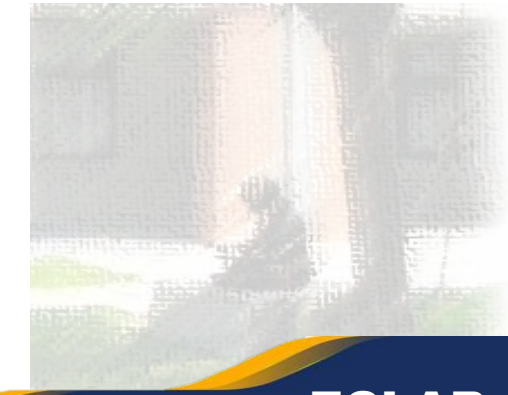
Final Project

陳瀚坪、姜維、吳鎧帆

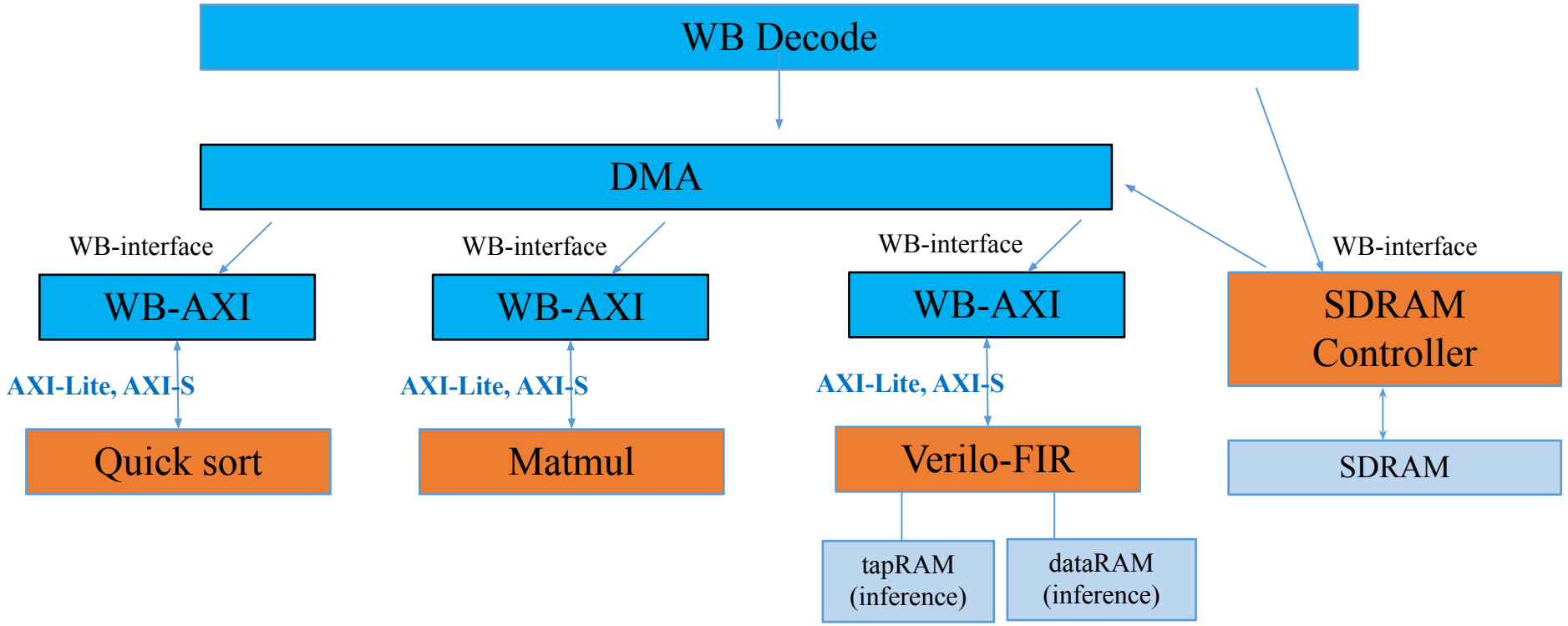


Outline

- Architecture
- SDRAM Controller
- Firmware Code
- DMA Transfer Data
- Section Linker
- Memory Mapping lds
- Quick Sort (Insertion Sort)

Module Block

- DMA
- FIR, Matmul, Quick sort
- SDRAM

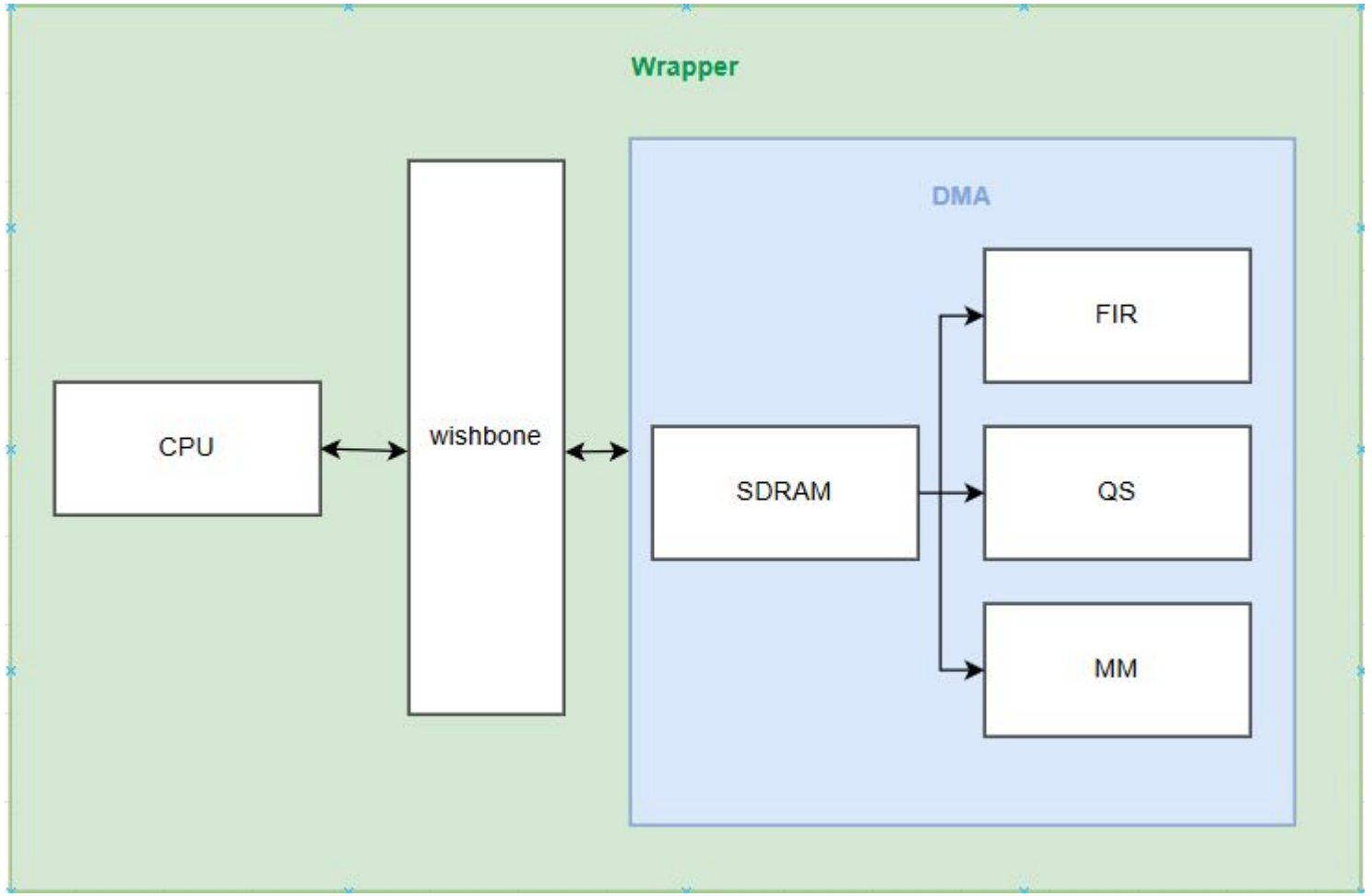


Architecture

- Free CPU resource
- Improve data transfer

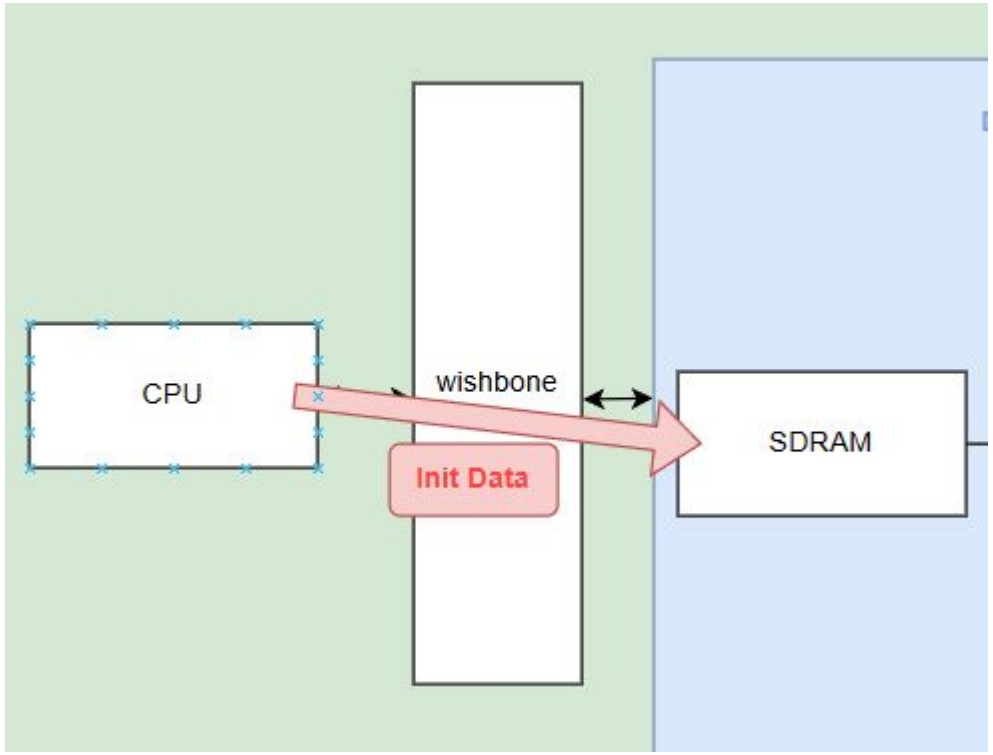
Flow :

1. Init module input data
2. Start
3. dma transfer data from SDRAM to Accelarator
4. CPU wait done



SDRAM Control

- SDRAM for Wishbon port




```
assign valid         = wbs_stb_i & wbs_cyc_i & (wbs_adr_i[31:16] == 16'h3800);
assign ctrl_in_valid = wbs_we_i ? valid : ~ctrl_in_valid_q && valid;
assign wbs_ack_o     = (wbs_we_i) ? ~ctrl_busy && valid : ctrl_out_valid;
assign bram_mask     = wbs_sel_i & {4{wbs_we_i}};
assign ctrl_addr     = wbs_adr_i[22:0];


// Assuming LA probes [65:64] are for controlling the count clk & reset
assign clk           = (~la_oenb[64]) ? la_data_in[64]: wb_clk_i;
assign rst           = (~la_oenb[65]) ? la_data_in[65]: wb_rst_i;

always @(posedge clk) begin
    if (rst) begin
        ctrl_in_valid_q <= 1'b0;
    end
    else begin
        if (~wbs_we_i && valid && ~ctrl_busy && ctrl_in_valid_q == 1'b0)
            ctrl_in_valid_q <= 1'b1;
        else if (ctrl_out_valid)
            ctrl_in_valid_q <= 1'b0;
    end
end

sdram_controller user_sdram_controller (
    .clk(clk),
    .rst(rst),

    .bram_mask(bram_mask),
    .user_addr(ctrl_addr),
    .rw(wbs_we_i),
    .data_in(wbs_dat_i),
    .data_out(wbs_dat_o),
    .busy(ctrl_busy),
    .in_valid(ctrl_in_valid),
    .out_valid(ctrl_out_valid)
);
```

SDRAM Control

- Memory mapping
- Change code/data in different memory bank

```
MEMORY {
    vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
    dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
    dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
    flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
    mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000

    mprjram : ORIGIN = 0x38000000, LENGTH = 0x00000200
    mprjdata : ORIGIN = 0x38000200, LENGTH = 0x00000200
    hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
    csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
}
```

```
.data :
{
    . = ALIGN(8);
    _fdata = .;
    *(.data .data.* .gnu.linkonce.d.*)
    *(.data1)
    _gp = ALIGN(16);
    *(.sdata .sdata.* .gnu.linkonce.s.*)
    . = ALIGN(8);
    _edata = .;
} > mprjdata AT > flash

.bss :
{
    . = ALIGN(8);
    _fbss = .;
    *(.dynsbss)
    *(.sbss .sbss.* .gnu.linkonce.sb.*)
    *(.scommon)
    *(.dynbss)
    *(.bss .bss.* .gnu.linkonce.b.*)
    *(COMMON)
    . = ALIGN(8);
    _ebss = .;
    _end = .;
} > mprjdata AT > flash

.mprjram :
{
    . = ALIGN(8);
    _fsram = .;
    *libgcc.a:*(.text .text.*)
} > mprjram AT > flash
```

Firmware code

- Change FIR transfer Input data method

Origin :

```
void __attribute__ ( ( section ( ".mprjram" ) ) ) fir_excute() {
    // StartMark
    wb_write(checkbits, 0x00A50000);

    // ap_start
    wb_write(reg_fir_ap_ctrl, 1);

    uint8_t register t = 0;

    wb_write(reg_fir_x_in, t);

    for (uint8_t t = 1; t < N; t++) {
        temp = wb_read(reg_fir_y_out);
        wb_write(reg_fir_x_in, t);
        outputsignal[t - 1] = temp;
    }

    temp = wb_read(reg_fir_y_out);
    outputsignal[t] = temp;
    // let TB check the final Y by using MPRJ[31:24]
    // and send the EndMark 5A signal at MPRJ[23:16]
    // check ap_done
    wb_read(reg_fir_ap_ctrl);
    wb_write(checkbits, outputsignal[N-1] << 24 | 0x005A0000);
}
```

Start change to init() to free CPU

Now :

```
#define x_data_base      0x38000200 // X input data
#define x_data_array ((volatile uint32_t *)x_data_base)
```

```
void __attribute__ ( ( section ( ".mprjram" ) ) ) initfir() {

    // program data length
    wb_write(reg_fir_len, data_length);

    // program coefficient
    for (uint32_t i = 0; i < 11; i++) {
        wb_write(adr_ofst(reg_fir_coeff, 4*i), taps[i]);
    }
    // program data
    for (uint32_t i = 0; i < N; i++) {
        x_data_array[i] = i;
    }

    // StartMark
    wb_write(checkbits, 0x00A50000);

    // ap_start
    wb_write(reg_fir_ap_ctrl, 1);
}
```

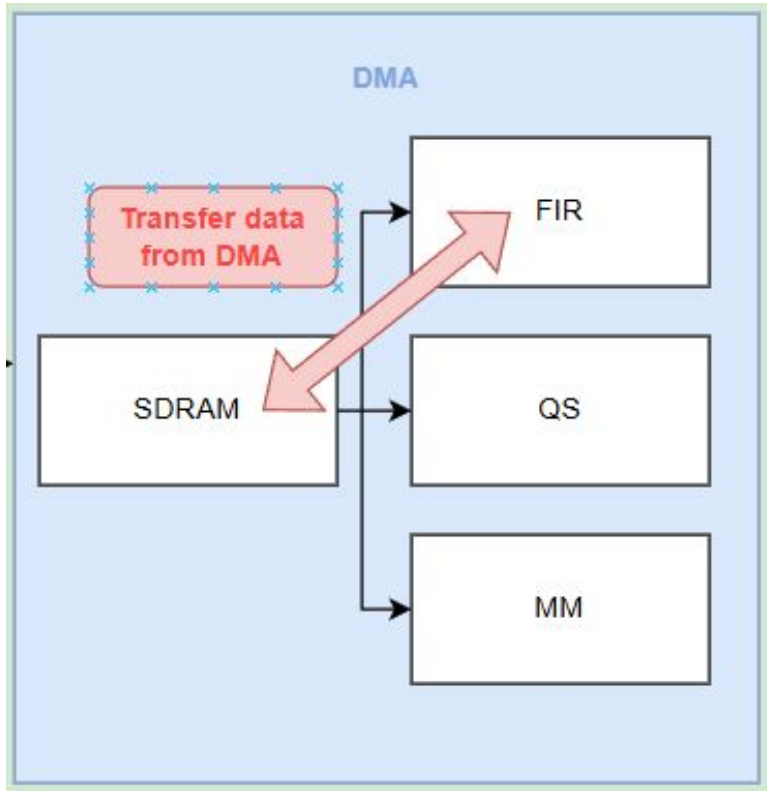
```
void __attribute__ ( ( section ( ".mprjram" ) ) ) fir_excute() {


    wb_read(reg_fir_ap_ctrl);

    // - Can Add golden data compare on there (0x3800_0600)

    wb_write(checkbits, outputsignal[N-1] << 24 | 0x005A0000);
}
```

DMA Transfer Data

DMA

- wait for 0x38000100 to start.
- after start counter will plus 4 after ack
- 0x38000200 FIR Input
- 0x38000400 FIR Output

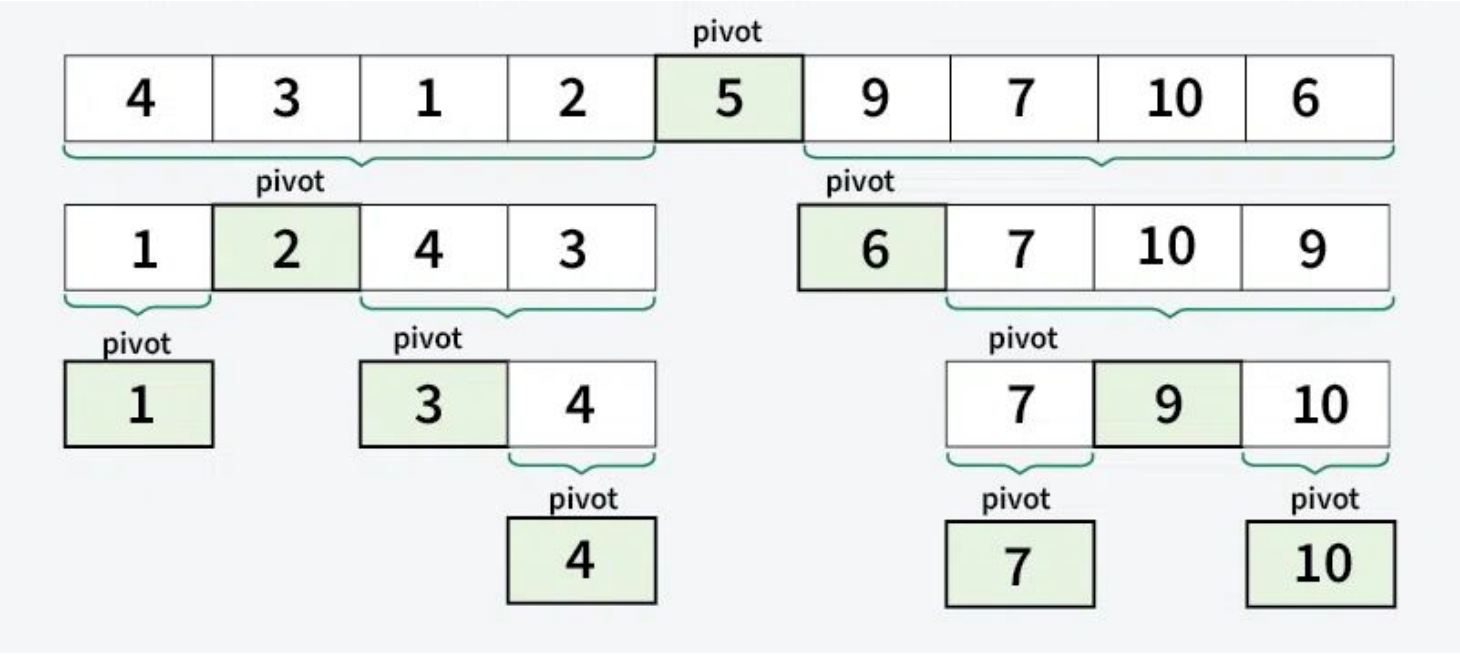
```
reg [(pDATA_WIDTH-1):0] x_addr_ptr;
reg [(pDATA_WIDTH-1):0] y_addr_ptr;
always @(posedge clk)
begin
    if(rst | (state==RESET))
    begin
        x_addr_ptr <= 0;
        y_addr_ptr <= 0;
    end
    else if (dma_ack_o & (state == XDATA))
        x_addr_ptr <= x_addr_ptr + 'd4;
    else if (dma_ack_o & (state == YDATA))
        y_addr_ptr <= y_addr_ptr + 'd4;
end
```

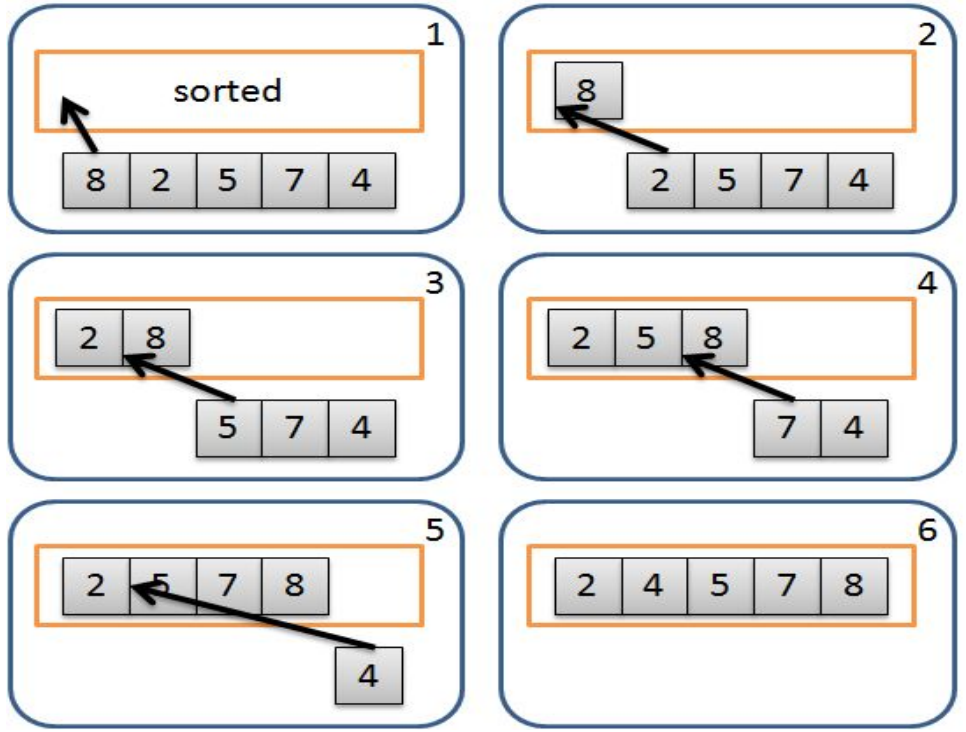


Quick Sort

- Sorting Algorithm
  - Quick sort
  - Insertion Sort

- Hardware Friendly
  - Serial Input is more suitable for insertion sort





Quick Sort

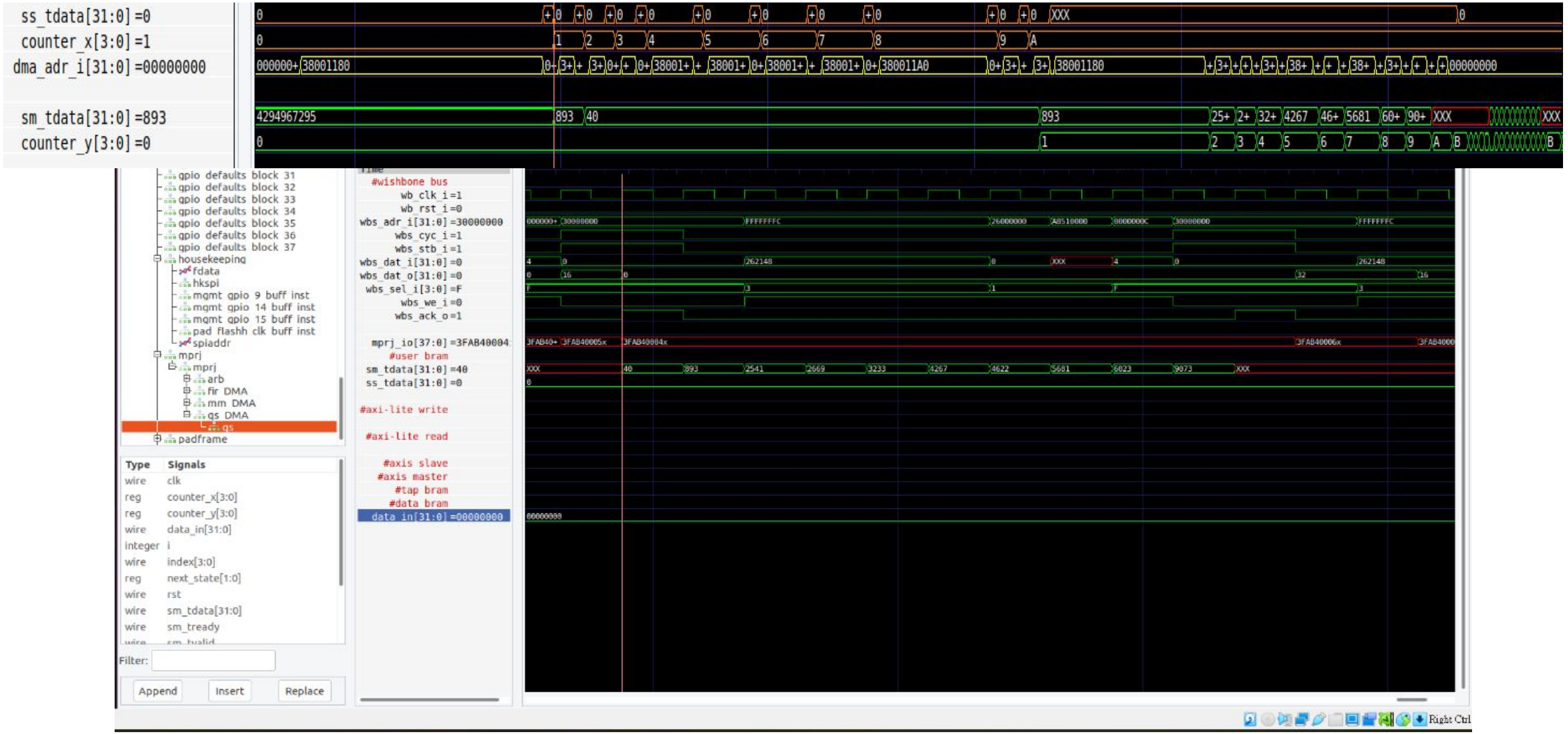
- FSM
- Insertion Compare Logic
- Sorted Data Logic

```
85 // Shift Register Update Logic
86 always @(posedge clk or posedge rst) begin
87     if (rst) begin
88         for (i = 0; i < 10; i = i + 1)
89             shift_reg_sorted[i] <= 32'hffff_ffff;
90     end
91     else begin
92         for (i = 0; i < 10; i = i + 1)
93             shift_reg_sorted[i] <= shift_reg[i];
94     end
95 end
96
97 // Sorting Logic
98 wire [3:0] index = (data_in < shift_reg_sorted[0]) ? 4'd0 :
99                    (data_in < shift_reg_sorted[1]) ? 4'd1 :
100                   (data_in < shift_reg_sorted[2]) ? 4'd2 :
101                   (data_in < shift_reg_sorted[3]) ? 4'd3 :
102                   (data_in < shift_reg_sorted[4]) ? 4'd4 :
103                   (data_in < shift_reg_sorted[5]) ? 4'd5 :
104                   (data_in < shift_reg_sorted[6]) ? 4'd6 :
105                   (data_in < shift_reg_sorted[7]) ? 4'd7 :
106                   (data_in < shift_reg_sorted[8]) ? 4'd8 : 4'd9;
```

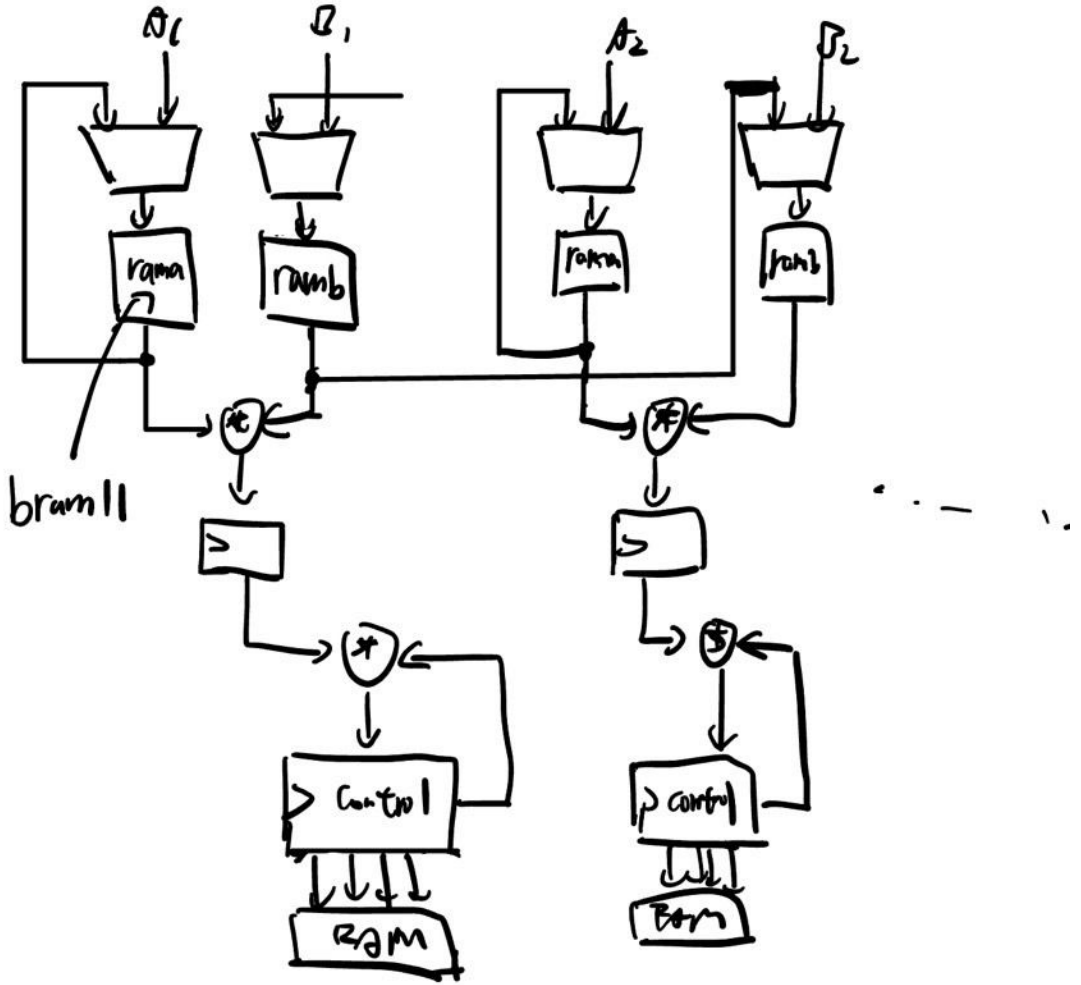
```
67 // Next State Logic
68 always @(*) begin
69     case (state)
70         S_RESET: next_state = S_IDLE;
71         S_IDLE: begin
72             if (counter_x == 4'd10)
73                 next_state = S_END;
74             else if (ss_tvalid)
75                 next_state = S_SHIFT;
76             else
77                 next_state = S_IDLE;
78         end
79         S_SHIFT: next_state = S_IDLE;
80         S_END:   next_state = (counter_y == 4'd10) ? S_RESET : S_END;
81         default: next_state = S_RESET;
82     endcase
83 end
```

```
110         case (index)
111             4'd0: begin
112                 shift_reg[0] = data_in;
113                 for (i = 1; i < 10; i = i + 1)
114                     shift_reg[i] = shift_reg_sorted[i - 1];
115             end
116             4'd1: begin
117                 for (i = 0; i < 1; i = i + 1)
118                     shift_reg[i] = shift_reg_sorted[i];
119                 shift_reg[1] = data_in;
120                 for (i = 2; i < 10; i = i + 1)
121                     shift_reg[i] = shift_reg_sorted[i - 1];
122             end
123             4'd2: begin
124                 for (i = 0; i < 2; i = i + 1)
125                     shift_reg[i] = shift_reg_sorted[i];
126                 shift_reg[2] = data_in;
127                 for (i = 3; i < 10; i = i + 1)
128                     shift_reg[i] = shift_reg_sorted[i - 1];
129             end
130             4'd3: begin
131                 for (i = 0; i < 3; i = i + 1)
132                     shift_reg[i] = shift_reg_sorted[i];
133                 shift_reg[3] = data_in;
134                 for (i = 4; i < 10; i = i + 1)
135                     shift_reg[i] = shift_reg_sorted[i - 1];
136             end
```

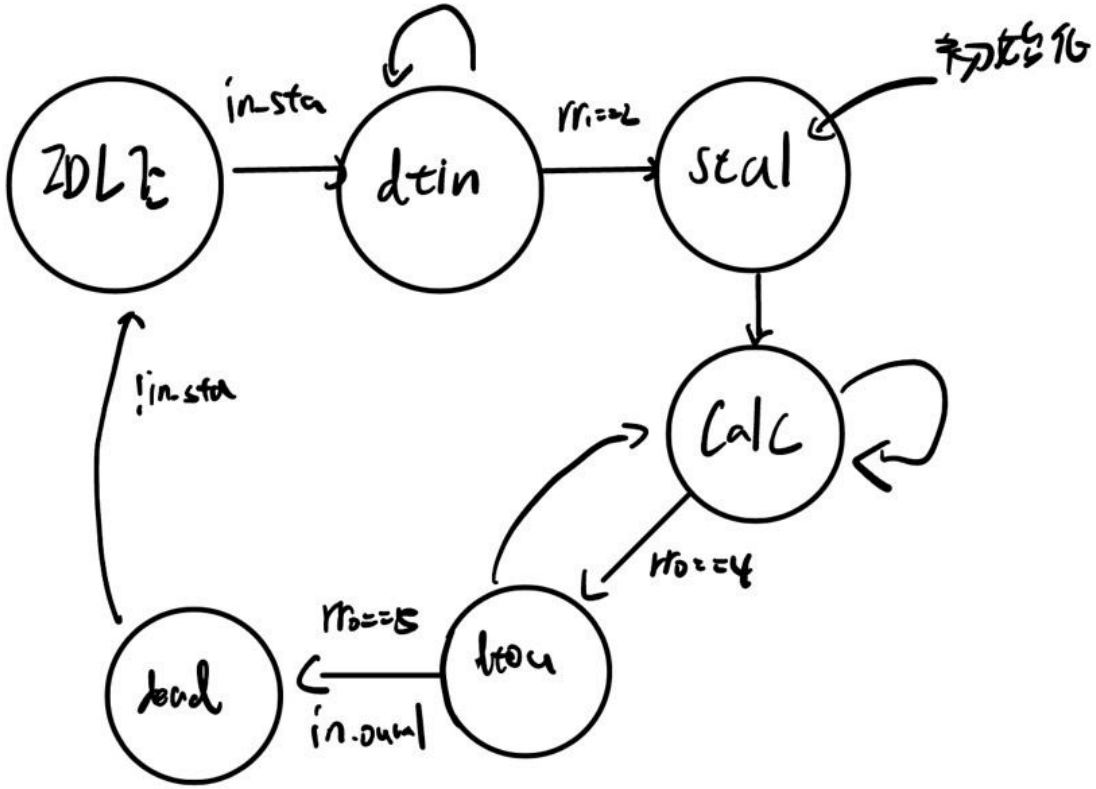
Quick Sort (Experiment)



Matmul



Matmul FSM



Matmul

```
MM Test passed
QS Test started
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0028
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x037d
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x09ed
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0a6d
QS Test passed
=====================================
All FIR/MM/QS and UART are finished!!
=====================================
```