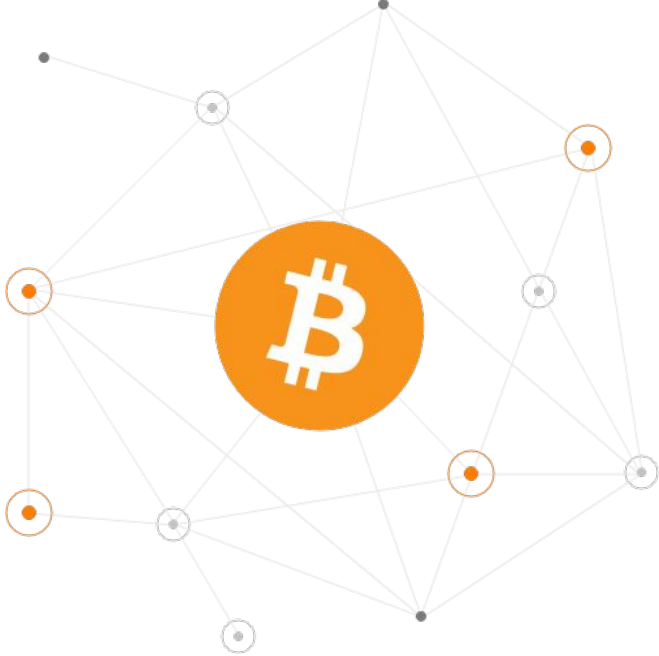
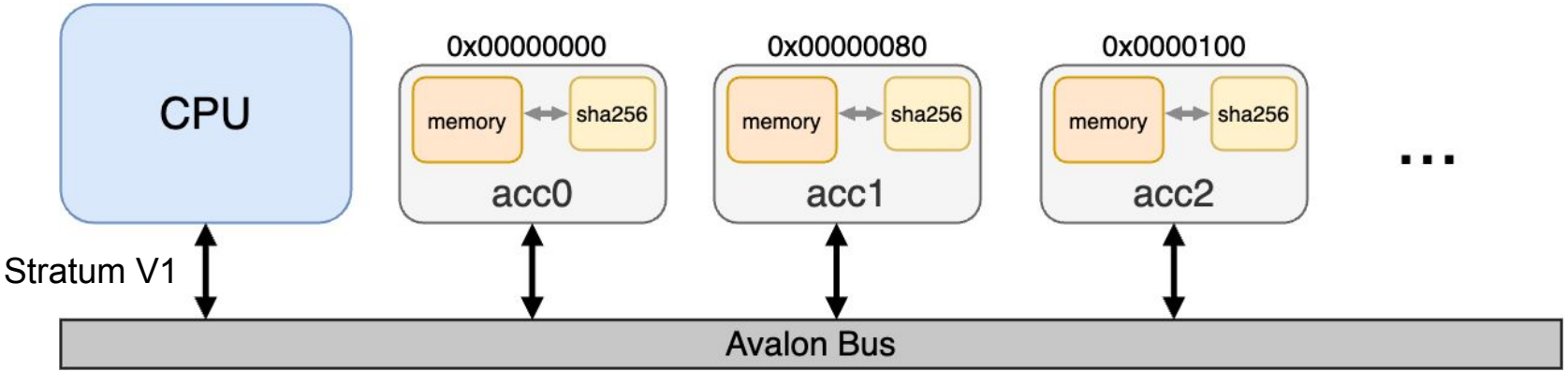
GoldMiner

Jules Comte
Mingyang Song
Zhe Mo
Tianyu Qin
Xueji Zhao



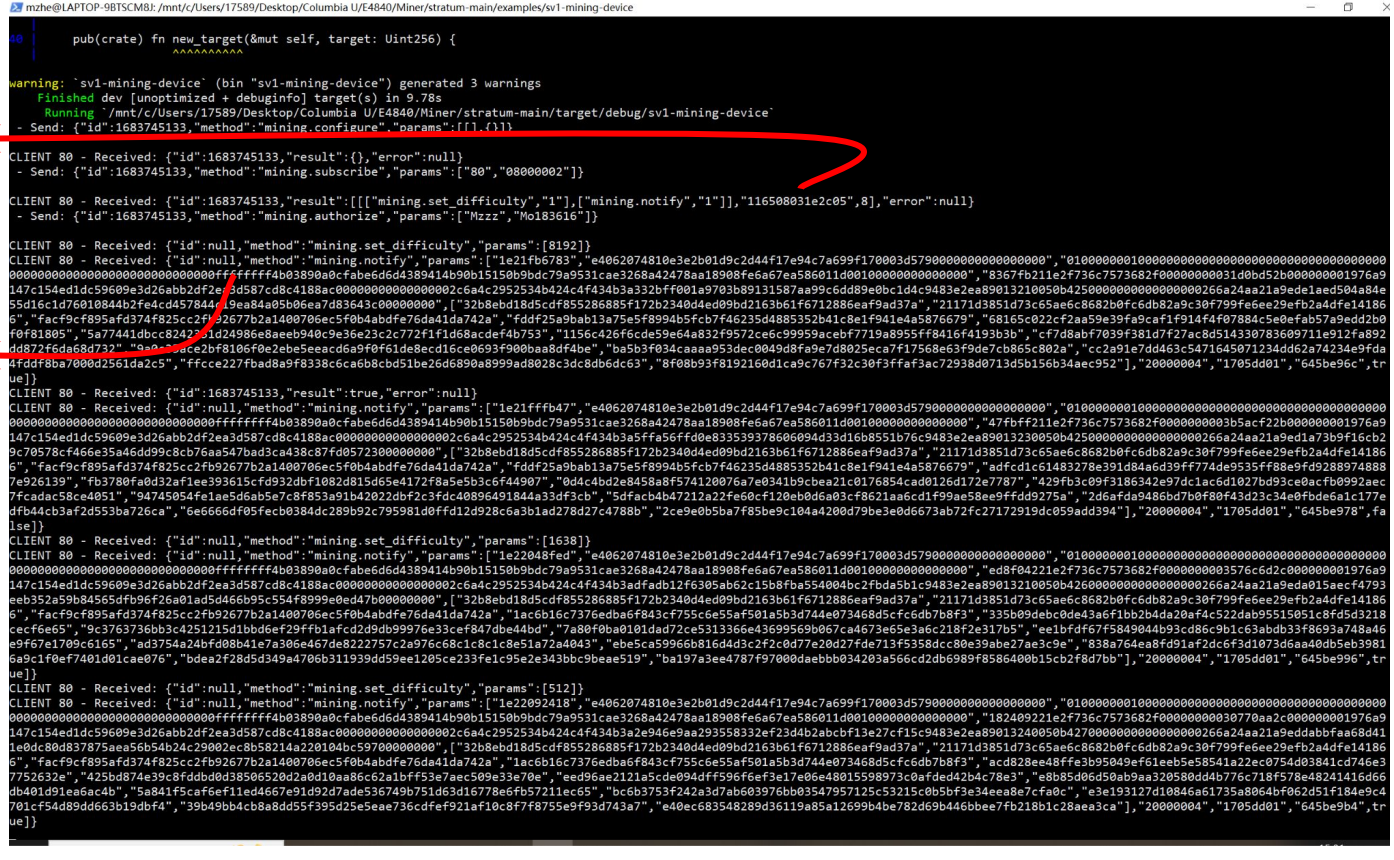
Overview of Project Design



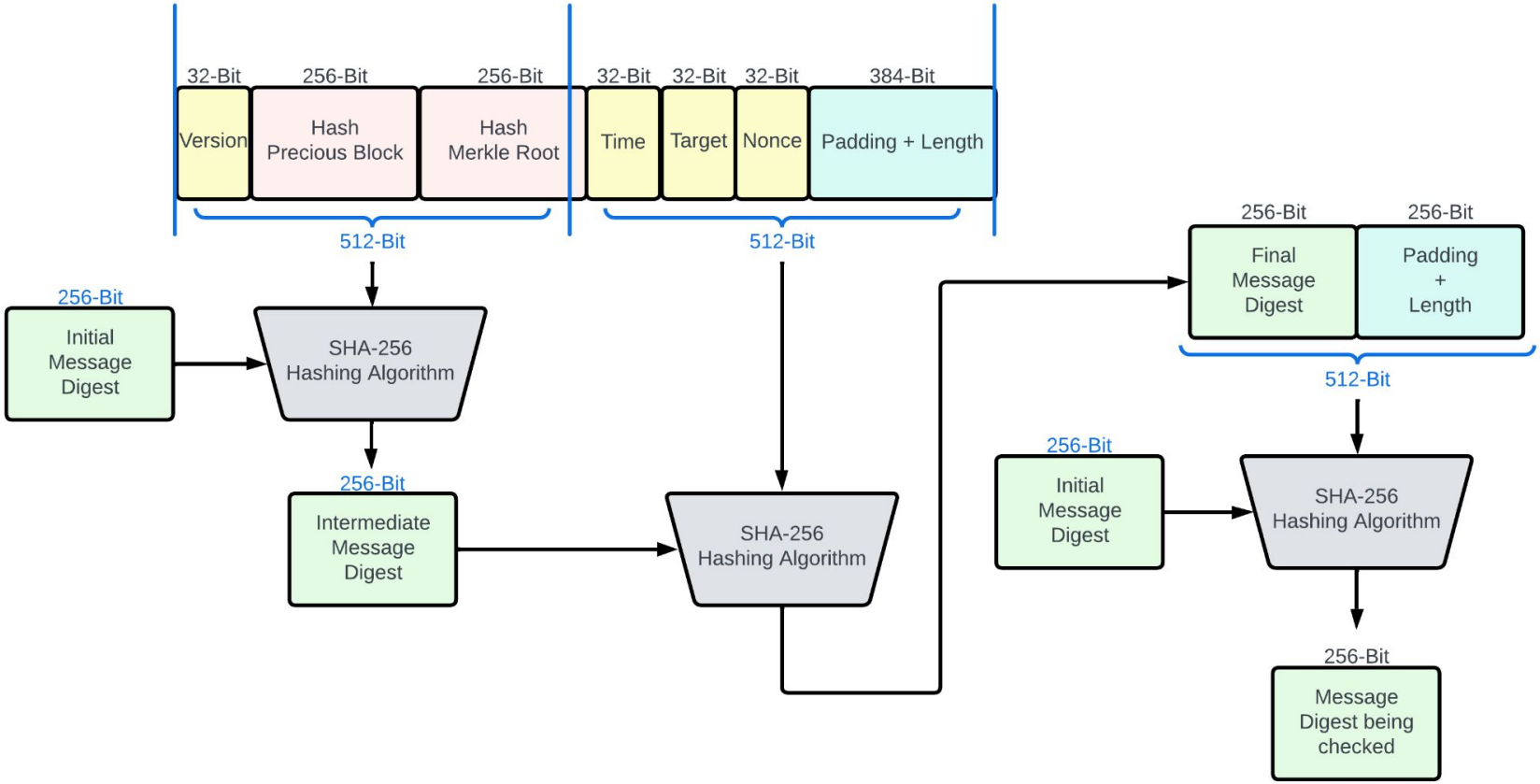
Message Extracted from Mining Pool

- Extranonce1
- Extranonce2_size

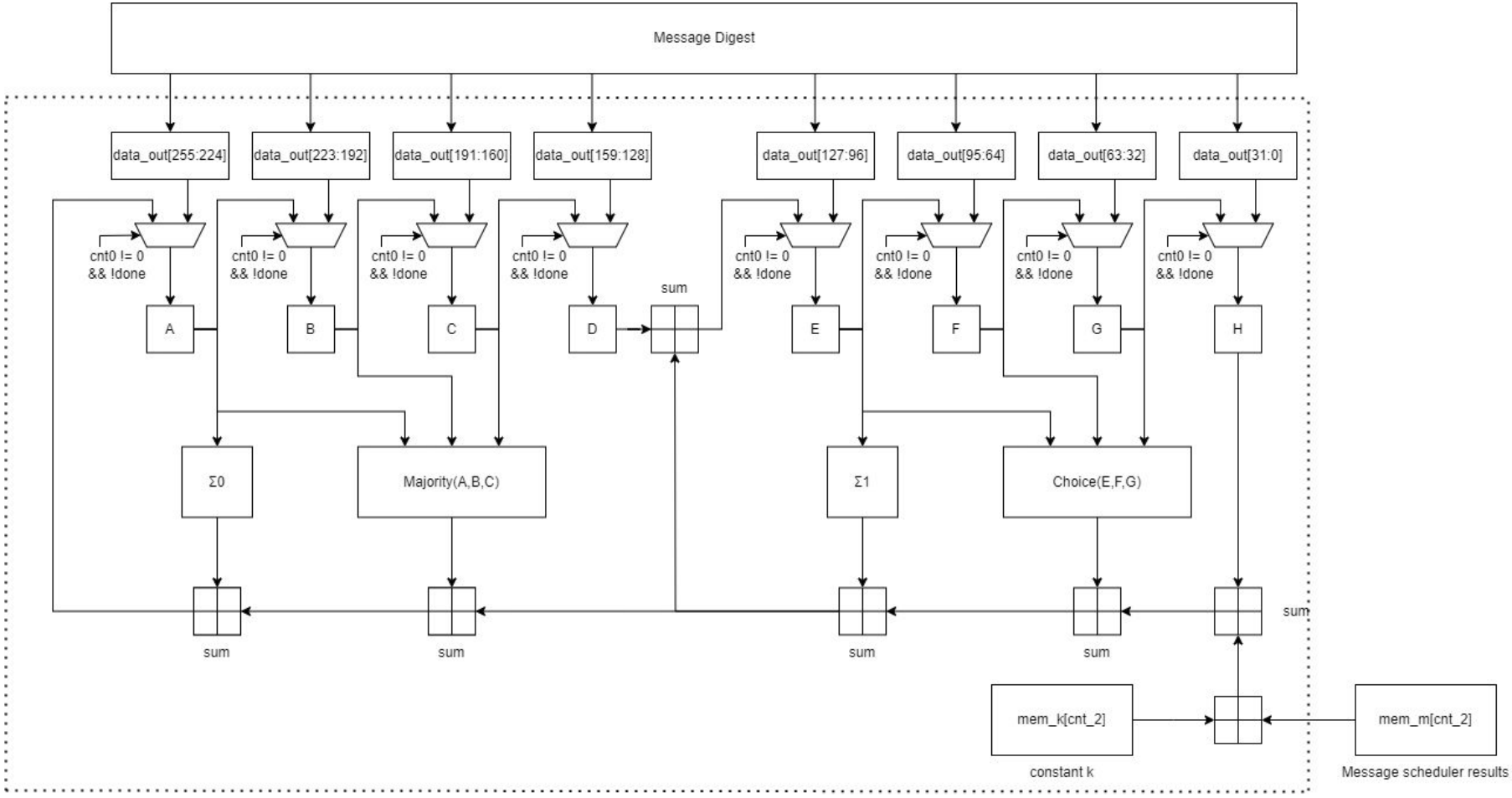
- id
- methods
- params
  - job id
  - …
  - Coinbase Transaction Part 1
  - Coinbase Transaction Part 2



Overview of SHA-256 Process



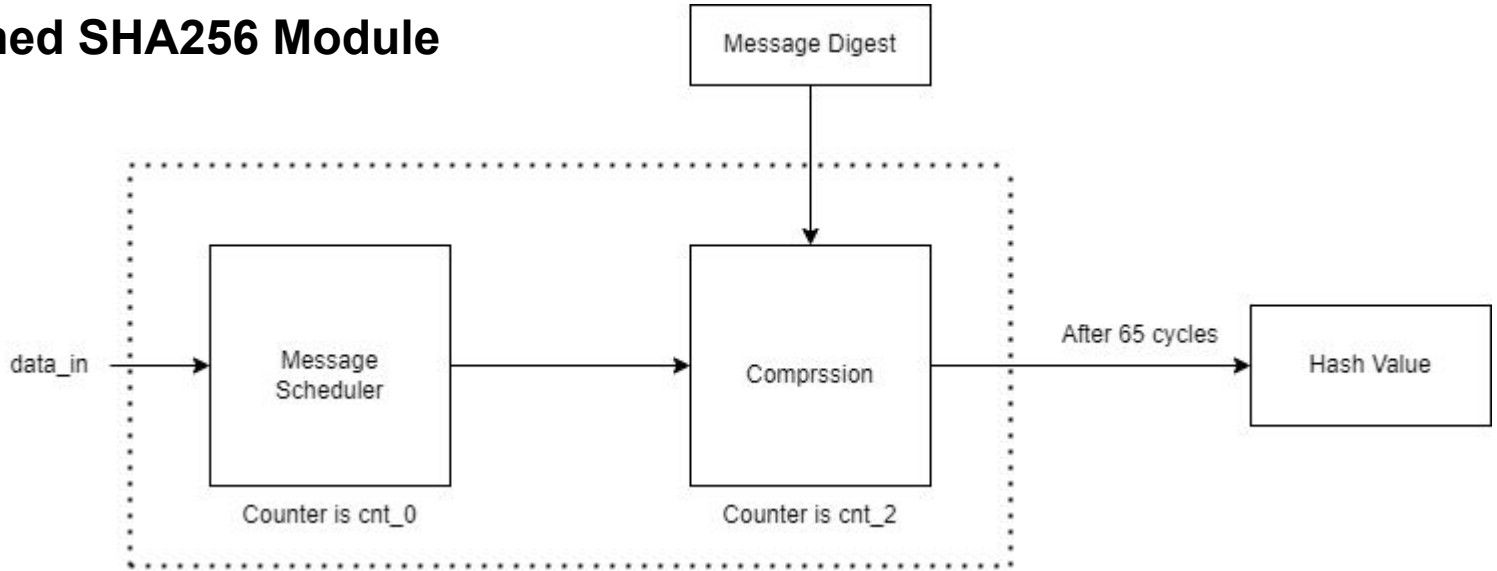
Compression



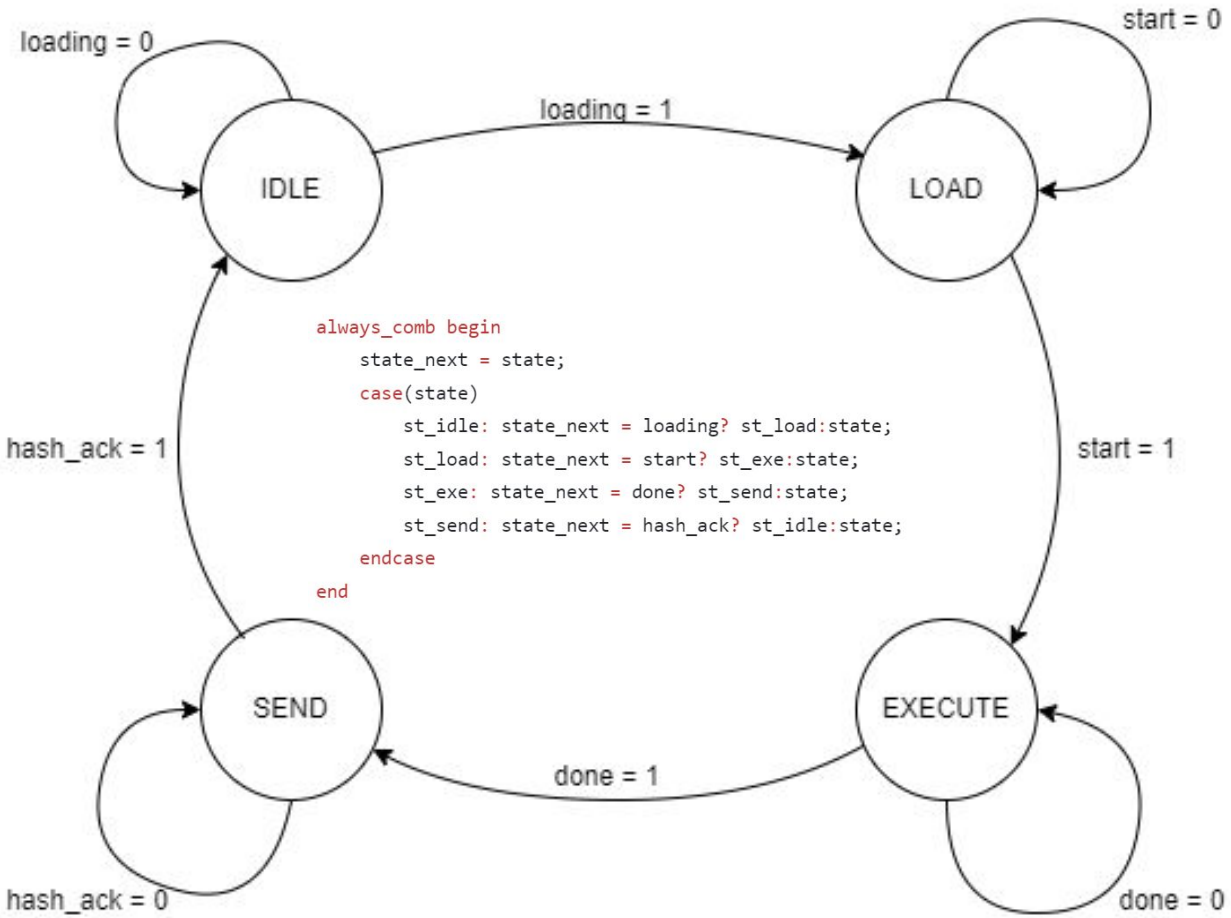
Message Scheduler

```
if(cnt_0 < 16){
        mem_m[cnt_0] reads from input data
}
else if(cnt_0 < 64){
        mem_m[cnt_0] = sig1(mem_m[cnt_0-2])+mem_m[cnt_0-7]+sig0(mem_m[cnt_0-5])+mem_m[cnt_0-16]
}
```

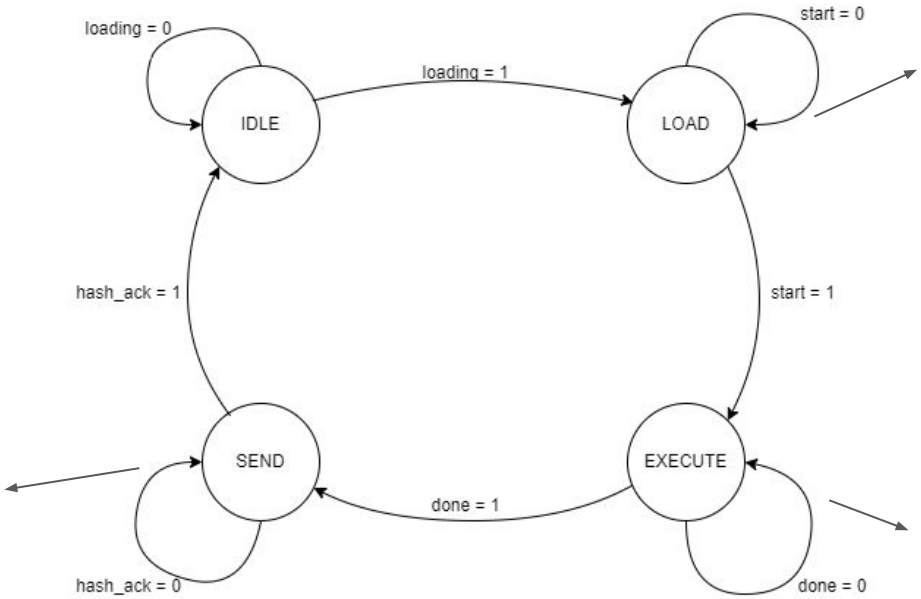
Combined SHA256 Module



Hardware Finite State Machine



Hardware Finite State Machine

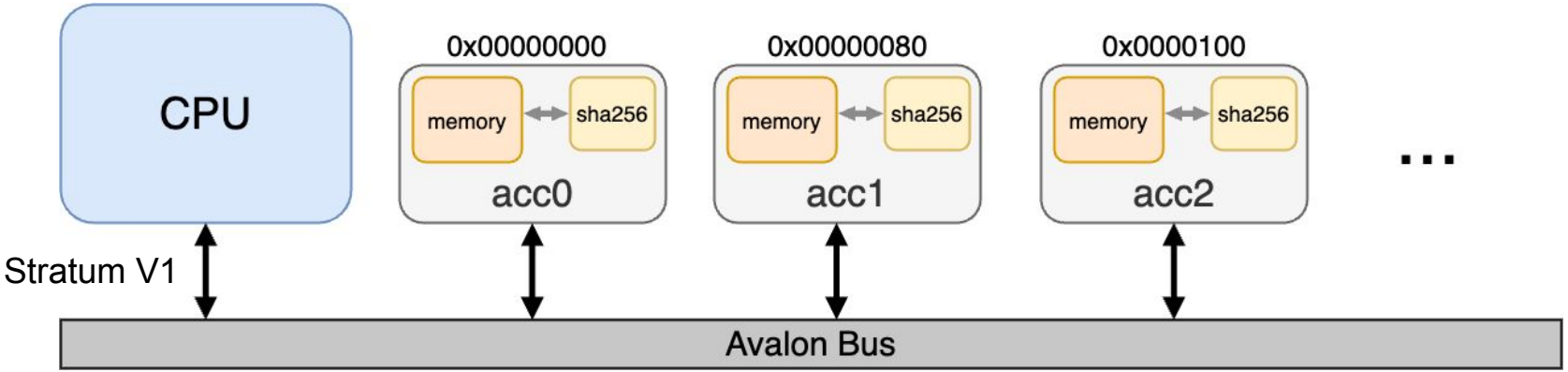



```
always_ff @ (posedge clk)
    if (reset)
        data_out <= 0;
    else
        if(state == st_send && reading)
            case(address)
                0: data_out <= hashvalue[31:0];
                1: data_out <= hashvalue[63:32];
                2: data_out <= hashvalue[95:64];
                3: data_out <= hashvalue[127:96];
                4: data_out <= hashvalue[159:128];
                5: data_out <= hashvalue[191:160];
                6: data_out <= hashvalue[223:192];
                7: data_out <= hashvalue[255:224];
                17: data_out <= 32'h11111111;
                default: data_out <= 2;
            endcase
        else
            data_out <= 32'h0f0ff0f0;
```

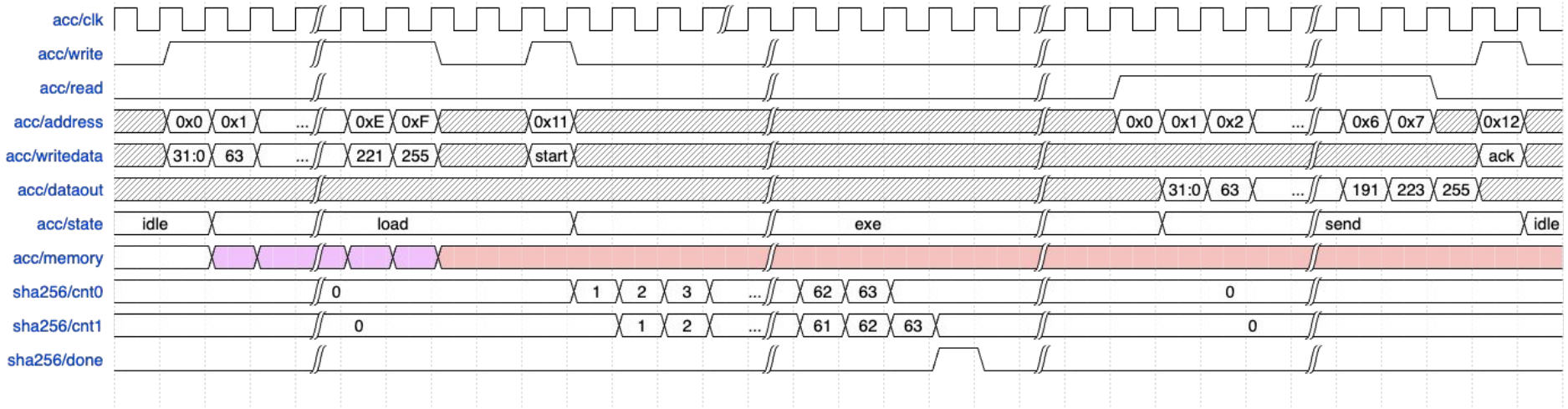
```
always_ff @ (posedge clk) begin
    if (reset)  buffer <= 0;
    else
        if(loading)
            case(address)
                0: buffer[31:0] <= writedata;
                1: buffer[63:32] <= writedata;
                2: buffer[95:64] <= writedata;
                3: buffer[127:96] <= writedata;
                4: buffer[159:128] <= writedata;
                5: buffer[191:160] <= writedata;
                6: buffer[223:192] <= writedata;
                7: buffer[255:224] <= writedata;
                8: buffer[287:256] <= writedata;
                9: buffer[319:288] <= writedata;
                10: buffer[351:320] <= writedata;
                11: buffer[383:352] <= writedata;
                12: buffer[415:384] <= writedata;
                13: buffer[447:416] <= writedata;
                14: buffer[479:448] <= writedata;
                15: buffer[511:480] <= writedata;
            endcase
end
```

```
/**** Module ports map ****/
sha256_module sha256_module_0(
    .clk(clk),
    .reset(acc_reset),
    .start(start),
    .data_in(buffer),
    .data_out(hashvalue),
    .done(done)
);
/*************************/
```

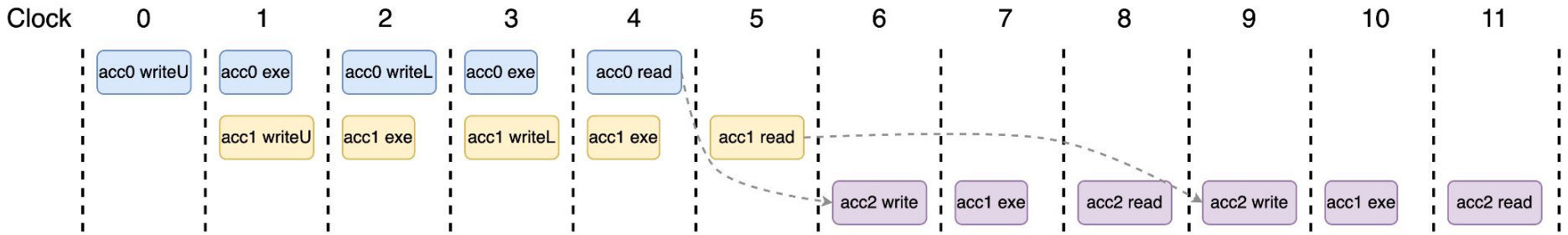
Overview of Project Design



Hardware Timing Diagram



Hardware Pipeline



Speedup: Approximately 25%

HW vs SW

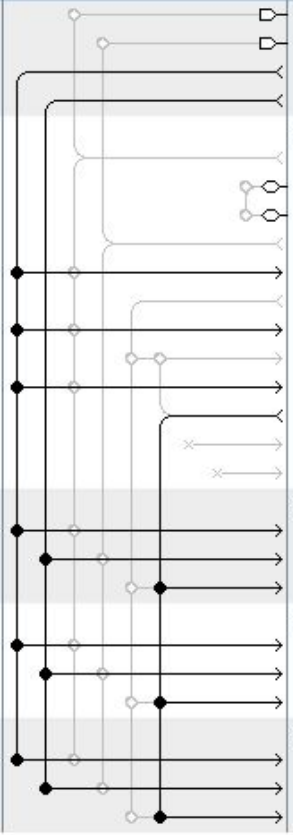
85% Speedup

```
root@de1-soc:~/sw# ./a.out s
Software SHA256
(SOFTWARE)Time elpased: 0.867471 seconds
----- OUTPUT OF sofware block0 -----
hash_output.r0: f260764c
hash_output.r1: 2ae540d6
hash_output.r2: 437a88a6
hash_output.r3: d9001e2e
hash_output.r4: 8fa8f8a8
hash_output.r5: bd650000
hash_output.r6: 0
hash_output.r7: 0
----- OUTPUT OF sofware block1 -----
hash_output.r0: ceb1e4e8
hash_output.r1: 3bcfa266
hash_output.r2: 97eef0d3
hash_output.r3: 55633391
hash_output.r4: 1061f23f
hash_output.r5: 7b77dc15
hash_output.r6: 0
hash_output.r7: 0
root@de1-soc:~/sw# ./a.out h
Hardware SHA256
(HARDWARE)Time elpased: 0.468272 seconds
----- OUTPUT OF ACC0 -----
hash_output0.r0: f260764c
hash_output0.r1: 2ae540d6
hash_output0.r2: 437a88a6
hash_output0.r3: d9001e2e
hash_output0.r4: 8fa8f8a8
hash_output0.r5: bd650000
hash_output0.r6: 0
hash_output0.r7: 0
----- OUTPUT OF ACC1 -----
hash_output1.r0: ceb1e4e8
hash_output1.r1: 3bcfa266
hash_output1.r2: 97eef0d3
hash_output1.r3: 55633391
hash_output1.r4: 1061f23f
hash_output1.r5: 7b77dc15
hash_output1.r6: 0
hash_output1.r7: 0
```

```
sha256_transform:
    @ args = 0, pretend = 0, frame = 320
    @ frame_needed = 1, uses_anonymous_args = 0
    push    {r7, lr}
    sub sp, sp, #320
    add r7, sp, #0
    adds    r3, r7, #4
    str r0, [r3]
    mov r3, r7
    str r1, [r3]
    movw    r3, #:lower16:__stack_chk_guard
    movt    r3, #:upper16:__stack_chk_guard
    ldr r3, [r3]
    str r3, [r7, #316]
    add r3, r7, #44
    movs    r2, #0
    str r2, [r3]
    add r3, r7, #48
    movs    r2, #0
    str r2, [r3]
    b   .L2
.L3:
    mov r2, r7
    add r3, r7, #48
    ldr r2, [r2]
    ldr r3, [r3]
    add r3, r3, r2
    ldrb    r3, [r3]    @ zero_extendqisi2
    mov r1, r3
    add r3, r7, #48
    ldr r3, [r3]
    adds    r3, r3, #1
    mov r2, r7
    ldr r2, [r2]
    add r3, r3, r2
    ldrb    r3, [r3]    @ zero_extendqisi2
    lsls    r3, r3, #8
    orr r2, r1, r3
    add r3, r7, #48
    ldr r3, [r3]
    adds    r3, r3, #2
    mov r1, r7
    ldr r1, [r1]
    add r3, r3, r1
```

Multi-Device Project

Project goal: 3 independent sha256 devices all managed by a single driver.



Connections	Name	Description	Export	Clock	Base	End	IRQ
	clk_0	Clock Source					
	clk_in	Clock Input	**clk**	***exported***			
	clk_in_reset	Reset Input	**reset**				
	clk	Clock Output	*Double-click to*	clk_0			
	clk_reset	Reset Output	*Double-click to*				
	hps_0	Arria V/Cyclone V Hard Proce...					
	h2f_user1_clock	Clock Output	*Double-click to*	hps_0_h2...			
	memory	Conduit	**hps_ddr3**				
	hps_io	Conduit	**hps**				
	h2f_reset	Reset Output	*Double-click to*				
	h2f_axi_clock	Clock Input	*Double-click to*	**clk_0**			
	h2f_axi_master	AXI Master	*Double-click to*	[h2f_axi_...			
	f2h_axi_clock	Clock Input	*Double-click to*	**clk_0**			
	f2h_axi_slave	AXI Slave	*Double-click to*	[f2h_axi_...			
	h2f_lw_axi_clock	Clock Input	*Double-click to*	**clk_0**			
	h2f_lw_axi_master	AXI Master	*Double-click to*	[h2f_lw_a...			
	f2h_irq0	Interrupt Receiver	*Double-click to*		IRQ 0	IRQ 31	
	f2h_irq1	Interrupt Receiver	*Double-click to*		IRQ 0	IRQ 31	
	vga_ball_0	VGA Ball					
	clock	Clock Input	*Double-click to*	**clk_0**			
	reset	Reset Input	*Double-click to*	[clock]			
	avalon_slave_0	Avalon Memory Mapped Slave	*Double-click to*	[clock]	**0x0000_0000**	0x0000_007f	
	vga_ball_1	VGA Ball					
	clock	Clock Input	*Double-click to*	**clk_0**			
	reset	Reset Input	*Double-click to*	[clock]			
	avalon_slave_0	Avalon Memory Mapped Slave	*Double-click to*	[clock]	**0x0000_0100**	0x0000_017f	
	vga_ball_2	VGA Ball					
	clock	Clock Input	*Double-click to*	**clk_0**			
	reset	Reset Input	*Double-click to*	[clock]			
	avalon_slave_0	Avalon Memory Mapped Slave	*Double-click to*	[clock]	**0x0000_0200**	0x0000_027f	

```
module vga_ball(input logic        clk,
                input logic        reset,
                input logic [31:0]  writedata,
                output logic [31:0]  readdata,
                input logic        write,
                input logic        read,
                input              chipselect,
                input logic [4:0]  address);
```

Multi-Device Project

Registers:

All the interface registers are 4-bytes wide.

address [0-15]: write-only, for the software to send the 64 byte input to the hardware.

address [0-7]: read-only, for the software to read the 32 byte hash.

address [15]: read-only, 1 when the hash is ready to be read from address [0-7].

address [31]: write-only, for the software to send a reset signal.

```
/* ioctls and their arguments */
#define WRITE_INPUT_0 _IOW(SHA256_MAGIC, 1, sha256_arg *)
#define WRITE_INPUT_1 _IOW(SHA256_MAGIC, 4, sha256_arg *)
#define WRITE_INPUT_2 _IOW(SHA256_MAGIC, 7, sha256_arg *)

#define READ_DONE_0   _IOW(SHA256_MAGIC, 2, sha256_arg *)
#define READ_DONE_1   _IOW(SHA256_MAGIC, 5, sha256_arg *)
#define READ_DONE_2   _IOW(SHA256_MAGIC, 8, sha256_arg *)

#define READ_HASH_0   _IOW(SHA256_MAGIC, 3, sha256_arg *)
#define READ_HASH_1   _IOW(SHA256_MAGIC, 6, sha256_arg *)
#define READ_HASH_2   _IOW(SHA256_MAGIC, 9, sha256_arg *)

#define RESET_0       _IOW(SHA256_MAGIC, 10, sha256_arg *)
#define RESET_1       _IOW(SHA256_MAGIC, 11, sha256_arg *)
#define RESET_2       _IOW(SHA256_MAGIC, 12, sha256_arg *)
#endif
```

```c
/*
 * Information about our device
 */
struct sha256_dev {
        struct resource res; /* Resource: our registers */
        void __iomem *virtbase; /* Where registers can be accessed in memory */
} dev[3];
```

```c
static int __init sha256_probe(struct platform_device *pdev)
{
        int ret;
        char *driver_names[] = {
                DRIVER_NAME_0,
                DRIVER_NAME_1,
                DRIVER_NAME_2,
        };

        /* Register ourselves as a misc device: creates /dev/sha256 */
        ret = misc_register(&sha256_misc_device[init_device_count]);

        /* Get the address of our registers from the device tree */
        ret = of_address_to_resource(pdev->dev.of_node, 0, &dev[init_device_count].res);
        if (ret) {
                ret = -ENOENT;
                goto out_deregister;
        }

        /* Make sure we can use these registers */
        if (request_mem_region(dev[init_device_count].res.start, resource_size(&dev[init_device_count].res),
                               driver_names[init_device_count]) == NULL) {
                ret = -EBUSY;
                goto out_deregister;
        }
```

```
        write_input(raw_fd, input, index);

        let mut done: c_uint = 0;
        loop {
            read_done(raw_fd, &mut done, index);
            if done == 1 {
                break;
            }
        }

        pointer += 64;
    }

    let mut hash: sha256_hash = sha256_hash::default();
    read_hash(raw_fd, &mut hash, index);
```

Multi-Device Testbench

Steps of the test:

1. Get a random block header from mempool.space.
2. Calculate its hash using the reference rust-bitcoin library.
3. Calculate the same hash using the three hardware devices.
4. Calculate the same hash using our own software implementation of SHA256.
5. Check all three of the hardware output against the golden output.

```rust
// GET HEADER BYTES
let header_bytes = get_block_header(height)?;

// GET GOLDEN HASH
let mut cursor = Cursor::new(header_bytes);
let header = Header::consensus_decode(&mut cursor)?;
let start = Instant::now();
let gold_hash = header.block_hash().to_raw_hash();
let end = Instant::now();
let duration = end - start;
*gold_time = *gold_time + duration;
```

```
start = Instant::now();
// The i parameter here sets which device to use
hash = sha256_hw::get_hash(&sha256_hw::get_hash(&header_bytes, i), i);
end = Instant::now();
duration = end - start;
*hw_time = *hw_time + duration;
```

```
start = Instant::now();
hash = sha256_sw::get_hash(&sha256_sw::get_hash(&header_bytes));
end = Instant::now();
duration = end - start;
*sw_time = *sw_time + duration;
```

```
//CHECK
if gold_hash.as_byte_array() == hash.as_slice() {
    print!("PASS  ");
    println!("{:?}", duration);
} else {
    println!("FAIL  ");
    println!("{:?}", duration);
}
```

Multi-Device Testbench Result

```
 BLOCK  390397
GOLDEN  0d3e24c943a9683d8c1353e21f62fa8f0293d166948c380000000000000000        26.85µs
 HW[0]  0d3e24c943a9683d8c1353e21f62fa8f0293d166948c380000000000000000  PASS  114.73µs
 HW[1]  0d3e24c943a9683d8c1353e21f62fa8f0293d166948c380000000000000000  PASS  74.571µs
 HW[2]  0d3e24c943a9683d8c1353e21f62fa8f0293d166948c380000000000000000  PASS  73.02µs
 SW     0d3e24c943a9683d8c1353e21f62fa8f0293d166948c380000000000000000  PASS  16.48µs

Average gold time: 25.754µs
Average hw time  : 87.114µs
Average sw time  : 16.528µs
```

Average time calculated based on 25 tests

THANK YOU VERY MUCH FOR LISTENNNNINNNGGG!!!