;
                assert fdom_data = expected(i);
                i := i + 1;
            end loop; -- 71240 ns

            wait;
        end process;
    end sim;
```

fft_fdom_ram.vhd

```
    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

    entity fft_fdom_ram is
        port (clk : std_logic;
              writedata_low : in signed(31 downto 0);
              writeaddr_low : in unsigned(3 downto 0);
              readdata_low : out signed(31 downto 0);
              readaddr_low : in unsigned(3 downto 0);
              write_en_low : in std_logic;
              writedata_high : in signed(31 downto 0);
              writeaddr_high : in unsigned(3 downto 0);
              readdata_high : out signed(31 downto 0);
              readaddr_high : in unsigned(3 downto 0);
              write_en_high : in std_logic;
              stage : in unsigned(1 downto 0);
              step  : in unsigned(2 downto 0));
    end fft_fdom_ram;

    architecture rtl of fft_fdom_ram is
        type data_array is array(0 to 15) of signed(31 downto 0);
        type addr_array is array(0 to 15) of unsigned(3 downto 0);
        signal row_writedata : data_array;
        signal row_writeaddr : addr_array;
        signal row_write_en : std_logic_vector(0 to 15);
        signal row_readdata : data_array;
        signal row_readaddr : addr_array;
        signal lowsel : unsigned(3 downto 0);
        signal highsel : unsigned(3 downto 0);
    begin
        WRITEGEN : for i in 0 to 15 generate
            row_writedata(i) <= writedata_low when lowsel = i else
                                writedata_high when highsel = i else
                                (others => '0');
            row_writeaddr(i) <= writeaddr_low when lowsel = i else
                                writeaddr_high when highsel = i else
                                (others => '0');
            row_write_en(i) <= write_en_low when lowsel = i else
                               write_en_high when highsel = i else '0';
            row_readaddr(i) <= readaddr_low when lowsel = i else
                               readaddr_high when highsel = i else
                               (others => '0');
        end generate WRITEGEN;

        readdata_low <= row_readdata(to_integer(lowsel));
        readdata_high <= row_readdata(to_integer(highsel));

        with stage select lowsel <=
            -- every even indexed row
            step & '0' when "00",
            -- top two rows of every group of four
            step(2 downto 1) & '0' & step(0) when "01",
            -- top four rows of every group of eight
            step(2) & '0' & step(1 downto 0) when "10",
            -- top 8 rows
            '0' & step when "11",
            "0000" when others;

        with stage select highsel <=
            -- every odd indexed row
            step & '1' when "00",
            -- bottom two rows of every group of four
            step(2 downto 1) & '1' & step(0) when "01",
            -- bottom four rows of every group of eight
            step(2) & '1' & step(1 downto 0) when "10",
            -- bottom eight rows
            '1' & step when "11",
            "1111" when others;

        LUMAP : for i in 0 to 15 generate
            ROW : entity work.fdom_row port map (
                clock => clk,
                data => std_logic_vector(row_writedata(i)),
                wraddress => std_logic_vector(row_writeaddr(i)),
                wren => row_write_en(i),
                signed(q) => row_readdata(i),
                rdaddress => std_logic_vector(row_readaddr(i))
            );
        end generate LUMAP;
    end rtl;
```

fft_recomb.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fft_recomb is
    port (clk : in std_logic;
          reset : in std_logic;
          done : out std_logic;
          rom_addr : out unsigned(3 downto 0);
          rom_data : in signed(31 downto 0);
          low_readaddr : out unsigned(3 downto 0);
          low_writeaddr : out unsigned(3 downto 0);
          low_readdata : in signed(31 downto 0);
          low_writedata : out signed(31 downto 0);
          low_write_en : out std_logic;
          high_readaddr : out unsigned(3 downto 0);
          high_writeaddr : out unsigned(3 downto 0);
          high_readdata : in signed(31 downto 0);
          high_writedata : out signed(31 downto 0);
          high_write_en : out std_logic);
end fft_recomb;

architecture rtl of fft_recomb is
    signal rom_real : signed(15 downto 0);
    signal rom_imag : signed(15 downto 0);
    signal even_real_s12 : signed(15 downto 0);
    signal even_imag_s12 : signed(15 downto 0);
    signal odd_real_s12 : signed(15 downto 0);
    signal odd_imag_s12 : signed(15 downto 0);
    signal even_real_s23 : signed(15 downto 0);
    signal even_imag_s23 : signed(15 downto 0);
    signal odd_real_s23 : signed(15 downto 0);
    signal odd_imag_s23 : signed(15 downto 0);
    signal addr_s12 : unsigned(3 downto 0);
    signal addr_s23 : unsigned(3 downto 0);
    signal write_s12 : std_logic;
    signal write_s23 : std_logic;
    signal pl_done : std_logic;
    signal done_delay : unsigned(2 downto 0);
begin
    S1 : entity work.recomb_stage1 port map (
        clk => clk,
        reset => reset,

        rom_addr => rom_addr,
        rom_data => rom_data,

        low_readaddr => low_readaddr,
        low_readdata => low_readdata,

        high_readaddr => high_readaddr,
        high_readdata => high_readdata,

        addrout => addr_s12,
        writeout => write_s12,

        even_real => even_real_s12,
        even_imag => even_imag_s12,
        odd_real => odd_real_s12,
        odd_imag => odd_imag_s12,
        rom_real => rom_real,
        rom_imag => rom_imag
    );

    S2 : entity work.recomb_stage2 port map (
        clk => clk,

        rom_real => rom_real,
        rom_imag => rom_imag,

        even_real_in => even_real_s12,
        even_imag_in => even_imag_s12,

        odd_real_in => odd_real_s12,
        odd_imag_in => odd_imag_s12,

        even_real_out => even_real_s23,
        even_imag_out => even_imag_s23,

        odd_real_out => odd_real_s23,
        odd_imag_out => odd_imag_s23,

        writein => write_s12,
        writeout => write_s23,
        addrin => addr_s12,
        addrout => addr_s23
    );

    S3 : entity work.recomb_stage3 port map (
        clk => clk,
        done => pl_done,

        even_real => even_real_s23,
        even_imag => even_imag_s23,
        odd_real => odd_real_s23,
        odd_imag => odd_imag_s23,

        addrin => addr_s23,
        writein => write_s23,

        low_writeaddr => low_writeaddr,
        low_writedata => low_writedata,
        low_write_en => low_write_en,

        high_writeaddr => high_writeaddr,
        high_writedata => high_writedata,
        high_write_en => high_write_en
    );

    -- make sure done goes low right after reset
    -- hold it there until first input propagates through pipeline
    done <= '1' when pl_done = '1' and done_delay = "111" else '0';

    process (clk)
    begin
        if rising_edge(clk) then
            if reset = '1' then
                done_delay <= "000";
            elsif done_delay /= "111" then
                done_delay <= done_delay + "1";
            end if;
        end if;
    end process;
end rtl;
```

fft_tdom_ram.vhd

```
    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

    entity fft_tdom_ram is
        port (clk : std_logic;

              readaddr_even : in unsigned(3 downto 0);
              readaddr_odd  : in unsigned(3 downto 0);
              readdata_even : out signed(15 downto 0);
              readdata_odd  : out signed(15 downto 0);
              readsel : in unsigned(2 downto 0);

              writeaddr : in unsigned(7 downto 0);
              writedata : in signed(15 downto 0);
              write_en : in std_logic);
    end fft_tdom_ram;

    architecture rtl of fft_tdom_ram is
        signal even_write_en : std_logic;
        signal odd_write_en : std_logic;
        signal short_waddr : unsigned(6 downto 0);
        signal even_raddr  : unsigned(6 downto 0);
        signal odd_raddr   : unsigned(6 downto 0);
    begin
        even_write_en <= write_en and (not writeaddr(0));
        odd_write_en <= write_en and writeaddr(0);

        short_waddr <= writeaddr(7 downto 1);
        even_raddr <= readaddr_even & readsel;
        odd_raddr <= readaddr_odd & readsel;
        EVEN_RAM : entity work.tdom_half_ram port map (
            clock => clk,
            data => std_logic_vector(writedata),
            rdaddress => std_logic_vector(even_raddr),
            wraddress => std_logic_vector(short_waddr),
            wren => even_write_en,
            signed(q) => readdata_even
        );

        ODD_RAM : entity work.tdom_half_ram port map (
            clock => clk,
            data => std_logic_vector(writedata),
            rdaddress => std_logic_vector(odd_raddr),
            wraddress => std_logic_vector(short_waddr),
            wren => odd_write_en,
            signed(q) => readdata_odd
        );
    end rtl;
```

recomb_stage1.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity recomb_stage1 is
    port (clk : in std_logic;
          reset : in std_logic;
          rom_addr : out unsigned(3 downto 0);
          rom_data : in signed(31 downto 0);
          low_readaddr : out unsigned(3 downto 0);
          low_readdata : in signed(31 downto 0);
          high_readaddr : out unsigned(3 downto 0);
          high_readdata : in signed(31 downto 0);
          addrout : out unsigned(3 downto 0);
          writeout : out std_logic;
          rom_real : out signed(15 downto 0);
          rom_imag : out signed(15 downto 0);
          even_real : out signed(15 downto 0);
          even_imag : out signed(15 downto 0);
          odd_real : out signed(15 downto 0);
          odd_imag : out signed(15 downto 0));
end recomb_stage1;

architecture rtl of recomb_stage1 is
    signal running : std_logic := '0';
    signal addr : unsigned(3 downto 0);
begin
    rom_addr <= addr;
    low_readaddr <= addr;
    high_readaddr <= addr;

    rom_real <= rom_data(31 downto 16);
    rom_imag <= rom_data(15 downto 0);
    even_real <= low_readdata(31 downto 16);
    even_imag <= low_readdata(15 downto 0);
    odd_real <= high_readdata(31 downto 16);
    odd_imag <= high_readdata(15 downto 0);

    process (clk)
    begin
        if rising_edge(clk) then
            writeout <= running;
            addrout <= addr;

            if reset = '1' then
```

```
                running <= '1';
                addr <= x"0";
            elsif addr = x"f" then
                running <= '0';
            else
                addr <= addr + "1";
            end if;
        end if;
    end process;
end rtl;
```

recomb_stage2.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity recomb_stage2 is
    port (clk : in std_logic;
          rom_real : in signed(15 downto 0);
          rom_imag : in signed(15 downto 0);
          even_real_in : in signed(15 downto 0);
          even_imag_in : in signed(15 downto 0);
          odd_real_in : in signed(15 downto 0);
          odd_imag_in : in signed(15 downto 0);
          even_real_out : out signed(15 downto 0);
          even_imag_out : out signed(15 downto 0);
          odd_real_out : out signed(15 downto 0);
          odd_imag_out : out signed(15 downto 0);
          writein : in std_logic;
          writeout : out std_logic;
          addrin : in unsigned(3 downto 0);
          addrout : out unsigned(3 downto 0));
end recomb_stage2;

architecture rtl of recomb_stage2 is
    signal odd_real_mid : signed(31 downto 0);
    signal odd_imag_mid : signed(31 downto 0);
    signal even_real_mid : signed(15 downto 0);
    signal even_imag_mid : signed(15 downto 0);
    signal addrmid : unsigned(3 downto 0);
    signal writemid : std_logic;
begin
    MULT : entity work.complex_mult port map (
        realx => odd_real_in,
        imagx => odd_imag_in,
        realy => rom_real,
        imagy => rom_imag,
        realz => odd_real_mid,
        imagz => odd_imag_mid,
        clk => clk
    );

    process (clk)
    begin
        if rising_edge(clk) then
            writemid <= writein;
            writeout <= writemid;
            addrmid <= addrin;
            addrout <= addrmid;
            even_real_mid <= even_real_in;
            even_imag_mid <= even_imag_in;
            even_real_out <= even_real_mid;
            even_imag_out <= even_imag_mid;
            odd_real_out <= odd_real_mid(31 downto 16);
            odd_imag_out <= odd_imag_mid(31 downto 16);
        end if;
    end process;
end rtl;
```

recomb_stage3.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity recomb_stage3 is
    port (clk : in std_logic;
          done : out std_logic;

          even_real : in signed(15 downto 0);
          even_imag : in signed(15 downto 0);
          odd_real : in signed(15 downto 0);
          odd_imag : in signed(15 downto 0);

          addrin : in unsigned(3 downto 0);
          writein : in std_logic;

          low_writeaddr : out unsigned(3 downto 0);
          low_writedata : out signed(31 downto 0);
          low_write_en : out std_logic;

          high_writeaddr : out unsigned(3 downto 0);
          high_writedata : out signed(31 downto 0);
          high_write_en : out std_logic);
end recomb_stage3;

architecture rtl of recomb_stage3 is
    signal write_en : std_logic;
    signal writeaddr : unsigned(3 downto 0);
    signal even_real_shift : signed(15 downto 0);
    signal even_imag_shift : signed(15 downto 0);
    signal odd_real_shift : signed(15 downto 0);
    signal odd_imag_shift : signed(15 downto 0);
    signal low_sum_real : signed(15 downto 0);
    signal low_sum_imag : signed(15 downto 0);
    signal high_diff_real : signed(15 downto 0);
    signal high_diff_imag : signed(15 downto 0);
begin
    low_write_en <= write_en;
    high_write_en <= write_en;
    low_writeaddr <= writeaddr;
    high_writeaddr <= writeaddr;

    even_real_shift <= even_real(15) & even_real(15 downto 1);
    even_imag_shift <= even_imag(15) & even_imag(15 downto 1);
    odd_real_shift <= odd_real(15) & odd_real(15 downto 1);
    odd_imag_shift <= odd_imag(15) & odd_imag(15 downto 1);

    low_sum_real <= even_real_shift + odd_real_shift;
    low_sum_imag <= even_imag_shift + odd_imag_shift;
    high_diff_real <= even_real_shift - odd_real_shift;
    high_diff_imag <= even_imag_shift - odd_imag_shift;

    process (clk)
    begin
        if rising_edge(clk) then
            done <= not writein;
            write_en <= writein;
            writeaddr <= addrin;
            low_writedata <= low_sum_real & low_sum_imag;
            high_writedata <= high_diff_real & high_diff_imag;
        end if;
    end process;
end rtl;
```

recomb_tb.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity recomb_tb is
end recomb_tb;

architecture sim of recomb_tb is
    signal clk : std_logic := '0';
    signal reset : std_logic;
    signal done : std_logic;
    signal rom_addr : unsigned(3 downto 0);
    signal rom_data : signed(31 downto 0);

    signal rc_low_readaddr : unsigned(3 downto 0);
    signal rc_low_writeaddr : unsigned(3 downto 0);
    signal rc_low_writedata : signed(31 downto 0);
    signal rc_low_write_en : std_logic;
    signal rc_high_readaddr : unsigned(3 downto 0);
    signal rc_high_writeaddr : unsigned(3 downto 0);
    signal rc_high_writedata : signed(31 downto 0);
    signal rc_high_write_en : std_logic;

    signal low_readaddr : unsigned(3 downto 0);
    signal low_writeaddr : unsigned(3 downto 0);
    signal low_readdata : signed(31 downto 0);
    signal low_writedata : signed(31 downto 0);
    signal low_write_en : std_logic;
    signal high_readaddr : unsigned(3 downto 0);
    signal high_writeaddr : unsigned(3 downto 0);
    signal high_readdata : signed(31 downto 0);
    signal high_writedata : signed(31 downto 0);
    signal high_write_en : std_logic;

    signal tb_readaddr : unsigned(3 downto 0);
    signal tb_writeaddr : unsigned(3 downto 0);
    signal tb_write_en : std_logic;
    signal tb_low_writedata : signed(31 downto 0);
    signal tb_high_writedata : signed(31 downto 0);

    signal stage : unsigned(1 downto 0);
    signal step : unsigned(2 downto 0);

    type expected_type is array(0 to 31) of signed(31 downto 0);
    type mem_type is array(0 to 15) of signed(31 downto 0);

    signal user_mem : std_logic;

    signal low_mem : mem_type :=
        (x"63147d3a", x"0c3ac903", x"f39e20aa", x"5fa3080c",
         x"8c34609a", x"9cf9709f", x"4efb55bf", x"271c2ce4",
         x"9b959d9b", x"ddfdc775", x"609585d9", x"b5f76fed",
         x"29fb2867", x"0c38abda", x"08a34c89", x"56df70a1");
    signal high_mem : mem_type :=
        (x"9ac10175", x"d03e0edd", x"425d0d5c", x"ac236b31",
         x"228caea3", x"52ac3a8c", x"5a8fdf6b", x"2b4e796a",
         x"d87a7236", x"e1bc27e1", x"c19c3b48", x"4301716c",
         x"8db55085", x"c590beaa", x"5e2cb01d", x"9641db23");
    constant expected : expected_type :=
        (x"183a3efa", x"fa67e825", x"0922136a", x"1e621a4d",
         x"cc3521eb", x"d9f74070", x"302627c1", x"15aa1c5d",
         x"cdcacecd", x"f077e1c8", x"3641bd40", x"d1ac2835",
         x"293005f7", x"1241e381", x"ee9138b7", x"455c4159",
         x"4ada3e40", x"11d3e0dd", x"ea7c0d40", x"4140edbf",
         x"bfff3eaf", x"c301302e", x"1ed42dfd", x"11721087",
         x"cdcacecd", x"ed85e5ac", x"2a53c898", x"e44a47b7",
         x"00ca226f", x"f9f7c859", x"1a1113d1", x"11822f47");
begin
    low_readaddr <= tb_readaddr when user_mem = '1' else rc_low_readaddr;
    low_writeaddr <= tb_writeaddr when user_mem = '1' else rc_low_writeaddr;
    low_writedata <= tb_low_writedata when user_mem = '1' else rc_low_writedata;
    low_write_en <= tb_write_en when user_mem = '1' else rc_low_write_en;

    high_readaddr <= tb_readaddr when user_mem = '1' else rc_high_readaddr;
    high_writeaddr <= tb_writeaddr when user_mem = '1' else rc_high_writeaddr;
    high_writedata <= tb_high_writedata when user_mem = '1' else rc_high_writedata;
    high_write_en <= tb_write_en when user_mem = '1' else rc_high_write_en;

    RCR16 : entity work.recomb_rom16 port map (
        addr => rom_addr,
        data => rom_data,
        clk =>  clk
    );

    FDOM_RAM : entity work.fft_fdom_ram port map (
        readdata_low => low_readdata,
        readaddr_low => low_readaddr,
        writedata_low => low_writedata,
        writeaddr_low => low_writeaddr,
        write_en_low => low_write_en,
        readdata_high => high_readdata,
        readaddr_high => high_readaddr,
        writedata_high => high_writedata,
        writeaddr_high => high_writeaddr,
        write_en_high => high_write_en,
        reset => '0',
        stage => stage,
        step => step,
        clk => clk
    );

    RC : entity work.fft_recomb port map (
        clk => clk,
        reset => reset,
        done => done,
        rom_addr => rom_addr,
        rom_data => rom_data,
        low_readaddr => rc_low_readaddr,
        low_readdata => low_readdata,
        low_writeaddr => rc_low_writeaddr,
        low_writedata => rc_low_writedata,
        low_write_en => rc_low_write_en,
        high_readaddr => rc_high_readaddr,
        high_readdata => high_readdata,
        high_writeaddr => rc_high_writeaddr,
        high_writedata => rc_high_writedata,
        high_write_en => rc_high_write_en
    );

    clk <= not clk after 10 ns;

    process
        variable i : integer range 0 to 32;
    begin
        user_mem <= '1';
        stage <= "00";
        step <= "000";
        wait for 10 ns; -- 10 ns
        tb_write_en <= '1';

        i := 0;
        while i < 16 loop
            tb_writeaddr <= to_unsigned(i, 4);
            tb_low_writedata <= low_mem(i);
            tb_high_writedata <= high_mem(i);
            wait for 20 ns;
            i := i + 1;
        end loop; -- 330 ns

        tb_write_en <= '0';
        user_mem <= '0';
        stage <= "11";
        step <= "111";
        wait for 20 ns; -- 350 ns

        reset <= '1';
        wait for 20 ns; -- 370 ns
        stage <= "00";
        step <= "000";
        reset <= '0';

        wait for 420 ns; -- 790 ns
        assert done = '1';
        user_mem <= '1';
        wait for 20 ns; -- 810 ns

        i := 0;
        while i < 16 loop
            tb_readaddr <= to_unsigned(i, 4);
            wait for 40 ns;
            assert low_readdata = expected(i);
            assert high_readdata = expected(i + 16);
            i := i + 1;
        end loop; -- 1450 ns
        wait;
    end process;
end sim;
```

sd_controller.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sd_controller is
port (
    clk50           : in std_logic;
    cs              : out std_logic;
    mosi            : out std_logic;
    miso            : in std_logic;
    sclk            : out std_logic;
    start           : in std_logic;
    ready           : out std_logic;
    err             : out std_logic;
    waiting         : out std_logic;
    ccs             : out std_logic;

    writedata       : out signed(15 downto 0);
    writeaddr       : out unsigned(7 downto 0);
    write_en        : out std_logic;
    blockaddr       : in unsigned(31 downto 0);

    state_debug     : out std_logic_vector(7 downto 0);
    resp_debug      : out std_logic_vector(7 downto 0)
);
end sd_controller;

architecture rtl of sd_controller is
    signal clk_enable : std_logic := '1';
    signal clk_divider : unsigned(1 downto 0) := "00";
    signal counter : unsigned(7 downto 0);

    constant cmd0  : std_logic_vector(47 downto 0) := x"400000000095";
    constant cmd8  : std_logic_vector(47 downto 0) := x"48000001AA87";
    constant cmd55 : std_logic_vector(47 downto 0) := x"770000000065";
    -- IMPORTANT!!! HCS bit must be set in ACMD41, contrary to embed_lab9
    constant cmd41 : std_logic_vector(47 downto 0) := x"694000000077";
    constant cmd58 : std_logic_vector(47 downto 0) := x"7a00000000fd";
    signal command : std_logic_vector(47 downto 0);
    type sd_state is (reset_state, reset_clks1, reset_clks2,
                      send_cmd, wait_resp, recv_resp,
                      clear_input, check_clear,
                      check_cmd0, check_cmd8_head, check_cmd8_extra,
                      check_cmd58_head, check_cmd58_ccs,
                      check_cmd17, wait_block_start, write_word,
                      check_cmd55, check_cmd41, cmd_done, cmd_err);
    signal state : sd_state := reset_state;
    signal return_state : sd_state;
    signal sclk_sig : std_logic;
    signal response : std_logic_vector(15 downto 0) := (others => '1');
    signal clearbuf : std_logic_vector(7 downto 0);
    signal clrcount : unsigned(2 downto 0) := "111";

    signal hold_start : std_logic;
    signal state_indicator : unsigned(7 downto 0) := x"00";

    signal word_count : unsigned(7 downto 0);
begin
    sclk <= sclk_sig;
    ready <= '1' when state = cmd_done else '0';
    err <= '1' when state = cmd_err else '0';
    resp_debug <= response(7 downto 0);
    state_debug <= std_logic_vector(state_indicator);
    waiting <= '1' when state = wait_resp else '0';
    clk_enable <= '1' when clk_divider = "11" else '0';

    -- clock divider for sd clock
    process(clk50)
    begin
        if rising_edge(clk50) then
            clk_divider <= clk_divider + 1;

            -- hold start so that it is visible on next clock enable
            if start = '1' then
                hold_start <= '1';
            end if;

            if state /= cmd_done then
                hold_start <= '0';
            end if;

            if state = write_word and clk_enable = '1' then
                write_en <= '1';
            else
                write_en <= '0';
            end if;
        end if; -- rising_edge(clk50)
    end process;

    mosi <= command(47);
    cs <= '1' when state = reset_clks1 or
```

```
                 state = cmd_done or
                 state = clear_input or
                 state = cmd_err else '0';

process(clk50)
begin

if rising_edge(clk50) then
if clk_enable = '1' then

case state is

    -- asserting mosi and cs high for at least 74 clocks
    when reset_state =>
        command <= (others => '1');
        sclk_sig <= '0';
        counter <= to_unsigned(160, 8);
        state <= reset_clks1;

    when reset_clks1 =>
        if counter = x"00" then
            counter <= to_unsigned(32, 8);
            state <= reset_clks2;
        else
            counter <= counter - "1";
            sclk_sig <= not sclk_sig;
        end if;

    when reset_clks2 =>
        if counter = 0 then
            command <= cmd0;
            counter <= to_unsigned(47, 8);
            return_state <= check_cmd0;
            state <= send_cmd;
            state_indicator <= x"00";
        else
            counter <= counter - "1";
            sclk_sig <= not sclk_sig;
        end if;

    -- make sure reset was successful
    when check_cmd0 =>
        if response(7 downto 0) = x"01" then
            command <= cmd8;
            return_state <= check_cmd8_head;
            state <= clear_input;
            state_indicator <= x"08";
        else
            state_indicator <= x"00";
            state <= cmd_err;
        end if;

    -- make sure card supports v2 of protocol
    when check_cmd8_head =>
        if response(2) = '0' then
            counter <= to_unsigned(31, 8);
            state <= recv_resp;
            return_state <= check_cmd8_extra;
        else
            state <= cmd_err;
        end if;

    -- make sure voltage is OK
    when check_cmd8_extra =>
        if response(11 downto 0) = "000110101010" then
            command <= cmd55;
            state <= clear_input;
            return_state <= check_cmd55;
```

```
                state_indicator <= x"55";
            else
                response(15 downto 12) <= (others => '0');
                state <= cmd_err;
            end if;

        -- make sure cmd58 is OK, then check CCS
        when check_cmd58_head =>
            if response(2) = '1' then
                state_indicator <= x"58";
                state <= cmd_err;
            else
                counter <= to_unsigned(15, 8);
                return_state <= check_cmd58_ccs;
                state <= recv_resp;
            end if;

        -- is this standard or high capacity card?
        when check_cmd58_ccs =>
            ccs <= response(14);
            counter <= to_unsigned(15, 8);
            state <= recv_resp;
            return_state <= cmd_done;

        -- make sure application commands are OK
        when check_cmd55 =>
            if response(7 downto 0) = x"01" then
                command <= cmd41;
                state <= clear_input;
                return_state <= check_cmd41;
                state_indicator <= x"41";
            else
                state_indicator <= x"55";
                state <= cmd_err;
            end if;

        -- is SD card ready for I/O yet?
        when check_cmd41 =>
            if response(7 downto 0) = x"00" then
                -- if so, check what type of card this is
                command <= cmd58;
                return_state <= check_cmd58_head;
                state_indicator <= x"58";
                state <= clear_input;
            elsif response(7 downto 0) = x"01" then
                -- still not ready? read another byte of response
                state <= recv_resp;
                counter <= to_unsigned(7, 8);
                state_indicator <= x"41";
                return_state <= check_cmd41;
            elsif response(7 downto 0) = x"ff" then
                -- response has ended but we're still not ready, send ACMD41 again
                state <= clear_input;
                command <= cmd55;
                return_state <= check_cmd55;
                state_indicator <= x"55";
            else
                state_indicator <= x"41";
                state <= cmd_err;
            end if;

        -- send all 48 bits of the command
        when send_cmd =>
            if sclk_sig = '1' then
                if counter = x"00" then
                    state <= wait_resp;
                    counter <= to_unsigned(127, 8);
                else
                    counter <= counter - "1";
                    command <= command(46 downto 0) & "1";
                end if;
            end if;
            sclk_sig <= not sclk_sig;

        -- SD card could take a few clock cycles to respond a MISO
        when wait_resp =>
            if sclk_sig = '1' and miso = '0' then
                -- We've already gotten the first bit
                -- so only need to ready 7 more
                counter <= to_unsigned(6, 8);
                state <= recv_resp;
                response <= (others => '0');
            end if;
            sclk_sig <= not sclk_sig;

        -- Read bits from MISO
        when recv_resp =>
            if sclk_sig = '1' then
                response <= response(14 downto 0) & miso;
                if counter = 0 then
                    counter <= to_unsigned(7, 8);
                    state <= return_state;
                else
                    counter <= counter - "1";
                end if;
            end if;
            sclk_sig <= not sclk_sig;

        -- always deselect chip for 8 clock cycles and wait
        -- for miso to clear up before sending next command
        when clear_input =>
            if sclk_sig = '1' then
                clearbuf <= clearbuf(6 downto 0) & miso;
                if clrcount = 0 then
                    state <= check_clear;
                    clrcount <= "111";
                else
                    clrcount <= clrcount - "1";
                end if;
            end if;
            sclk_sig <= not sclk_sig;

        -- input is clear if we get byte of all 1s
        when check_clear =>
            if clearbuf = x"ff" then
                state <= send_cmd;
                counter <= to_unsigned(47, 8);
            else
                state <= clear_input;
            end if;

        -- the idle state
        when cmd_done =>
            if hold_start = '1' then
                -- send the read block command (cmd17)
                counter <= to_unsigned(47, 8);
                command <= x"51" & std_logic_vector(blockaddr) & x"ff";
                return_state <= check_cmd17;
                state <= clear_input;
                state_indicator <= x"17";
            end if;

        when check_cmd17 =>
            if response(7 downto 0) = x"00" then
                -- read command OK, wait for start byte
                counter <= to_unsigned(7, 8);
```

```
                return_state <= wait_block_start;
                state <= recv_resp;
                word_count <= x"00";
            else
                state <= cmd_err;
            end if;

        -- wait for beginning of block
        when wait_block_start =>
            -- block starts once we get the byte "fe"
            if response(7 downto 0) = x"fe" then
                counter <= to_unsigned(15, 8);
                return_state <= write_word;
                state <= recv_resp;
            else
                counter <= to_unsigned(7, 8);
                return_state <= wait_block_start;
                state <= recv_resp;
            end if;

        when write_word =>
            writedata <= signed(response);
            writeaddr <= word_count;

            -- if this is the last block
            if word_count = x"ff" then
                -- read the CRC (last 2 bytes) and ignore it
                counter <= to_unsigned(15, 8);
                return_state <= cmd_done;
                state <= recv_resp;
            else
                counter <= to_unsigned(15, 8);
                return_state <= write_word;
                state <= recv_resp;
                word_count <= word_count + 1;
            end if;

        when cmd_err =>
            sclk_sig <= sclk_sig;

    end case; -- state

    end if; -- clk_en = '1'
    end if; -- rising_edge(clk)

    end process;

end rtl;
```

sdbuf.vhd

```
-- sdbuf.vhd

-- This file was auto-generated as part of a generation operation.
-- If you edit it your changes will probably be lost.

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity sdbuf is
  port (
    clk         : in   std_logic                     := '0';             --          clock.clk
    reset_n     : in   std_logic                     := '0';             --          reset.
        reset_n
    read        : in   std_logic                     := '0';             -- avalon_slave_0.read
    chipselect : in  std_logic                     := '0';             --               .
        chipselect
    address    : in  std_logic_vector(7 downto 0)  := (others => '0'); --               .
        address
    readdata   : out std_logic_vector(15 downto 0);                    --               .
        readdata
    sdbuf_rden : out std_logic;                                        --    conduit_end.
        export
    sdbuf_addr : out std_logic_vector(7 downto 0);                     --               .
        export
    sdbuf_data : in  std_logic_vector(15 downto 0) := (others => '0')  --               .
        export
  );
end entity sdbuf;

architecture rtl of sdbuf is
  component de2_sd_buffer is
    port (
      clk        : in  std_logic                     := 'X';             -- clk
      reset_n    : in  std_logic                     := 'X';             -- reset_n
      read       : in  std_logic                     := 'X';             -- read
      chipselect : in  std_logic                     := 'X';             -- chipselect
      address    : in  std_logic_vector(7 downto 0)  := (others => 'X'); -- address
      readdata   : out std_logic_vector(15 downto 0);                    -- readdata
      sdbuf_rden : out std_logic;                                        -- export
      sdbuf_addr : out std_logic_vector(7 downto 0);                     -- export
      sdbuf_data : in  std_logic_vector(15 downto 0) := (others => 'X')  -- export
    );
  end component de2_sd_buffer;

begin

  sdbuf : component de2_sd_buffer
    port map (
      clk        => clk,        --          clock.clk
      reset_n    => reset_n,    --          reset.reset_n
      read       => read,       -- avalon_slave_0.read
      chipselect => chipselect, --               .chipselect
      address    => address,    --               .address
      readdata   => readdata,   --               .readdata
      sdbuf_rden => sdbuf_rden, --    conduit_end.export
      sdbuf_addr => sdbuf_addr, --               .export
      sdbuf_data => sdbuf_data  --               .export
    );

end architecture rtl; -- of sdbuf
```

sevenseg.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sevenseg is
    port(number  : in std_logic_vector(3 downto 0);
         display : out std_logic_vector(6 downto 0));
end sevenseg;

architecture rtl of sevenseg is
begin
    with number select
        display <= "1000000" when x"0",
                   "1111001" when x"1",
                   "0100100" when x"2",
                   "0110000" when x"3",
                   "0011001" when x"4",
                   "0010010" when x"5",
                   "0000010" when x"6",
                   "1111000" when x"7",
                   "0000000" when x"8",
                   "0011000" when x"9",
                   "0001000" when x"A",
                   "0000011" when x"b",
                   "1000110" when x"C",
                   "0100001" when x"d",
                   "0000110" when x"E",
                   "0001110" when x"F",
                   "1111111" when others;
end rtl;
```

visualizer.vhd

```
-------------------------------------------------------------------------------
--
-- Visualizer
-- draws rectangles of varying heights to correspond to
-- fft frequency bins and their respective amplitudes
--
-- for kanto music player
--
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity visualizer is

port (
    clk25   : in std_logic;                    -- Should be 25.125 MHz
    clk50 : in std_logic;
    reset_data: in std_logic;
    fft_fdom_addr : out unsigned(7 downto 0);
    fft_fdom_data : in signed(31 downto 0);

    ledr17 : out std_logic;
    ledr16 : out std_logic;
    ledr15 : out std_logic;

    sw_r : in std_logic;
    sw_g : in std_logic;
    sw_b : in std_logic;

    VGA_CLK,                          -- Clock
    VGA_HS,                           -- H_SYNC
    VGA_VS,                           -- V_SYNC
    VGA_BLANK,                        -- BLANK
    VGA_SYNC : out std_logic;         -- SYNC
    VGA_R,                            -- Red[9:0]
    VGA_G,                            -- Green[9:0]
    VGA_B : out std_logic_vector(9 downto 0); -- Blue[9:0]

    vga_text_buffer_x : out std_logic_vector(9 downto 0);
    vga_text_buffer_y : out std_logic_vector(6 downto 0);
    vga_text_buffer_pixel : in std_logic
);

end visualizer;

architecture rtl of visualizer is

  -- Video parameters
```

```
    constant HTOTAL       : integer := 800;
    constant HSYNC        : integer := 96;
    constant HBACK_PORCH  : integer := 48;
    constant HACTIVE      : integer := 640;
    constant HFRONT_PORCH : integer := 16;

    constant VTOTAL       : integer := 525;
    constant VSYNC        : integer := 2;
    constant VBACK_PORCH  : integer := 33;
    constant VACTIVE      : integer := 480;
    constant VFRONT_PORCH : integer := 10;

    constant bar_w : integer := 40;

    type states is (initializing, reading_data);

    -- Signals for the video controller
    signal Hcount : unsigned(9 downto 0);  -- Horizontal position (0-800)
    signal Vcount : unsigned(9 downto 0);  -- Vertical position (0-524)
    signal EndOfLine, EndOfField : std_logic;

    signal vga_hblank, vga_hsync,
      vga_vblank, vga_vsync : std_logic;  -- Sync. signals

    signal rectangle : std_logic;  -- rectangle area

    type ram_type is array (0 to 15) of unsigned(19 downto 0);

    signal current_sum : ram_type := ((others=>(others =>'0')));
    signal next_sum : ram_type := ((others=>(others =>'0')));

    signal address_r      : integer := 512;
    signal index          : integer := 0;
    signal sram_base      : integer := 0;
    signal counter  : integer := 0;
    signal addr_counter   : unsigned(7 downto 0)  := x"00";
    signal sum_counter    : unsigned(4 downto 0)  := "00000";
    signal test_ones      : unsigned (15 downto 0)  := "1111111111111111";
    signal test_zeros     : std_logic_vector (15 downto 0)  := "0000111111111111";
    signal test_half      : std_logic_vector (15 downto 0)  := "0111111111111111";
    signal oldsum : unsigned(19 downto 0);
    signal last_fdom_data : signed(15 downto 0);

    signal hpos : integer range -144 to 800;
    signal vpos : integer range -10 to 525;

    -- reset stuff
    signal reset          : std_logic := '0'; -- resets the screen

  begin

    -- Horizontal and vertical counters

    fft_fdom_addr <= addr_counter;
    oldsum <= next_sum(to_integer(sum_counter(4 downto 1)));

    GetData : process (clk50)
    variable state : states := initializing;
    begin
    if rising_edge(clk50) then
      case state is
          when initializing =>
        if reset_data = '1' then
                  next_sum <= ((others=>(others =>'0')));
                  last_fdom_data <= (others => '0');
                  sum_counter <= (others => '0');
                  addr_counter <= (others => '0');
                  reset <= '0';
```

```
              state := reading_data;
          else
              state:= initializing;
                    reset<='0';
          end if;
            when reading_data =>
                -- 4 downto 1 - we're grouping the 32 frequency bins into sets of
                -- two. so when summing, we can ignore the LSB and this happens
                -- on its own

                -- also ensure that we're only only adding positive number (absolute value)
                if last_fdom_data(15) = '1' then
                         next_sum(to_integer(sum_counter(4 downto 1))) <= oldsum
                                + unsigned(not last_fdom_data(14 downto 0));
                else
                    next_sum(to_integer(sum_counter(4 downto 1))) <= oldsum
                                + unsigned(last_fdom_data(14 downto 0));
                end if;
                if sum_counter = x"1F" then -- count up to 31
                    addr_counter <= x"00";
                    sum_counter <= "00000";
                    state := initializing;
                    ledr15 <= '0';
                    ledr16 <= '1';
                    ledr17 <= '0';
                else -- if we haven't yet reached 31
                    ledr15 <= '1';
                    ledr16 <= '0';
                    ledr17 <= '1';
                    addr_counter  <= addr_counter + 1;
                    sum_counter <= addr_counter(4 downto 0);
                    last_fdom_data <= fft_fdom_data(31 downto 16);

                    state := reading_data;

                end if;
          end case;
        end if;
      --end if;
    end process GetData;

      -- Horizontal and vertical counters

      HCounter : process (clk25)
      begin
        if rising_edge(clk25) then
          if reset = '1' then
            Hcount <= (others => '0');
          elsif EndOfLine = '1' then
            Hcount <= (others => '0');
          else
            Hcount <= Hcount + 1;
          end if;
        end if;
      end process HCounter;

      EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

      VCounter: process (clk25)
      begin
        if rising_edge(clk25) then
          if reset = '1' then
            Vcount <= (others => '0');
          elsif EndOfLine = '1' then
            if EndOfField = '1' then
              Vcount <= (others => '0');
            else
              Vcount <= Vcount + 1;
            end if;
          end if;
        end if;
      end process VCounter;

      EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

      -- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK

      HSyncGen : process (clk25)
      begin
        if rising_edge(clk25) then
          if reset = '1' or EndOfLine = '1' then
            vga_hsync <= '1';
          elsif Hcount = HSYNC - 1 then
            vga_hsync <= '0';
          end if;
        end if;
      end process HSyncGen;

      HBlankGen : process (clk25)
      begin
        if rising_edge(clk25) then
          if reset = '1' then
            vga_hblank <= '1';
          elsif Hcount = HSYNC + HBACK_PORCH then
            vga_hblank <= '0';
          elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
            vga_hblank <= '1';
          end if;
        end if;
      end process HBlankGen;

      VSyncGen : process (clk25)
      begin
        if rising_edge(clk25) then
          if reset = '1' then
            vga_vsync <= '1';
          elsif EndOfLine ='1' then
            if EndOfField = '1' then
              vga_vsync <= '1';
            elsif Vcount = VSYNC - 1 then
              vga_vsync <= '0';
            end if;
          end if;
        end if;
      end process VSyncGen;

      VBlankGen : process (clk25)
      begin
        if rising_edge(clk25) then
          if reset = '1' then
            vga_vblank <= '1';
          elsif EndOfLine = '1' then
            if Vcount = VSYNC + VBACK_PORCH - 1 then
              vga_vblank <= '0';
            elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
              vga_vblank <= '1';
            end if;
          end if;
        end if;
      end process VBlankGen;

    -- continuously get the pixel from the vga text buffer
    -- doesn't matter if we're out of range - we only use the
    -- vga_text_buffer_pixel value when we're in range, anyway
    hpos <= to_integer(Hcount) - (HSYNC + HBACK_PORCH);
```

```
vpos <= VTOTAL - VFRONT_PORCH - to_integer(Vcount);
vga_text_buffer_x <= std_logic_vector(to_unsigned(hpos, 10));
vga_text_buffer_y <= std_logic_vector(to_unsigned(to_integer(Vcount) - (VSYNC + VBACK_PORCH),
    7));

RectangleGen: process (clk25)
    variable height : unsigned(7 downto 0);
    variable sum_index : integer range 0 to 15;
begin
  if rising_edge(clk25) then
    if reset='1' then
      rectangle<='0';
        -- if we're inside the top 5 lines, allow the
        -- vga text buffer to handle output
        elsif vpos < 480 and vpos >= 400 then
            rectangle <= vga_text_buffer_pixel;
        -- is it inside the drawable region
        elsif hpos >= 0 and hpos <= 16 * bar_w then
            if hpos <= bar_w then
                sum_index := 0;
            elsif hpos <= 2 * bar_w then
                sum_index := 1;
            elsif hpos <= 3 * bar_w then
                sum_index := 2;
            elsif hpos <= 4 * bar_w then
                sum_index := 3;
            elsif hpos <= 5 * bar_w then
                sum_index := 4;
            elsif hpos <= 6 * bar_w then
                sum_index := 5;
            elsif hpos <= 7 * bar_w then
                sum_index := 6;
            elsif hpos <= 8 * bar_w then
                sum_index := 7;
            elsif hpos <= 9 * bar_w then
                sum_index := 8;
            elsif hpos <= 10 * bar_w then
                sum_index := 9;
            elsif hpos <= 11 * bar_w then
                sum_index := 10;
            elsif hpos <= 12 * bar_w then
                sum_index := 11;
            elsif hpos <= 13 * bar_w then
                sum_index := 12;
            elsif hpos <= 14 * bar_w then
                sum_index := 13;
            elsif hpos <= 15 * bar_w then
                sum_index := 14;
            else
                sum_index := 15;
            end if;

            height := current_sum(sum_index)(8 downto 1);

            if vpos < height then
                rectangle <= '1';
            else
                rectangle <= '0';
            end if;
    else
      rectangle<='0';
    end if;

        if vga_hblank = '1' and vga_vblank = '1' then
            current_sum <= next_sum;
        end if;
  end if;
end process RectangleGen;
```

```
  -- Registered video signals going to the video DAC

  VideoOut: process (clk25, reset)
  begin
    if reset = '1' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    elsif clk25'event and clk25 = '1' then
      if rectangle = '1' then
        if sw_r = '1' then
            VGA_R <= "0000000000";
         else VGA_R <= "1111111111";
         end if;

        if sw_g = '1' then
           VGA_G <= "0000000000";
        else VGA_G <= "1111111111";
        end if;

        if sw_b = '1' then
           VGA_B <= "0000000000";
        else
           VGA_B <= "1111111111";
        end if;
    elsif vga_hblank = '0' and vga_vblank ='0' then
        VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
    else
        VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
    end if;
  end if;
  end process VideoOut;

  VGA_CLK <= clk25;
  VGA_HS <= not vga_hsync;
  VGA_VS <= not vga_vsync;
  VGA_SYNC <= '0';
  VGA_BLANK <= not (vga_hsync or vga_vsync);

end rtl;
```

A.2 Verilog

de2_i2c_av_config.v

```
/*
 * I2C bus control for initializing the audio and video chips on the DE2 board
 *
 * Adapted by Stephen A. Edwards, Columbia University, sedwards@cs.columbia.edu
 */

module de2_i2c_av_config( iCLK, iRST_N, I2C_SCLK, I2C_SDAT );

// From host

input   iCLK;
input   iRST_N;

// I2C bus
```

```
output  I2C_SCLK;
inout   I2C_SDAT;

// Internal Registers/Wires
reg     [15:0]  mI2C_CLK_DIV;
reg     [23:0]  mI2C_DATA;
reg             mI2C_CTRL_CLK;
reg             mI2C_GO;
wire            mI2C_END;
wire            mI2C_ACK;
reg     [15:0]  LUT_DATA;
reg     [5:0]   LUT_INDEX;
reg     [3:0]   mSetup_ST;

// Clock frequencies
parameter CLK_Freq      = 50000000; // 50 MHz
parameter I2C_Freq      = 20000;    // 20 kHz

parameter LUT_SIZE      = 50;

// Audio Data Index
parameter SET_LIN_L     = 0;
parameter SET_LIN_R     = 1;
parameter SET_HEAD_L    = 2;
parameter SET_HEAD_R    = 3;
parameter A_PATH_CTRL   = 4;
parameter D_PATH_CTRL   = 5;
parameter POWER_ON      = 6;
parameter SET_FORMAT    = 7;
parameter SAMPLE_CTRL   = 8;
parameter SET_ACTIVE    = 9;
// Video Data Index
parameter SET_VIDEO     = 10;

// I2C Control Clock
always @(posedge iCLK or negedge iRST_N)
begin
  if (!iRST_N)
    begin
      mI2C_CTRL_CLK <= 0;
      mI2C_CLK_DIV  <= 0;
    end
  else
    begin
     if ( mI2C_CLK_DIV < (CLK_Freq/I2C_Freq) )
        mI2C_CLK_DIV <= mI2C_CLK_DIV+1;
     else
       begin
         mI2C_CLK_DIV  <= 0;
         mI2C_CTRL_CLK <= ~mI2C_CTRL_CLK;
       end
    end
end


de2_i2c_controller u0 (
  .CLOCK(mI2C_CTRL_CLK), //  Controller Work Clock
  .I2C_SCLK(I2C_SCLK),   //  I2C CLOCK
  .I2C_SDAT(I2C_SDAT),   //  I2C DATA
  .I2C_DATA(mI2C_DATA),  //  DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
  .GO(mI2C_GO),          //  GO transfor
  .END(mI2C_END),        //  END transfor
  .ACK(mI2C_ACK),        //  ACK
  .RESET(iRST_N)
);

// Configuration control
always @(posedge mI2C_CTRL_CLK or negedge iRST_N)
```

```
begin
  if(!iRST_N)
  begin
    LUT_INDEX <= 0;
    mSetup_ST <= 0;
    mI2C_GO   <= 0;
  end
  else
  begin
    if (LUT_INDEX<LUT_SIZE)
    begin
      case(mSetup_ST)
      0:  begin
            if(LUT_INDEX<SET_VIDEO)
              mI2C_DATA <= {8'h34,LUT_DATA};
            else
              mI2C_DATA <= {8'h40,LUT_DATA};
            mI2C_GO   <= 1;
            mSetup_ST <= 1;
          end
      1:  begin
            if (mI2C_END)
              begin
                if (!mI2C_ACK)
                  mSetup_ST <= 2;
                else
                  mSetup_ST <= 0;
              mI2C_GO <= 0;
            end
          end
      2:  begin
            LUT_INDEX <= LUT_INDEX+1;
            mSetup_ST <= 0;
          end
      endcase
    end
  end
end

// Configuration data LUT
always
begin
  case (LUT_INDEX)
  // Audio Config Data
  SET_LIN_L     : LUT_DATA <= 16'h0079;
  SET_LIN_R     : LUT_DATA <= 16'h0279;
  SET_HEAD_L    : LUT_DATA <= 16'h047B;
  SET_HEAD_R    : LUT_DATA <= 16'h067B;
  A_PATH_CTRL   : LUT_DATA <= 16'h08F8;
  D_PATH_CTRL   : LUT_DATA <= 16'h0A04;
  POWER_ON      : LUT_DATA <= 16'h0C00;
  SET_FORMAT    : LUT_DATA <= 16'h0E01;
  SAMPLE_CTRL   : LUT_DATA <= 16'h1020;
  SET_ACTIVE    : LUT_DATA <= 16'h1201;
  // Video Config Data
  SET_VIDEO+0   : LUT_DATA <= 16'h1500;
  SET_VIDEO+1   : LUT_DATA <= 16'h1741;
  SET_VIDEO+2   : LUT_DATA <= 16'h3a16;
  SET_VIDEO+3   : LUT_DATA <= 16'h5004;
  SET_VIDEO+4   : LUT_DATA <= 16'hc305;
  SET_VIDEO+5   : LUT_DATA <= 16'hc480;
  SET_VIDEO+6   : LUT_DATA <= 16'h0e80;
  SET_VIDEO+7   : LUT_DATA <= 16'h5020;
  SET_VIDEO+8   : LUT_DATA <= 16'h5218;
  SET_VIDEO+9   : LUT_DATA <= 16'h58ed;
  SET_VIDEO+10  : LUT_DATA <= 16'h77c5;
  SET_VIDEO+11  : LUT_DATA <= 16'h7c93;
  SET_VIDEO+12  : LUT_DATA <= 16'h7d00;
```

```
    SET_VIDEO+13 : LUT_DATA <= 16'hd048;
    SET_VIDEO+14 : LUT_DATA <= 16'hd5a0;
    SET_VIDEO+15 : LUT_DATA <= 16'hd7ea;
    SET_VIDEO+16 : LUT_DATA <= 16'he43e;
    SET_VIDEO+17 : LUT_DATA <= 16'hea0f;
    SET_VIDEO+18 : LUT_DATA <= 16'h3112;
    SET_VIDEO+19 : LUT_DATA <= 16'h3281;
    SET_VIDEO+20 : LUT_DATA <= 16'h3384;
    SET_VIDEO+21 : LUT_DATA <= 16'h37A0;
    SET_VIDEO+22 : LUT_DATA <= 16'he580;
    SET_VIDEO+23 : LUT_DATA <= 16'he603;
    SET_VIDEO+24 : LUT_DATA <= 16'he785;
    SET_VIDEO+25 : LUT_DATA <= 16'h5000;
    SET_VIDEO+26 : LUT_DATA <= 16'h5100;
    SET_VIDEO+27 : LUT_DATA <= 16'h0050;
    SET_VIDEO+28 : LUT_DATA <= 16'h1000;
    SET_VIDEO+29 : LUT_DATA <= 16'h0402;
    SET_VIDEO+30 : LUT_DATA <= 16'h0b00;
    SET_VIDEO+31 : LUT_DATA <= 16'h0a20;
    SET_VIDEO+32 : LUT_DATA <= 16'h1100;
    SET_VIDEO+33 : LUT_DATA <= 16'h2b00;
    SET_VIDEO+34 : LUT_DATA <= 16'h2c8c;
    SET_VIDEO+35 : LUT_DATA <= 16'h2df2;
    SET_VIDEO+36 : LUT_DATA <= 16'h2eee;
    SET_VIDEO+37 : LUT_DATA <= 16'h2ff4;
    SET_VIDEO+38 : LUT_DATA <= 16'h30d2;
    SET_VIDEO+39 : LUT_DATA <= 16'h0e05;
    default      : LUT_DATA <= 16'hxxxx;
    endcase
  end

  endmodule
```

de2_i2c_controller.v

```
  // --------------------------------------------------------------------
  // Copyright (c) 2005 by Terasic Technologies Inc.
  // --------------------------------------------------------------------
  //
  // Permission:
  //
  //   Terasic grants permission to use and modify this code for use
  //   in synthesis for all Terasic Development Boards and Altrea Development
  //   Kits made by Terasic.  Other use of this code, including the selling,
  //   duplication, or modification of any portion is strictly prohibited.
  //
  // Disclaimer:
  //
  //   This VHDL or Verilog source code is intended as a design reference
  //   which illustrates how these types of functions can be implemented.
  //   It is the user's responsibility to verify their design for
  //   consistency and functionality through the use of formal
  //   verification methods.  Terasic provides no warranty regarding the use
  //   or functionality of this code.
  //
  // --------------------------------------------------------------------
  //
  //                     Terasic Technologies Inc
  //                     356 Fu-Shin E. Rd Sec. 1. JhuBei City,
  //                     HsinChu County, Taiwan
  //                     302
  //
  //                     web: http://www.terasic.com/
  //                     email: support@terasic.com
  //
  // --------------------------------------------------------------------
```

```
//
// Major Functions:i2c controller
//
// --------------------------------------------------------------------
//
// Revision History :
// --------------------------------------------------------------------
//   Ver  :| Author            :| Mod. Date :| Changes Made:
//   V1.0 :| Joe Yang          :| 05/07/10  :|      Initial Revision
// --------------------------------------------------------------------
module de2_i2c_controller (
  CLOCK,
  I2C_SCLK, // I2C CLOCK
  I2C_SDAT, // I2C DATA
  I2C_DATA, // DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
  GO,       // GO transfor
  END,      // END transfor
  W_R,      // W_R
  ACK,      // ACK
  RESET,
  // TEST
  SD_COUNTER,
  SDO
);

input  CLOCK;
input  [23:0] I2C_DATA;
input  GO;
input  RESET;
input  W_R;
inout  I2C_SDAT;
output I2C_SCLK;
output END;
output ACK;

// TEST
output [5:0] SD_COUNTER;
output SDO;

reg SDO;
reg SCLK;
reg END;
reg [23:0] SD;
reg [5:0] SD_COUNTER;

wire I2C_SCLK = SCLK | (((SD_COUNTER >= 4) & (SD_COUNTER <= 30))? ~CLOCK : 0);
wire I2C_SDAT = SDO ? 1'bz : 0;

reg ACK1, ACK2, ACK3;
wire ACK = ACK1 | ACK2 | ACK3;

//--I2C COUNTER

always @(negedge RESET or posedge CLOCK)
begin
  if (!RESET)
    SD_COUNTER = 6'b111111;
  else
    begin
      if (GO == 0)
        SD_COUNTER = 0;
      else
        if (SD_COUNTER < 6'b111111)
          SD_COUNTER = SD_COUNTER + 1;
    end
end

always @(negedge RESET or posedge CLOCK )
```

```
begin
  if (!RESET)
    begin
      SCLK = 1;
      SDO = 1;
      ACK1 = 0;
      ACK2 = 0;
      ACK3 = 0;
      END = 1;
    end
else
  case (SD_COUNTER)
  6'd0  : begin ACK1 = 0; ACK2 = 0; ACK3 = 0; END = 0; SDO = 1; SCLK = 1; end

  // Start
  6'd1  : begin SD = I2C_DATA; SDO = 0; end
  6'd2  : SCLK = 0;

  // Slave Address
  6'd3  : SDO = SD[23];
  6'd4  : SDO = SD[22];
  6'd5  : SDO = SD[21];
  6'd6  : SDO = SD[20];
  6'd7  : SDO = SD[19];
  6'd8  : SDO = SD[18];
  6'd9  : SDO = SD[17];
  6'd10 : SDO = SD[16];
  6'd11 : SDO = 1'b1;   //ACK

  // Sub-address
  6'd12  : begin SDO = SD[15]; ACK1 = I2C_SDAT; end
  6'd13  : SDO = SD[14];
  6'd14  : SDO = SD[13];
  6'd15  : SDO = SD[12];
  6'd16  : SDO = SD[11];
  6'd17  : SDO = SD[10];
  6'd18  : SDO = SD[9];
  6'd19  : SDO = SD[8];
  6'd20  : SDO = 1'b1; // ACK

  // Data
  6'd21  : begin SDO = SD[7]; ACK2 = I2C_SDAT; end
  6'd22  : SDO = SD[6];
  6'd23  : SDO = SD[5];
  6'd24  : SDO = SD[4];
  6'd25  : SDO = SD[3];
  6'd26  : SDO = SD[2];
  6'd27  : SDO = SD[1];
  6'd28  : SDO = SD[0];
  6'd29  : SDO = 1'b1;  // ACK

  // Stop
  6'd30 : begin SDO = 1'b0; SCLK = 1'b0; ACK3 = I2C_SDAT; end
  6'd31 : SCLK = 1'b1;
  6'd32 : begin SDO = 1'b1; END = 1; end
  endcase
end

endmodule
```

A.3 C

main.c

```
#include <stdio.h>
```

```c
#include <system.h>
#include <io.h>
#include <stdint.h>
#include "vga.h"

#define KANTO_BLOCKADDR 0
#define KANTO_READBLOCK 4
#define KANTO_PLAY 8
#define KANTO_DONE 12
#define KANTO_TRACK 16
#define KANTO_KEYS 20
#define KANTO_CCS 24

#define MAX_TRACKS 8

uint32_t track_table[MAX_TRACKS];
char track_titles[MAX_TRACKS][60];
int track_count = 0;
unsigned char curtrack;
uint32_t track_start;
uint32_t track_end;

unsigned char selected_track;
unsigned char selected_row;
unsigned char list_top_track;
char buffer[81];
int playing = 0;

#define NEXT_TRACK 0x1
#define LAST_TRACK 0x2
#define FAST_FORWARD 0x4
#define REWIND 0x8

/* number of blocks in a second */
static int BLOCK_SECOND;
static int SD_CCS;

#define wait_for_done() while (!IORD_8DIRECT(KANTO_CTRL_BASE, KANTO_DONE))

static inline uint32_t sdbuf_read_word(unsigned char offset)
{
  uint32_t word, upper, lower;

  upper = IORD_16DIRECT(SDBUF_BASE, 4 * offset) & 0xffff;
  lower = IORD_16DIRECT(SDBUF_BASE, 4 * offset + 2) & 0xffff;

  word = upper << 16 | lower;

  return word;
}

static inline void stop_playback(void)
{
  playing = 0;
  IOWR_8DIRECT(KANTO_CTRL_BASE, KANTO_PLAY, 0);
  wait_for_done();
  if (curtrack >= list_top_track && curtrack <= list_top_track + 3) {
    int listnum = curtrack - list_top_track;
    snprintf(buffer, sizeof(buffer), "%c  %c%u. %s", (curtrack == selected_track) ? '*' : ' ',
         ' ', curtrack, track_titles[curtrack]);
    vga_write_string(buffer, listnum + 1);
  }
}

static inline void start_playback(void)
{
  playing = 1;
  IOWR_8DIRECT(KANTO_CTRL_BASE, KANTO_PLAY, 1);
```

```
    if (curtrack >= list_top_track && curtrack <= list_top_track + 3) {
      int listnum = curtrack - list_top_track;
      snprintf(buffer, sizeof(buffer), "%c  %c%u. %s", (curtrack == selected_track) ? '*' : ' ',
           '>', curtrack, track_titles[curtrack]);
      vga_write_string(buffer, listnum + 1);
    }
  }

  static inline void read_block(uint32_t addr)
  {
    IOWR_32DIRECT(KANTO_CTRL_BASE, KANTO_BLOCKADDR, addr);
    // pulse the readblock signal
    IOWR_8DIRECT(KANTO_CTRL_BASE, KANTO_READBLOCK, 1);
    IOWR_8DIRECT(KANTO_CTRL_BASE, KANTO_READBLOCK, 0);
    wait_for_done();
  }

  static void setup_track_table(void)
  {
    int i, j;
    uint32_t word;

    for (i = 0; i < MAX_TRACKS; i++) {
      track_table[i] = sdbuf_read_word(i * 16);
      if (track_table[i] != 0)
        track_count++;
      for (j = 0; j < 15; j += 1) {
        word = sdbuf_read_word(i * 16 + 1 + j);
        track_titles[i][j * 4 + 0] = word >> 24 & 0xff;
        track_titles[i][j * 4 + 1] = word >> 16 & 0xff;
        track_titles[i][j * 4 + 2] = word >> 8 & 0xff;
        track_titles[i][j * 4 + 3] = word >> 0 & 0xff;
      }
      printf("%i.  %s\n", i, (char * ) &track_titles[i]);
    }
    track_count--;
    printf("Track count: %d\n", track_count);
  }

  static inline void check_curtrack(void)
  {
    if (curtrack >= track_count)
      curtrack = 0;
  }

  static inline void seek_to_track(int track)
  {
    curtrack = track;
    check_curtrack();
    track_start = track_table[curtrack];
    track_end = track_table[curtrack + 1];

    read_block(track_start);
    printf("Setting current track to %d\n", curtrack);
    IOWR_8DIRECT(KANTO_CTRL_BASE, KANTO_TRACK, curtrack);
  }

  static void selection_up()
  {
    if (selected_track == 0)
      return;
    if (selected_row == 0) {
      int x, i;
      list_top_track--;
      for (x = 1, i = --selected_track; i < track_count && x < 5; x++, i++) {
        snprintf(buffer, sizeof(buffer), "%c  %c%u. %s", (i == selected_track) ? '*' : ' ', (i
            == curtrack && playing) ? '>' : ' ', i, track_titles[i]);
        vga_write_string(buffer, x);
      }
      return;
    }
    vga_write_character(' ', 0, selected_row + 1);
    vga_write_character('*', 0, --selected_row + 1);
    selected_track--;
  }

  static void selection_down()
  {
    if (selected_track == track_count - 1)
      return;
    if (selected_row == 3) {
      int x, i;
      list_top_track++;
      for (x = 4, i = ++selected_track; i >= 0 && x >= 1; x--, i--) {
        snprintf(buffer, sizeof(buffer), "%c  %c%u. %s", (i == selected_track) ? '*' : ' ', (i
            == curtrack && playing) ? '>' : ' ', i, track_titles[i]);
        vga_write_string(buffer, x);
      }
      return;
    }
    vga_write_character(' ', 0, selected_row + 1);
    vga_write_character('*', 0, ++selected_row + 1);
    selected_track++;
  }

  static inline void selection_play()
  {
    stop_playback();
    seek_to_track(selected_track);
    start_playback();
  }

  int ignore_next_key = 0;

  static void key_receive(uint32_t blockaddr)
  {
    unsigned short key;
    key = IORD_8DIRECT(PS2_BASE, 4);

    if (ignore_next_key) {
      ignore_next_key = 0;
      return;
    }

    if (key == 0xf0) {
      ignore_next_key = 1;
      return;
    }

    switch (key) {

    case 0x31: // 'n' next track
      stop_playback();
      seek_to_track(curtrack + 1);
      start_playback();
      break;

    case 0x4d: // 'p' previous track
      stop_playback();
      if ((blockaddr - track_start) < 2 * BLOCK_SECOND)
        seek_to_track(curtrack - 1);
      else
        seek_to_track(curtrack);
      start_playback();
      break;
    case 0x2b: // 'f' fast forward
      stop_playback();
      if (track_end - blockaddr < 5 * BLOCK_SECOND)
        seek_to_track(curtrack + 1);
      else
        read_block(blockaddr + 5 * BLOCK_SECOND);
      start_playback();
      break;

    case 0x32: // 'b' rewind
      stop_playback();
      if (blockaddr - track_start < 5 * BLOCK_SECOND)
        seek_to_track(curtrack);
      else
        read_block(blockaddr - 5 * BLOCK_SECOND);
      start_playback();
      break;

    case 0x3b: // 'j' move down
      selection_down();
      break;
    case 0x42: // 'k' move up
      selection_up();
      break;
    case 0x5a: // 'enter' select
      selection_play();
      break;
    case 0x29: // 'space' play/pause
      if (playing)
        stop_playback();
      else
        start_playback();
      break;
    }
  }

  int main()
  {
    uint32_t blockaddr;
    int i;

    SD_CCS = IORD_8DIRECT(KANTO_CTRL_BASE, KANTO_CCS);
    BLOCK_SECOND = (SD_CCS) ? 172 : 172 * 512;

    printf("Hello, Kanto\n");
    vga_write_string("                            == [ Hello, Kanto ] ==       ", 0);

      // stop playback
      stop_playback();

      printf("Starting initialization\n");
      // read first (metadata) block
      read_block(0);
      setup_track_table();
      printf("Track table read\n");

      seek_to_track(0);

      selected_track = 0;
      selected_row = 0;
      list_top_track = 0;

      for (i = 0; i < track_count && i < 4; i++) {
        snprintf(buffer, sizeof(buffer), "%c   %u. %s", (i == selected_row) ? '*' : ' ', i,
            track_titles[i]);
        vga_write_string(buffer, i + 1);
      }
      printf("First block read\n");

      start_playback();

      printf("Playing audio\n");

      for (;;) {
        blockaddr = IORD_32DIRECT(KANTO_CTRL_BASE, KANTO_BLOCKADDR);

        if (IORD_8DIRECT(PS2_BASE, 0))
          key_receive(blockaddr);
        else if (blockaddr >= track_end)
          seek_to_track(curtrack + 1);
      }

    return 0;
  }
```

vga.h

```
/*
 * vga.h
 *
 *  Created on: May 13, 2013
 *      Author: jy2432
 */

#ifndef VGA_H_
#define VGA_H_

void vga_write_character(char c, unsigned int x, unsigned int y);

void vga_write_string(char *s, unsigned int y);


#endif /* VGA_H_ */
```

vga.c

```
#include <system.h>
#include <io.h>
#include <stdio.h>
#include <stdint.h>
#include "vga.h"

uint32_t font[] = {
  0x00007ec3, 0x9999f3e7, 0xe7ffe7e7, 0x7e000000,
  0x00000000, 0x0076dc00, 0x76dc0000, 0x00000000,
  0x00006ef8, 0xd8d8dcd8, 0xd8d8f86e, 0x00000000,
  0x00000000, 0x006edbdb, 0xdfd8db6e, 0x00000000,
  0x00000000, 0x10387cfe, 0x7c381000, 0x00000000,
  0x008888f8, 0x8888003e, 0x08080808, 0x00000000,
  0x00f880e0, 0x8080003e, 0x20382020, 0x00000000,
  0x00708880, 0x8870003c, 0x223c2422, 0x00000000,
  0x00808080, 0x80f8003e, 0x20382020, 0x00000000,
  0x11441144, 0x11441144, 0x11441144, 0x11441144,
  0x55aa55aa, 0x55aa55aa, 0x55aa55aa, 0x55aa55aa,
  0xdd77dd77, 0xdd77dd77, 0xdd77dd77, 0xdd77dd77,
  0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff,
  0x00000000, 0x000000ff, 0xffffffff, 0xffffffff,
  0xffffffff, 0xffffff00, 0x00000000, 0x00000000,
  0xf0f0f0f0, 0xf0f0f0f0, 0xf0f0f0f0, 0xf0f0f0f0,
  0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f,
  0x0088c8a8, 0x98880020, 0x2020203e, 0x00000000,
  0x00888850, 0x5020003e, 0x08080808, 0x00000000,
  0x00000000, 0x0e38e038, 0x0e00fe00, 0x00000000,
  0x00000000, 0xe0380e38, 0xe000fe00, 0x00000000,
  0x00000006, 0x0cfe1830, 0xfe60c000, 0x00000000,
  0x00000000, 0x061e7efe, 0x7e1e0600, 0x00000000,
  0x00000000, 0xc0f0fcfe, 0xfcf0c000, 0x00000000,
  0x0000183c, 0x7e181818, 0x18181818, 0x00000000,
  0x00001818, 0x18181818, 0x187e3c18, 0x00000000,
  0x00000000, 0x00180cfe, 0x0c180000, 0x00000000,
  0x00000000, 0x003060fe, 0x60300000, 0x00000000,
  0x0000183c, 0x7e181818, 0x187e3c18, 0x00000000,
  0x00000000, 0x00286cfe, 0x6c280000, 0x00000000,
  0x00000000, 0x063666fe, 0x60300000, 0x00000000,
  0x00000000, 0x0080fe6e, 0x6c6c6c6c, 0x00000000,
  0x00000000, 0x00000000, 0x00000000, 0x00000000,
  0x0000183c, 0x3c3c1818, 0x18001818, 0x00000000,
  0x00666666, 0x24000000, 0x00000000, 0x00000000,
  0x0000006c, 0x6cfe6c6c, 0x6cfe6c6c, 0x00000000,
  0x0010107c, 0xd6d0d07c, 0x1616d67c, 0x10100000,
  0x00000000, 0xc2c60c18, 0x3060c686, 0x00000000,
  0x0000386c, 0x6c3876dc, 0xcccccc76, 0x00000000,
  0x00181818, 0x30000000, 0x00000000, 0x00000000,
  0x00000c18, 0x30303030, 0x3030180c, 0x00000000,
  0x00003018, 0x0c0c0c0c, 0x0c0c1830, 0x00000000,
  0x00000000, 0x00663cff, 0x3c660000, 0x00000000,
  0x00000000, 0x0018187e, 0x18180000, 0x00000000,
  0x00000000, 0x00000000, 0x00181818, 0x30000000,
  0x00000000, 0x000000fe, 0x00000000, 0x00000000,
  0x00000000, 0x00000000, 0x00001818, 0x00000000,
  0x00000000, 0x00060c18, 0x3060c000, 0x00000000,
  0x00007cc6, 0xceced6d6, 0xe6e6c67c, 0x00000000,
  0x00001838, 0x78181818, 0x1818187e, 0x00000000,
  0x00007cc6, 0x060c1830, 0x60c0c6fe, 0x00000000,
  0x00007cc6, 0x06063c06, 0x0606c67c, 0x00000000,
  0x00000c1c, 0x3c6cccfe, 0x0c0c0c1e, 0x00000000,
  0x0000fec0, 0xc0c0fc06, 0x0606c67c, 0x00000000,
  0x00003860, 0xc0c0fcc6, 0xc6c6c67c, 0x00000000,
  0x0000fec6, 0x06060c18, 0x30303030, 0x00000000,
  0x00007cc6, 0xc6c67cc6, 0xc6c6c67c, 0x00000000,
  0x00007cc6, 0xc6c67e06, 0x06060c78, 0x00000000,
  0x00000000, 0x18180000, 0x00181800, 0x00000000,
  0x00000000, 0x18180000, 0x00181830, 0x00000000,
  0x00000006, 0x0c183060, 0x30180c06, 0x00000000,
  0x00000000, 0x00fe0000, 0xfe000000, 0x00000000,
  0x00000060, 0x30180c06, 0x0c183060, 0x00000000,
  0x00007cc6, 0xc60c1818, 0x18001818, 0x00000000,
  0x00007cc6, 0xc6c6dede, 0xdedcc07c, 0x00000000,
  0x00001038, 0x6cc6c6fe, 0xc6c6c6c6, 0x00000000,
  0x0000fc66, 0x66667c66, 0x666666fc, 0x00000000,
  0x00003c66, 0xc2c0c0c0, 0xc0c2663c, 0x00000000,
  0x0000f86c, 0x66666666, 0x66666cf8, 0x00000000,
  0x0000fe66, 0x62687868, 0x606266fe, 0x00000000,
  0x0000fe66, 0x62687868, 0x606060f0, 0x00000000,
  0x00003c66, 0xc2c0c0de, 0xc6c6663a, 0x00000000,
  0x0000c6c6, 0xc6c6fec6, 0xc6c6c6c6, 0x00000000,
  0x00003c18, 0x18181818, 0x1818183c, 0x00000000,
  0x00001e0c, 0x0c0c0c0c, 0xcccccc78, 0x00000000,
  0x0000e666, 0x666c7878, 0x6c6666e6, 0x00000000,
  0x0000f060, 0x60606060, 0x606266fe, 0x00000000,
  0x0000c6ee, 0xfefed6c6, 0xc6c6c6c6, 0x00000000,
  0x0000c6e6, 0xf6fedece, 0xc6c6c6c6, 0x00000000,
  0x00007cc6, 0xc6c6c6c6, 0xc6c6c67c, 0x00000000,
  0x0000fc66, 0x66667c60, 0x606060f0, 0x00000000,
  0x00007cc6, 0xc6c6c6c6, 0xc6d6de7c, 0x0c0e0000,
  0x0000fc66, 0x66667c6c, 0x666666e6, 0x00000000,
  0x00007cc6, 0xc660380c, 0x06c6c67c, 0x00000000,
  0x00007e7e, 0x5a181818, 0x1818183c, 0x00000000,
  0x0000c6c6, 0xc6c6c6c6, 0xc6c6c67c, 0x00000000,
  0x0000c6c6, 0xc6c6c6c6, 0xc66c3810, 0x00000000,
  0x0000c6c6, 0xc6c6d6d6, 0xd6feee6c, 0x00000000,
  0x0000c6c6, 0x6c7c3838, 0x7c6cc6c6, 0x00000000,
  0x00006666, 0x66663c18, 0x1818183c, 0x00000000,
  0x0000fec6, 0x860c1830, 0x60c2c6fe, 0x00000000,
  0x00003c30, 0x30303030, 0x3030303c, 0x00000000,
  0x00000000, 0x00c06030, 0x180c0600, 0x00000000,
  0x00003c0c, 0x0c0c0c0c, 0x0c0c0c3c, 0x00000000,
  0x10386cc6, 0x00000000, 0x00000000, 0x00000000,
  0x00000000, 0x00000000, 0x00000000, 0x0000ff00,
  0x00303030, 0x18000000, 0x00000000, 0x00000000,
  0x00000000, 0x00780c7c, 0xcccccc76, 0x00000000,
  0x0000e060, 0x60786c66, 0x6666667c, 0x00000000,
  0x00000000, 0x007cc6c0, 0xc0c0c67c, 0x00000000,
  0x00001c0c, 0x0c3c6ccc, 0xcccccc76, 0x00000000,
  0x00000000, 0x007cc6fe, 0xc0c0c67c, 0x00000000,
  0x0000386c, 0x6460f060, 0x606060f0, 0x00000000,
  0x00000000, 0x0076cccc, 0xcccccc7c, 0x0ccc7800,
  0x0000e060, 0x606c7666, 0x666666e6, 0x00000000,
  0x00001818, 0x00381818, 0x1818183c, 0x00000000,
  0x00000606, 0x000e0606, 0x06060606, 0x66663c00,
  0x0000e060, 0x60666c78, 0x786c66e6, 0x00000000,
  0x00007030, 0x30303030, 0x30303418, 0x00000000,
  0x00000000, 0x00ecfed6, 0xd6d6d6c6, 0x00000000,
  0x00000000, 0x00dc6666, 0x66666666, 0x00000000,
  0x00000000, 0x007cc6c6, 0xc6c6c67c, 0x00000000,
  0x00000000, 0x00dc6666, 0x6666667c, 0x6060f000,
  0x00000000, 0x0076cccc, 0xcccccc7c, 0x0c0c1e00,
  0x00000000, 0x00dc7666, 0x606060f0, 0x00000000,
  0x00000000, 0x007cc660, 0x380cc67c, 0x00000000,
  0x00001030, 0x30fc3030, 0x3030361c, 0x00000000,
  0x00000000, 0x00cccccc, 0xcccccc76, 0x00000000,
  0x00000000, 0x00666666, 0x66663c18, 0x00000000,
  0x00000000, 0x00c6c6d6, 0xd6d6fe6c, 0x00000000,
  0x00000000, 0x00c66c38, 0x38386cc6, 0x00000000,
  0x00000000, 0x00c6c6c6, 0xc6c6c67e, 0x060cf800,
  0x00000000, 0x00fecc18, 0x3060c6fe, 0x00000000,
  0x00000e18, 0x18187018, 0x1818180e, 0x00000000,
  0x00001818, 0x18181818, 0x18181818, 0x00000000,
  0x00007018, 0x18180e18, 0x18181870, 0x00000000,
  0x000076dc, 0x00000000, 0x00000000, 0x00000000,
  0x00660066, 0x6666663c, 0x1818183c, 0x00000000,
};


void vga_write_character(char c, unsigned int x, unsigned int y)
{
  // clear out the top bit
  c &= 0x7f;

  int j;
  for (j = 0; j < 4; j++) {
    uint32_t symbol = font[4 * c + j];

    IOWR_32DIRECT(VGA_BASE, y * 80 * 16 + x * 16 + j * 4, symbol);
  }
}

void vga_write_string(char *s, unsigned int y)
{
  unsigned int x = 0;
  if (!s) return;
  while (*s)
    vga_write_character(*s++, x++, y);
  for (; x < 80; x++)
    vga_write_character(' ', x, y);
}
```

fft/comp2real.c

```
#include <stdio.h>
#include <stdlib.h>

/* Takes complex number pairs line by line on standard input,
 * combines them using the formula num = (real << 16) | imag
 * and then writes them out to standard output */

int main(void)
{
  int real, imag, comb;

  while (fscanf(stdin, "%d %d\n", &real, &imag) == 2) {
    real &= 0xffff;
    imag &= 0xffff;
    comb = (real << 16) | imag;
    printf("%d\n", comb);
  }

  return 0;
}
```

fft/dftcoeffgen.c

```
#include <stdlib.h>
#include <stdio.h>
#include <complex.h>
#include <math.h>
#include <limits.h>
#include <stdint.h>

#define N 16

int main(void)
{
  int16_t coreal, coimag;
  int n, k;
  float complex coeff;

  for (k = 0; k < N; k++) {
    for (n = 0; n < N; n++) {
      coeff = SHRT_MAX * cexp(2 * M_PI * I * k * n / N);
      coreal = (int16_t) creal(coeff);
      coimag = (int16_t) cimag(coeff);

      printf("%d %d\n", coreal, coimag);
    }
  }

  return 0;
}
```

fft/dftsim.c

```
#include <stdlib.h>
#include <stdio.h>
#include <complex.h>
#include <math.h>
```

```
#include <limits.h>
#include <stdint.h>

#define N 16

/* Simulates the DFT stage of the FFT computation
 * Takes 16 numbers line by line on standard input
 * Produces 16 complex number pairs on standard output */

int main(void)
{
  int16_t tdom[N];
  int16_t fdom[N][2];
  float complex coeff;
  int32_t sumreal, sumimag, multreal, multimag;
  int16_t coreal, coimag;
  int n, k;

  for (n = 0; n < N; n++)
    fscanf(stdin, "%hd\n", &tdom[n]);

  for (k = 0; k < N; k++) {
    sumreal = 0;
    sumimag = 0;

    for (n = 0; n < N; n++) {
      coeff = SHRT_MAX * cexp(2 * M_PI * I * k * n / N);
      coreal = (int16_t) creal(coeff);
      coimag = (int16_t) cimag(coeff);
      multreal = coreal * tdom[n];
      multimag = coimag * tdom[n];
      sumreal += (multreal >> 4);
      sumimag += (multimag >> 4);
    }

    fdom[k][0] = sumreal >> 16;
    fdom[k][1] = sumimag >> 16;

    printf("%d %d\n", fdom[k][0], fdom[k][1]);
  }

  return 0;
}
```

fft/rccoeffgen.c

```
#include <stdio.h>
#include <complex.h>
#include <stdlib.h>
#include <limits.h>
#include <math.h>

int main(int argc, char *argv[])
{
  int k, n;
  float coeff;
  short real, imag;

  if (argc < 2) {
    fprintf(stderr, "Usage: %s N\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  n = atoi(argv[1]);

  for (k = 0; k < n / 2; k++) {
    coeff = SHRT_MAX * cexp(-2 * M_PI * I * k / n);
    real = creal(coeff);
    imag = cimag(coeff);
    printf("%d %d\n", real, imag);
  }

  return 0;
}
```

fft/recombsim.c

```
#include <stdio.h>
#include <complex.h>
#include <stdlib.h>
#include <stdint.h>
#include <limits.h>
#include <math.h>

/* Simulates the recombination stage of the FFT computation
 * Takes n / 2 complex number pairs on standard input
 * prints out n complex number pairs on output */

int main(int argc, char *argv[])
{
  int k, n;
  float coeff;
  int16_t coreal, coimag;
  int16_t evenreal, evenimag;
  int16_t oddreal, oddimag;
  int32_t multreal, multimag;
  int16_t *resreal, *resimag;

  if (argc < 2) {
    fprintf(stderr, "Usage: %s N\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  n = atoi(argv[1]);

  resreal = malloc(sizeof(int16_t) * n);
  resimag = malloc(sizeof(int16_t) * n);

  if (resreal == NULL || resimag == NULL) {
    fprintf(stderr, "Could not allocated memory.\n");
    return EXIT_FAILURE;
  }

  for (k = 0; k < n / 2; k++) {
    fscanf(stdin, "%hd %hd %hd %hd\n", &evenreal, &evenimag,
                    &oddreal, &oddimag);
    coeff = SHRT_MAX * cexp(-2 * M_PI * I * k / n);
    coreal = creal(coeff);
    coimag = cimag(coeff);
    multreal = oddreal * coreal - oddimag * coimag;
    multimag = oddreal * coimag + oddimag * coreal;
    evenreal >>= 1;
    evenimag >>= 1;
    multreal >>= 17;
    multimag >>= 17;
    resreal[k] = evenreal + multreal;
    resimag[k] = evenimag + multimag;
    resreal[k + n / 2] = evenreal - multreal;
    resimag[k + n / 2] = evenimag - multimag;
  }

  for (k = 0; k < n; k++) {
    printf("%d %d\n", resreal[k], resimag[k]);
  }

  return 0;
}
```

fft/sumbins.c

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i, n, res;
  unsigned int sum = 0;
  short real;

  if (argc < 2) {
    fprintf(stderr, "Usage: sumbins n\n");
    exit(EXIT_FAILURE);
  }

  n = atoi(argv[1]);

  for (i = 0; ; i++) {
    res = fscanf(stdin, "%hd\n", &real);

    if (res == 0 || res == EOF)
      break;
    else if (res < 0) {
      perror("fscanf");
      exit(EXIT_FAILURE);
    }

    if (i % n == 0 && i != 0) {
      printf("%d\n", sum);
      sum = 0;
    }

    if (real < 0)
      sum -= real;
    else
      sum += real;
  }

  printf("%d\n", sum);

  return 0;
}
```

sdcard/crc7_calc.c

```
#include <stdlib.h>
#include <stdio.h>

/* CRC7 Calculator for SD card SPI protocol
 * Based off of the implementation in the Linux Kernel */

typedef unsigned char u8;

const u8 crc7_syndrome_table[256] = {
          0x00, 0x09, 0x12, 0x1b, 0x24, 0x2d, 0x36, 0x3f,
          0x48, 0x41, 0x5a, 0x53, 0x6c, 0x65, 0x7e, 0x77,
          0x19, 0x10, 0x0b, 0x02, 0x3d, 0x34, 0x2f, 0x26,
          0x51, 0x58, 0x43, 0x4a, 0x75, 0x7c, 0x67, 0x6e,
          0x32, 0x3b, 0x20, 0x29, 0x16, 0x1f, 0x04, 0x0d,
          0x7a, 0x73, 0x68, 0x61, 0x5e, 0x57, 0x4c, 0x45,
          0x2b, 0x22, 0x39, 0x30, 0x0f, 0x06, 0x1d, 0x14,
          0x63, 0x6a, 0x71, 0x78, 0x47, 0x4e, 0x55, 0x5c,
          0x64, 0x6d, 0x76, 0x7f, 0x40, 0x49, 0x52, 0x5b,
          0x2c, 0x25, 0x3e, 0x37, 0x08, 0x01, 0x1a, 0x13,
          0x7d, 0x74, 0x6f, 0x66, 0x59, 0x50, 0x4b, 0x42,
          0x35, 0x3c, 0x27, 0x2e, 0x11, 0x18, 0x03, 0x0a,
          0x56, 0x5f, 0x44, 0x4d, 0x72, 0x7b, 0x60, 0x69,
          0x1e, 0x17, 0x0c, 0x05, 0x3a, 0x33, 0x28, 0x21,
          0x4f, 0x46, 0x5d, 0x54, 0x6b, 0x62, 0x79, 0x70,
          0x07, 0x0e, 0x15, 0x1c, 0x23, 0x2a, 0x31, 0x38,
          0x41, 0x48, 0x53, 0x5a, 0x65, 0x6c, 0x77, 0x7e,
          0x09, 0x00, 0x1b, 0x12, 0x2d, 0x24, 0x3f, 0x36,
          0x58, 0x51, 0x4a, 0x43, 0x7c, 0x75, 0x6e, 0x67,
          0x10, 0x19, 0x02, 0x0b, 0x34, 0x3d, 0x26, 0x2f,
          0x73, 0x7a, 0x61, 0x68, 0x57, 0x5e, 0x45, 0x4c,
          0x3b, 0x32, 0x29, 0x20, 0x1f, 0x16, 0x0d, 0x04,
          0x6a, 0x63, 0x78, 0x71, 0x4e, 0x47, 0x5c, 0x55,
          0x22, 0x2b, 0x30, 0x39, 0x06, 0x0f, 0x14, 0x1d,
          0x25, 0x2c, 0x37, 0x3e, 0x01, 0x08, 0x13, 0x1a,
          0x6d, 0x64, 0x7f, 0x76, 0x49, 0x40, 0x5b, 0x52,
          0x3c, 0x35, 0x2e, 0x27, 0x18, 0x11, 0x0a, 0x03,
          0x74, 0x7d, 0x66, 0x6f, 0x50, 0x59, 0x42, 0x4b,
          0x17, 0x1e, 0x05, 0x0c, 0x33, 0x3a, 0x21, 0x28,
          0x5f, 0x56, 0x4d, 0x44, 0x7b, 0x72, 0x69, 0x60,
          0x0e, 0x07, 0x1c, 0x15, 0x2a, 0x23, 0x38, 0x31,
          0x46, 0x4f, 0x54, 0x5d, 0x62, 0x6b, 0x70, 0x79
};

static inline u8 crc7_byte(u8 crc, u8 data)
{
  return crc7_syndrome_table[(crc << 1) ^ data];
}

int main(int argc, char *argv[])
{
  unsigned long long num;
  unsigned char crc, dbyte;
  int i;

  if (argc < 2) {
    fprintf(stderr, "Usage: %s [num]\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  num = strtoll(argv[1], NULL, 16);
  num &= 0xffffffffff; /* 5 byte number */
  printf("num : %llx\n", num);
  crc = 0;

  for (i = 4; i >= 0; i--) {
    dbyte = (num >> (i * 8));
    crc = crc7_byte(crc, dbyte);
  }

  crc = (crc << 1) | 0x01;

  printf("crc: %x\n", crc);

  return 0;
}
```

sdcard/pcm2text.c

```c
#include <stdio.h>
#include <stdlib.h>

/* Convert 16-bit big endian PCM data to textual integers */

int main(void)
{
  int lowbyte = 0;
  short sample;
  char byte;
  int err;

  while ((err = fread(&byte, 1, 1, stdin)) > 0) {
    if (lowbyte) {
      sample |= byte;
      printf("%hd\n", sample);
    } else
      sample = byte << 8;
    lowbyte = !lowbyte;
  }

  if (err < 0) {
    perror("fread");
    return -1;
  }

  return 0;
}
```

sdcard/testdata.c

```c
#include <stdio.h>
#include <stdlib.h>

#define TEST_SIZE 512
#define TEST_NB (2 * TEST_SIZE)

int main(int argc, char *argv[])
{
  unsigned char data[2 * TEST_NB];
  unsigned short i, out;
  FILE * f;

  if (argc < 2) {
    fprintf(stderr, "Usage: %s\n", argv[0]);
    return EXIT_SUCCESS;
  }

  for (i = 0; i < TEST_SIZE; i++) {
    out = (i & (1 << 8)) ? ~i : i;
    data[2 * i] = (out >> 8) & 0xff;
    data[2 * i + 1] = out & 0xff;
  }

  f = fopen(argv[1], "w");
  if (f == NULL) {
    perror("open()");
    return EXIT_FAILURE;
  }
  if (fwrite(data, 1, TEST_NB, f) != TEST_NB) {
    fprintf(stderr, "Write unsuccessful\n");
    return EXIT_FAILURE;
  }
  fclose(f);

  return 0;
}
```

sdcard/text2pcm.c

```c
#include <stdio.h>
#include <stdlib.h>

/* Converts text-format numbers on standard input
 * writes out 16-bit big-endian PCM data on standard output */

int main(void)
{
  short sample;
  unsigned char buffer[512];
  int i = 0;

  while (fscanf(stdin, "%hd\n", &sample) == 1) {
    buffer[i] = (sample >> 8) & 0xff;
    buffer[i + 1] = sample & 0xff;
    i += 2;

    if (i == 512) {
      fwrite(buffer, 2, 256, stdout);
      i = 0;
    }
  }

  return 0;
}
```

A.4 Python

fft/inputgen.py

```python
import numpy as np
import sys

# Produces N samples of a sine wave at W Hz frequency

if __name__ == '__main__':
    if len(sys.argv) < 3:
        print("Usage: " + sys.argv[0] + " N W")
        sys.exit(1)
    fs = 44100.
    N = int(sys.argv[1])
    W = float(sys.argv[2])
    i = np.array(range(0, N))
    t = i / fs
    x = (2 ** 15 - 1) * np.cos(2 * np.pi * W * t)
    for real in x:
        print int(real)
```

fft/plot.py

```python
import matplotlib.pyplot as plt
import math
import sys

if __name__ == '__main__':
    comp = False
    if len(sys.argv) >= 2 and sys.argv[1] == "comp":
        comp = True

    if comp:
        vals = [tuple([int(i) for i in s.strip().split(' ')]) for s in sys.stdin]
        plt.plot([math.sqrt(r ** 2 + i ** 2) for (r, i) in vals])
    else:
        vals = [int(s.strip()) for s in sys.stdin]
        plt.plot(vals)
    plt.show()
```

roms/arraygen.py

```
#!/usr/bin/env python

# Converts a stream of numbers into a VHDL array initializer

import struct
import sys
import math
import os.path
import sys

def convert_int(x, size):
    fmt = "x\"%%0%dx\"" % (size / 4)
    return fmt % (x & ((1 << size) - 1))

def convert_int_array(arr, size):
    return ', '.join([convert_int(x, size) for x in arr])



if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("Usage: arraygen.py dtype dsize [romfile]")
        sys.exit(1)

    dsize = int(sys.argv[1])

    if len(sys.argv) >= 3:
        romfile = open(sys.argv[2])
    else:
        romfile = sys.stdin

    values = [int(line.strip()) for line in romfile]
    arrstr = convert_int_array(values, dsize)

    print "(" + arrstr + ")"
```

roms/mifgen.py

```
import sys

def convert_int(x, size):
    fmt = "%%0%dx" % (size / 4)
    return fmt % (x & ((1 << size) - 1))

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("Usage: mifgen.py width [romfile]")
        sys.exit(1)

    width = int(sys.argv[1])

    if len(sys.argv) >= 3:
        romfile = open(sys.argv[2])
    else:
        romfile = sys.stdin

    values = [int(line.strip()) for line in romfile]
    depth = len(values)

    print("WIDTH=%d;" % width)
    print("DEPTH=%d;\n" % depth)
    print("ADDRESS_RADIX=UNS;")
    print("DATA_RADIX=HEX;\n")
    print("CONTENT BEGIN\n")

    for i, val in enumerate(values):
        print("%d : %s;" % (i, convert_int(val, width)))

    print("\nEND;")
```

sdcard/mkauimg.py

```
# Usage: ./mkauimg.py audio1.ogg [audio2.mp3 ...] output.raw
# Takes in several audio files, concatenates them, and converts
# then into raw 16-bit signed-integer big-endian PCM data.
#
# First words contain the lengths of all the files.

import sys
import subprocess
import os
import struct
import mutagen
from mutagen.mp3 import MP3

def get_song_title(filename):
    metadata = mutagen.File(filename)

    if type(metadata) is MP3:
        title = metadata['TIT2'].text[0]
        artist = metadata['TPE1'].text[0]
    else:
        title = metadata['title'][0]
        artist = metadata['artist'][0]

    return str(title + " - " + artist)

def make_header(offset, filename):

    if filename is not None:
        title = get_song_title(filename)
        print title

        if len(title) >= 60:
            raise Exception("Song title is too long")

        header = struct.pack(">I", offset) + title
        header = header.ljust(64, '\0')
    else:
        header = struct.pack(">I", offset).ljust(64, '\0')

    return header
```

```
if __name__ == '__main__':
    filesizelist = []
    curoff = 1
    offsetlist = [curoff]
    inputs = sys.argv[1:-1]

    if len(inputs) >= 8:
        print "Too many songs. Will only write the first seven."
        inputs = inputs[:7]

    for i, inptfile in enumerate(inputs):
        # convert to PCM using sox
        sox_call = "sox \"" + inptfile + "\" -b 16 -e signed-integer -B -c 1 -t raw
            tempfileformkau" + str(i)
        print(sox_call)
        os.system(sox_call)

# get number of characters, need to convert to 512byte block
        filelength = os.path.getsize("tempfileformkau" + str(i))
        newlength = int((filelength - 1)/512 + 1)
        filesizelist.append(newlength)

    for size in filesizelist:
        curoff += size
        offsetlist.append(curoff)

    print("Track offsets: " + str(offsetlist))

    header = ''.join([make_header(off, fname)
                      for (off, fname) in zip(offsetlist[:-1], inputs)])
    header += make_header(offsetlist[-1], None)
    header = header.ljust(512, '\0')

    f = open(sys.argv[-1], "w")
    f.write(header)

    for i in range(0, len(inputs)):
        fin = open("tempfileformkau" + str(i), "r")
        data = fin.read(512)
        while len(data) == 512:
            f.write(data)
            data = fin.read(512)

        if len(data) > 0:
            f.write(data)
            padding = '\0' * (512 - len(data))
            f.write(padding)

    for i in range(0, len(sys.argv) - 2):
        os.remove("tempfileformkau" + str(i))
```

A.5 Shell

fft/fftsim.sh

```
#!/bin/bash

# Simulates the hardware FFT compilation using dftsim and recombsim
# Takes 256 numbers line by line on standard input and produces
# 256 complex number pairs line by line on standard output

if [ -z "$1" ]; then
  cat > fftinputs.txt
  inputs=( $(<fftinputs.txt) )
```

```
else
  inputs=( $(<"$1") )
fi

reorder=(0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15)

for i in {0..15}
do
  for j in {0..15}
  do
    base=$(expr $j \* 16 )
    addr=$(expr $base + ${reorder[$i]})
    echo ${inputs[$addr]}
  done | ./dftsim > dft${i}.txt
done

for i in {0..7}
do
  even=$(expr 2 \* $i)
  odd=$(expr $even + 1)
  paste -d " " dft${even}.txt dft${odd}.txt | ./recombsim 32 > recomb1-${i}.txt
done

for i in {0..3}
do
  even=$(expr 2 \* $i)
  odd=$(expr $even + 1)
  paste -d " " recomb1-${even}.txt recomb1-${odd}.txt | ./recombsim 64 > recomb2-${i}.txt
done

for i in {0..1}
do
  even=$(expr 2 \* $i)
  odd=$(expr $even + 1)
  paste -d " " recomb2-${even}.txt recomb2-${odd}.txt | ./recombsim 128 > recomb3-${i}.txt
done

paste -d " " recomb3-0.txt recomb3-1.txt | ./recombsim 256

rm dft{0..15}.txt
rm recomb1-{0..7}.txt
rm recomb2-{0..3}.txt
rm recomb3-{0..1}.txt
```

sdcard/playimage.sh

```
#!/bin/sh
# Plays raw PCM data

play -b 16 -e signed-integer -B -c 1 -r 44100 -t raw $1
```