SOC LAB#4-1 Report

Group no: 4

Members:

M11107410 羅善寬

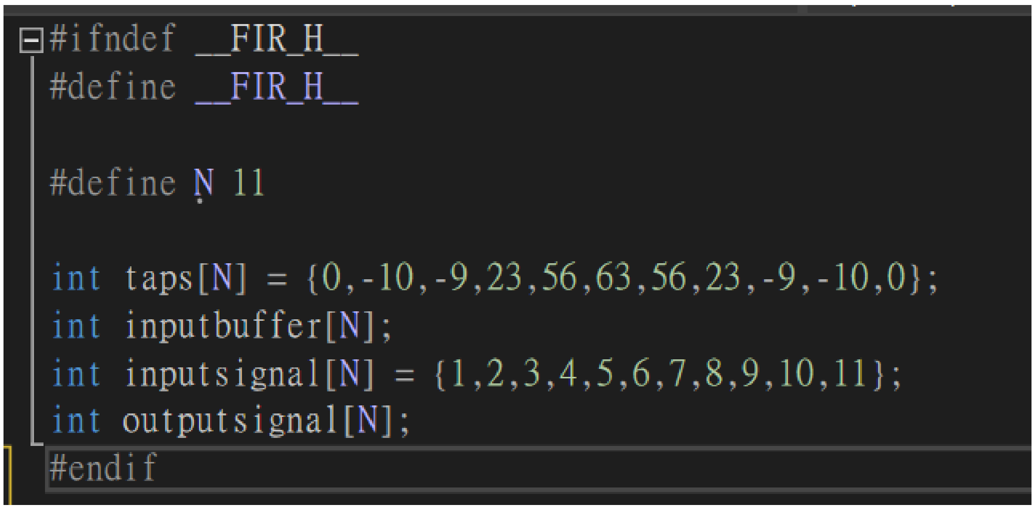
M11107004 曹榮恩

M11107409 陳昱碩

1. Prepare firmware code & RTL:

1.1 Generate data in header file – fir.h:

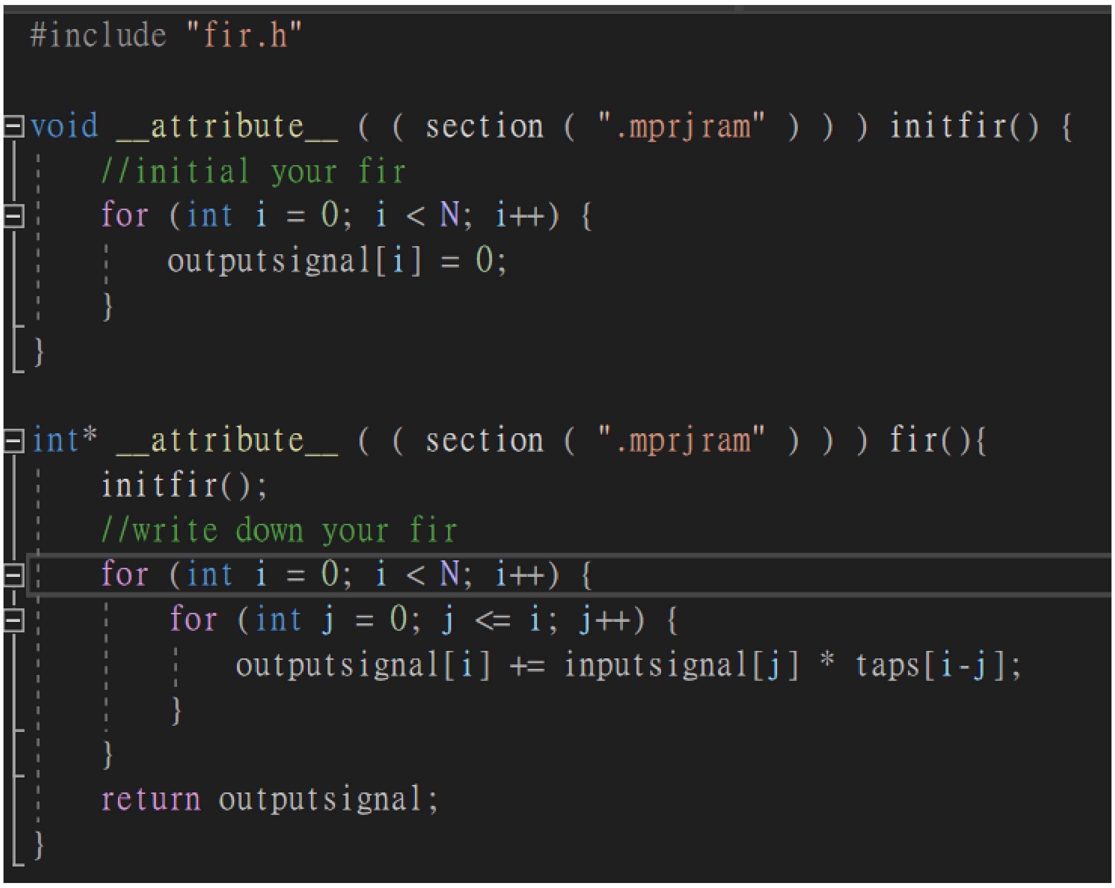
➢ Define taps parameters and inputsignal as lab3 in header file.



```c
#ifndef __FIR_H__
#define __FIR_H__

#define N 11

int taps[N] = {0,-10,-9,23,56,63,56,23,-9,-10,0};
int inputbuffer[N];
int inputsignal[N] = {1,2,3,4,5,6,7,8,9,10,11};
int outputsignal[N];
#endif
```

1.2 C code – fir.c:

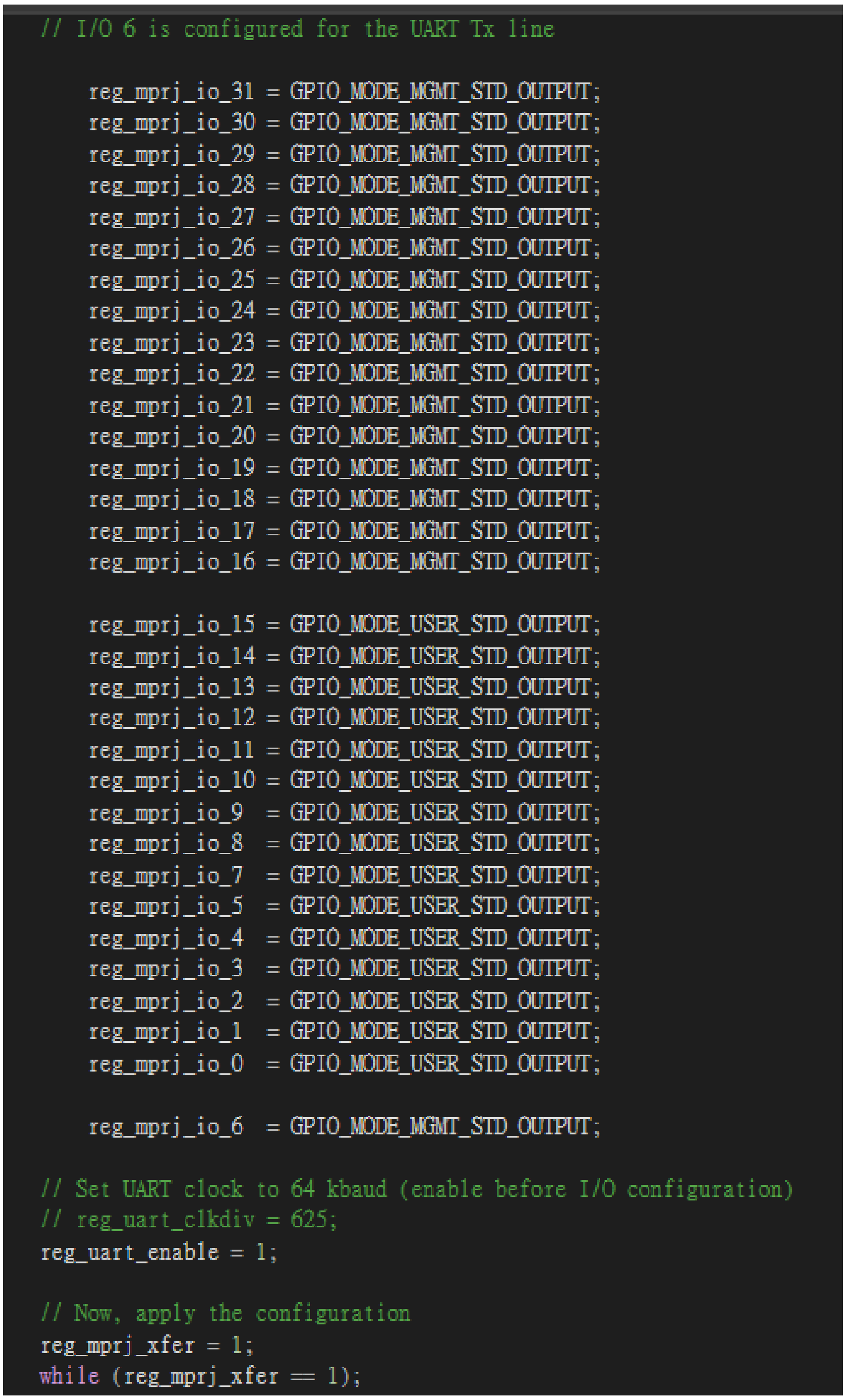
➢ Implement FIR function in c code.



```c
#include "fir.h"

void __attribute__ ( ( section ( ".mprjram" ) ) ) initfir() {
    //initial your fir
    for (int i = 0; i < N; i++) {
        outputsignal[i] = 0;
    }
}

int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir(){
    initfir();
    //write down your fir
    for (int i = 0; i < N; i++) {
        for (int j = 0; j <= i; j++) {
            outputsignal[i] += inputsignal[j] * taps[i-j];
        }
    }
    return outputsignal;
}
```

1.3 Firmware management in main(): (Already designed)

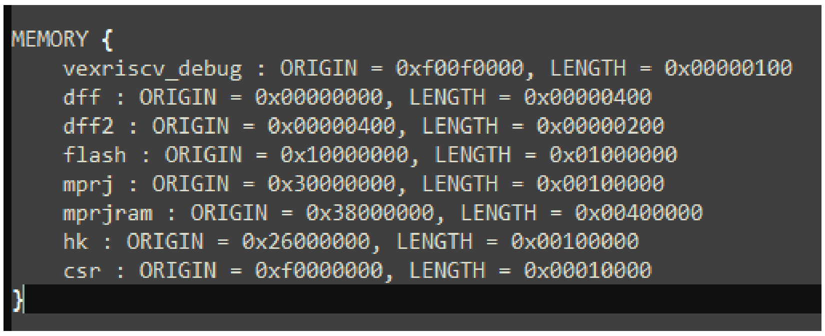
- In testbench/counter_la_fir.c, parameter reg_mprj_xfer will be initially to 1, and will not start fir until the external signal is given to 0.



```
// I/O 6 is configured for the UART Tx line

    reg_mprj_io_31 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_30 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_29 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_28 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_27 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_26 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_25 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_24 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_23 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_22 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_21 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_20 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_19 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_18 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_17 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_16 = GPIO_MODE_MGMT_STD_OUTPUT;

    reg_mprj_io_15 = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_14 = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_13 = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_12 = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_11 = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_10 = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_9  = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_8  = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_7  = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_5  = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_4  = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_3  = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_2  = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_1  = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_0  = GPIO_MODE_USER_STD_OUTPUT;

    reg_mprj_io_6  = GPIO_MODE_MGMT_STD_OUTPUT;

// Set UART clock to 64 kbaud (enable before I/O configuration)
// reg_uart_clkdiv = 625;
reg_uart_enable = 1;

// Now, apply the configuration
reg_mprj_xfer = 1;
while (reg_mprj_xfer == 1);
```

1.4 Linker for address arrangement: (Already designed)

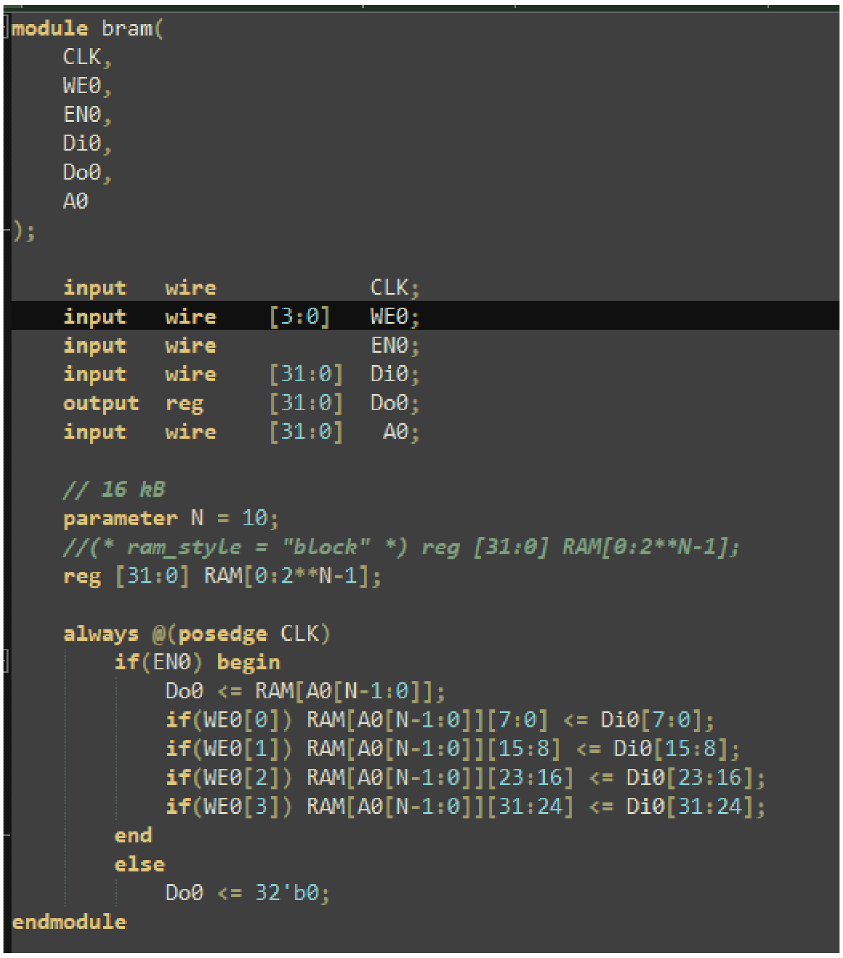
- In firmware/section.ids, mpjram is our bram, it's original address is at 0x38000000, and it's size is 4 KB.



```
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 = 0x00400000
    hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
    csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
}
```

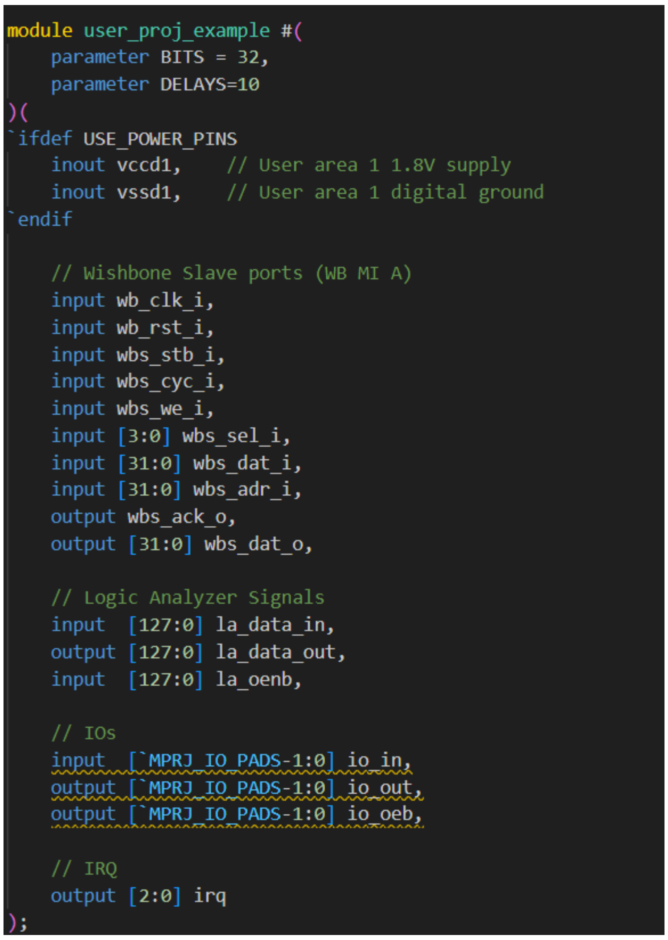
1.5 Design BRAM in user_project

- Estimated the required size of RAM

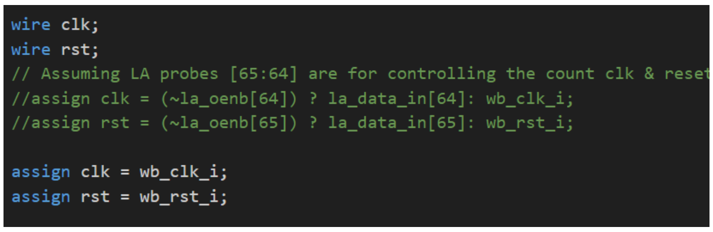


```
module bram(
    CLK,
    WE0,
    EN0,
    Di0,
    Do0,
    A0
);

    input   wire            CLK;
    input   wire    [3:0]   WE0;
    input   wire            EN0;
    input   wire    [31:0]  Di0;
    output  reg     [31:0]  Do0;
    input   wire    [31:0]   A0;

    // 16 kB
    parameter N = 10;
    //(* ram_style = "block" *) reg [31:0] RAM[0:2**N-1];
    reg [31:0] RAM[0:2**N-1];

    always @(posedge CLK)
        if(EN0) begin
            Do0 <= RAM[A0[N-1:0]];
            if(WE0[0]) RAM[A0[N-1:0]][7:0] <= Di0[7:0];
            if(WE0[1]) RAM[A0[N-1:0]][15:8] <= Di0[15:8];
            if(WE0[2]) RAM[A0[N-1:0]][23:16] <= Di0[23:16];
            if(WE0[3]) RAM[A0[N-1:0]][31:24] <= Di0[31:24];
        end
        else
            Do0 <= 32'b0;
endmodule
```

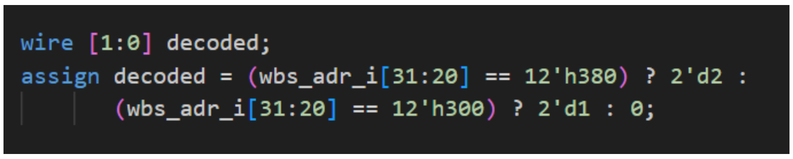
1.6 Design the controller connected with wishbone bus and ack response need to after Delay (10 delays)



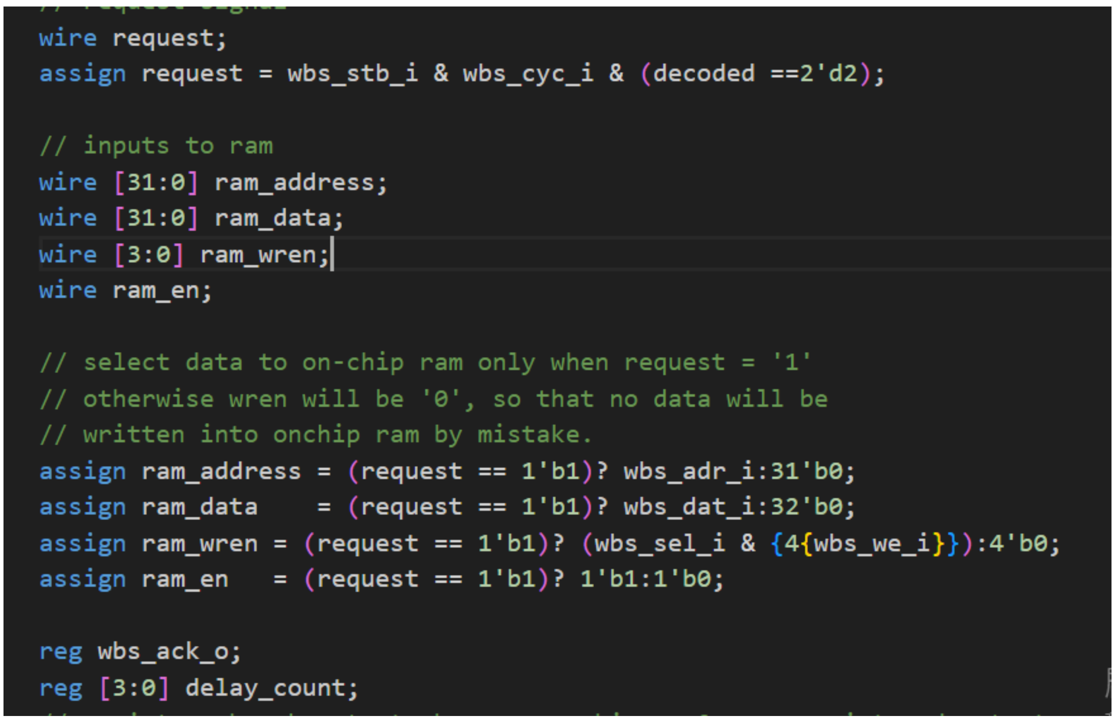
```
module user_proj_example #(
    parameter BITS = 32,
    parameter DELAYS=10
)(
`ifdef USE_POWER_PINS
    inout vccd1,    // User area 1 1.8V supply
    inout vssd1,    // User area 1 digital ground
`endif

    // Wishbone Slave ports (WB MI A)
    input wb_clk_i,
    input wb_rst_i,
    input wbs_stb_i,
    input wbs_cyc_i,
    input wbs_we_i,
    input [3:0] wbs_sel_i,
    input [31:0] wbs_dat_i,
    input [31:0] wbs_adr_i,
    output wbs_ack_o,
    output [31:0] wbs_dat_o,

    // Logic Analyzer Signals
    input  [127:0] la_data_in,
    output [127:0] la_data_out,
    input  [127:0] la_oenb,

    // IOs
    input  [`MPRJ_IO_PADS-1:0] io_in,
    output [`MPRJ_IO_PADS-1:0] io_out,
    output [`MPRJ_IO_PADS-1:0] io_oeb,

    // IRQ
    output [2:0] irq
);
```



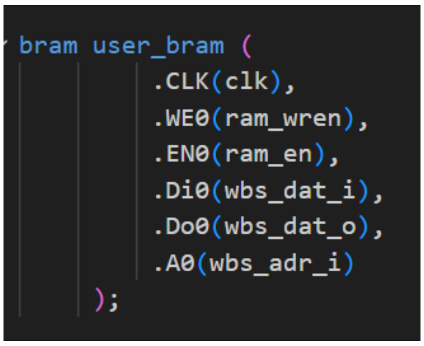
```
wire clk;
wire rst;
// Assuming LA probes [65:64] are for controlling the count clk & reset
//assign clk = (~la_oenb[64]) ? la_data_in[64]: wb_clk_i;
//assign rst = (~la_oenb[65]) ? la_data_in[65]: wb_rst_i;

assign clk = wb_clk_i;
assign rst = wb_rst_i;
```



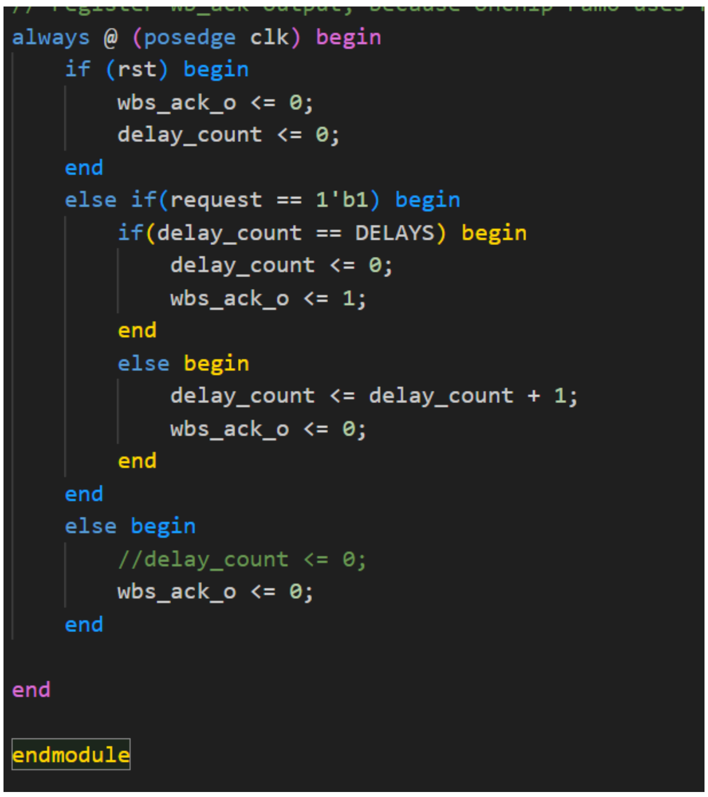
```
wire [1:0] decoded;
assign decoded = (wbs_adr_i[31:20] == 12'h380) ? 2'd2 :
        (wbs_adr_i[31:20] == 12'h300) ? 2'd1 : 0;
```



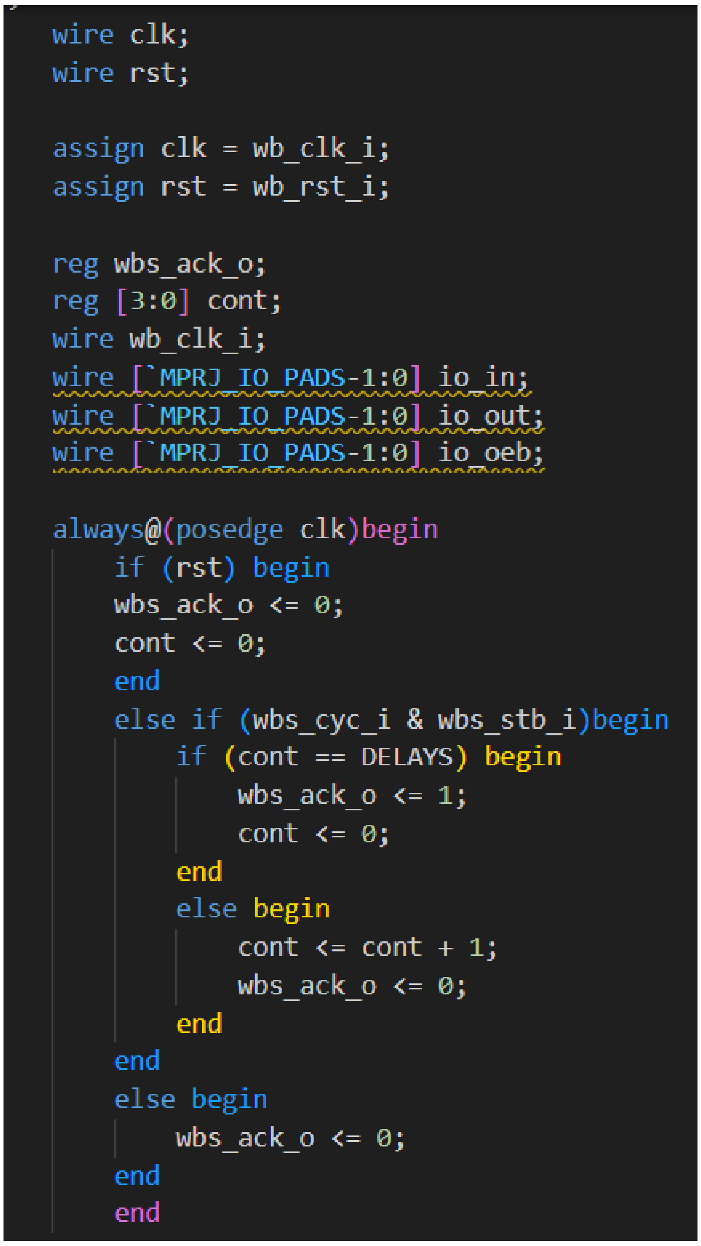
```verilog
wire request;
assign request = wbs_stb_i & wbs_cyc_i & (decoded ==2'd2);

// inputs to ram
wire [31:0] ram_address;
wire [31:0] ram_data;
wire [3:0] ram_wren;
wire ram_en;

// select data to on-chip ram only when request = '1'
// otherwise wren will be '0', so that no data will be
// written into onchip ram by mistake.
assign ram_address = (request == 1'b1)? wbs_adr_i:31'b0;
assign ram_data    = (request == 1'b1)? wbs_dat_i:32'b0;
assign ram_wren = (request == 1'b1)? (wbs_sel_i & {4{wbs_we_i}}):4'b0;
assign ram_en   = (request == 1'b1)? 1'b1:1'b0;

reg wbs_ack_o;
reg [3:0] delay_count;
```



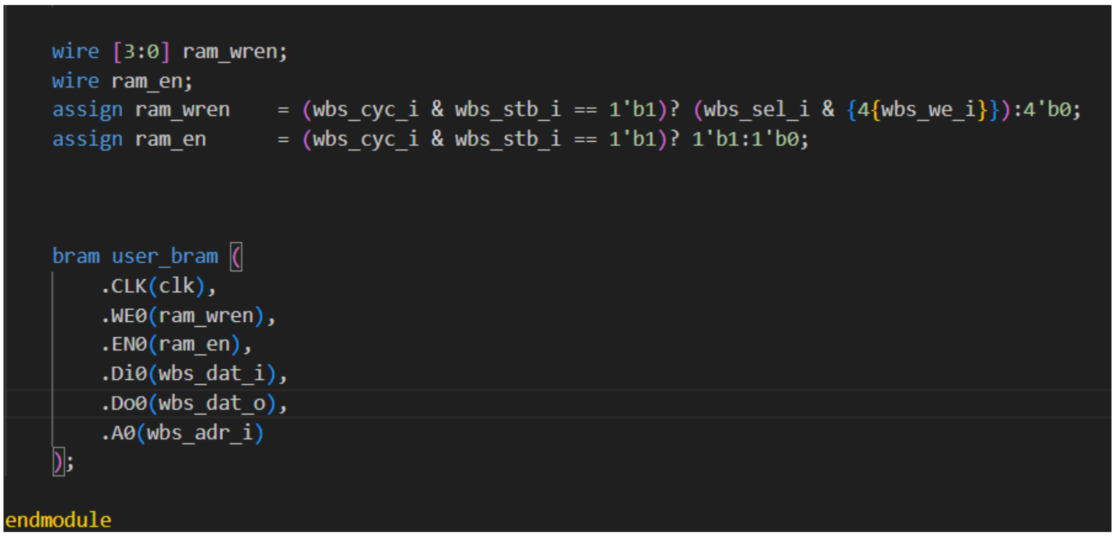
```verilog
bram user_bram (
        .CLK(clk),
        .WE0(ram_wren),
        .EN0(ram_en),
        .Di0(wbs_dat_i),
        .Do0(wbs_dat_o),
        .A0(wbs_adr_i)
    );
```



```verilog
always @ (posedge clk) begin
    if (rst) begin
        wbs_ack_o <= 0;
        delay_count <= 0;
    end
    else if(request == 1'b1) begin
        if(delay_count == DELAYS) begin
            delay_count <= 0;
            wbs_ack_o <= 1;
        end
        else begin
            delay_count <= delay_count + 1;
            wbs_ack_o <= 0;
        end
    end
    else begin
        //delay_count <= 0;
        wbs_ack_o <= 0;
    end

end

endmodule
```



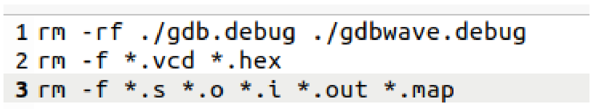
```
    wire clk;
    wire rst;

    assign clk = wb_clk_i;
    assign rst = wb_rst_i;

    reg wbs_ack_o;
    reg [3:0] cont;
    wire wb_clk_i;
    wire [`MPRJ_IO_PADS-1:0] io_in;
    wire [`MPRJ_IO_PADS-1:0] io_out;
    wire [`MPRJ_IO_PADS-1:0] io_oeb;

    always@(posedge clk)begin
        if (rst) begin
        wbs_ack_o <= 0;
        cont <= 0;
        end
        else if (wbs_cyc_i & wbs_stb_i)begin
            if (cont == DELAYS) begin
                wbs_ack_o <= 1;
                cont <= 0;
            end
            else begin
                cont <= cont + 1;
                wbs_ack_o <= 0;
            end
        end
        else begin
            wbs_ack_o <= 0;
        end
        end
```



```
    wire [3:0] ram_wren;
    wire ram_en;
    assign ram_wren    = (wbs_cyc_i & wbs_stb_i == 1'b1)? (wbs_sel_i & {4{wbs_we_i}}):4'b0;
    assign ram_en      = (wbs_cyc_i & wbs_stb_i == 1'b1)? 1'b1:1'b0;



    bram user_bram (
        .CLK(clk),
        .WE0(ram_wren),
        .EN0(ram_en),
        .Di0(wbs_dat_i),
        .Do0(wbs_dat_o),
        .A0(wbs_adr_i)
    );

endmodule
```

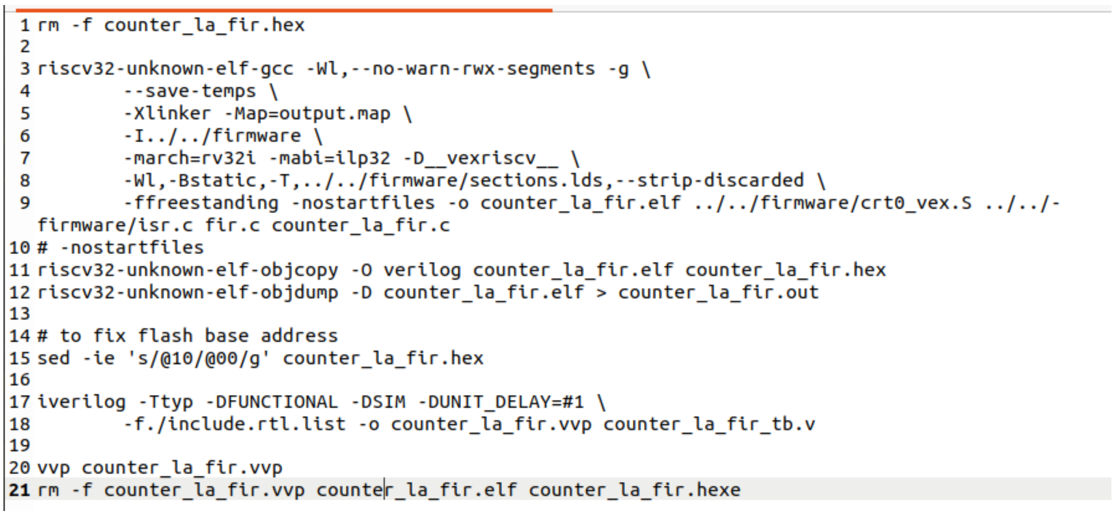
2. Compilation

2.1 Run_clean



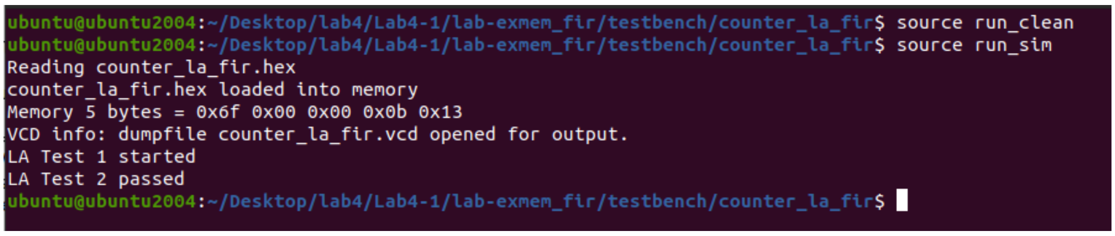
```
rm -rf ./gdb.debug ./gdbwave.debug
rm -f *.vcd *.hex
rm -f *.s *.o *.i *.out *.map
```

2.2 Run_sim



```
rm -f counter_la_fir.hex

riscv32-unknown-elf-gcc -Wl,--no-warn-rwx-segments -g \
        --save-temps \
        -Xlinker -Map=output.map \
        -I../../firmware \
        -march=rv32i -mabi=ilp32 -D__vexriscv__ \
        -Wl,-Bstatic,-T,../../firmware/sections.lds,--strip-discarded \
        -ffreestanding -nostartfiles -o counter_la_fir.elf ../../firmware/crt0_vex.S ../../firmware/isr.c fir.c counter_la_fir.c
# -nostartfiles
riscv32-unknown-elf-objcopy -O verilog counter_la_fir.elf counter_la_fir.hex
riscv32-unknown-elf-objdump -D counter_la_fir.elf > counter_la_fir.out

# to fix flash base address
sed -ie 's/@10/@00/g' counter_la_fir.hex

iverilog -Ttyp -DFUNCTIONAL -DSIM -DUNIT_DELAY=#1 \
        -f./include.rtl.list -o counter_la_fir.vvp counter_la_fir_tb.v

vvp counter_la_fir.vvp
rm -f counter_la_fir.vvp counter_la_fir.elf counter_la_fir.hexe
```

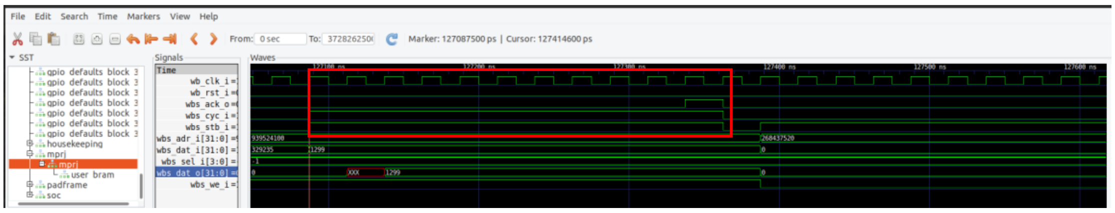
2.3 Compilation



```
ubuntu@ubuntu2004:~/Desktop/lab4/Lab4-1/lab-exmem_fir/testbench/counter_la_fir$ source run_clean
ubuntu@ubuntu2004:~/Desktop/lab4/Lab4-1/lab-exmem_fir/testbench/counter_la_fir$ source run_sim
Reading counter_la_fir.hex
counter_la_fir.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_fir.vcd opened for output.
LA Test 1 started
LA Test 2 passed
ubuntu@ubuntu2004:~/Desktop/lab4/Lab4-1/lab-exmem_fir/testbench/counter_la_fir$ 
```

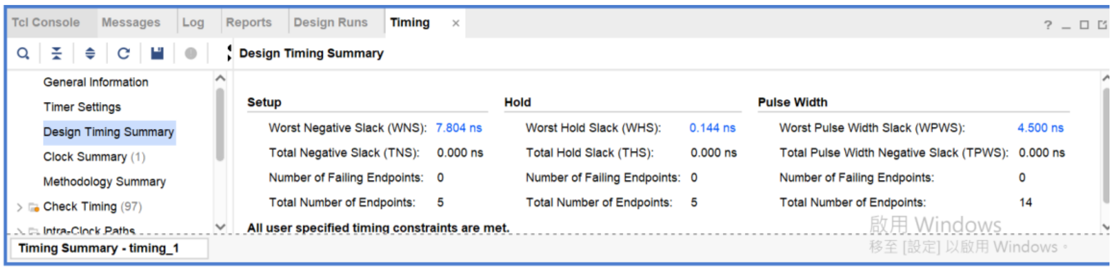
3. Synthesis & Verification

3.1 Waveform – Wishbone's ack will have a 10cycle delay when writing.



3.2 Timing report

- ➢ Operation system：Linux
- ➢ Priod：10ns



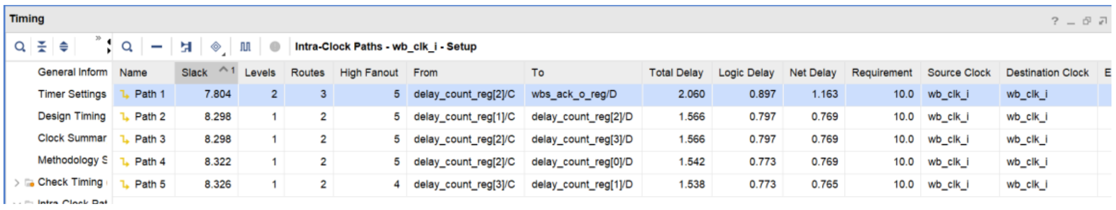
Design Timing Summary

General Information
Timer Settings
Design Timing Summary
Clock Summary (1)
Methodology Summary
Check Timing (97)
Intra-Clock Paths

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.804 ns	Worst Hold Slack (WHS):	0.144 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	5	Total Number of Endpoints:	5	Total Number of Endpoints:	14

All user specified timing constraints are met.

Timing Summary - timing_1


Timing

Intra-Clock Paths - wb_clk_i - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	7.804	2	3	5	delay_count_reg[2]/C	wbs_ack_o_reg/D	2.060	0.897	1.163	10.0	wb_clk_i	wb_clk_i
Path 2	8.298	1	2	5	delay_count_reg[1]/C	delay_count_reg[2]/D	1.566	0.797	0.769	10.0	wb_clk_i	wb_clk_i
Path 3	8.298	1	2	5	delay_count_reg[2]/C	delay_count_reg[3]/D	1.566	0.797	0.769	10.0	wb_clk_i	wb_clk_i
Path 4	8.322	1	2	5	delay_count_reg[2]/C	delay_count_reg[0]/D	1.542	0.773	0.769	10.0	wb_clk_i	wb_clk_i
Path 5	8.326	1	2	4	delay_count_reg[3]/C	delay_count_reg[1]/D	1.538	0.773	0.765	10.0	wb_clk_i	wb_clk_i

3.3 Synthesis report

```
1. Slice Logic
--------------

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |   21 |     0 |          0 |     53200 |  0.04 |
|   LUT as Logic          |   21 |     0 |          0 |     53200 |  0.04 |
|   LUT as Memory         |    0 |     0 |          0 |     17400 |  0.00 |
| Slice Registers         |    5 |     0 |          0 |    106400 | <0.01 |
|   Register as Flip Flop |    5 |     0 |          0 |    106400 | <0.01 |
|   Register as Latch     |    0 |     0 |          0 |    106400 |  0.00 |
| F7 Muxes                |    0 |     0 |          0 |     26600 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |     13300 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

```
---------------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
	   2 Input    4 Bit       Adders := 1
+---Registers :
	               32 Bit    Registers := 1
	                4 Bit    Registers := 1
	                1 Bit    Registers := 1
+---RAMs :
	              128K Bit	(4096 X 32 bit)          RAMs := 1
+---Muxes :
	   2 Input   32 Bit        Muxes := 6
	   2 Input    8 Bit        Muxes := 1
	   2 Input    4 Bit        Muxes := 2
	   3 Input    1 Bit        Muxes := 1
	   2 Input    1 Bit        Muxes := 2
---------------------------------------------------------------------------------
Finished RTL Component Statistics
```