SoC Lab4-1 Report

312510140 何律明
312410220 張承宗

Explanation of your firmware code:

fir.c

```c
#include "fir.h"

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

int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir() {
        initfir();
        //write down your fir
        for (int i=0; i<N ; i++){
                // shift
                inputbuffer[10] = inputbuffer[9];
                inputbuffer[9] = inputbuffer[8];
                inputbuffer[8] = inputbuffer[7];
                inputbuffer[7] = inputbuffer[6];
                inputbuffer[6] = inputbuffer[5];
                inputbuffer[5] = inputbuffer[4];
                inputbuffer[4] = inputbuffer[3];
                inputbuffer[3] = inputbuffer[2];
                inputbuffer[2] = inputbuffer[1];
                inputbuffer[1] = inputbuffer[0];

                // give input
                inputbuffer[0] = inputsignal[i];

                // calculate fir
                for (int j=0; j<N; j++) {
                        outputsignal[i] = outputsignal[i] + (inputbuffer[j] * taps[j]);
                }
        }

        return outputsignal;
}
```

就是模擬 shift ram 然後和對應的 tap parameter 相乘累加起來就是正確的答案。

Explanation for assembly code:

counter_la_fir.out

```
10000020 <trap_entry>:
10000020: fe112e23            sw  ra,-4(sp)
10000024: fe512c23            sw  t0,-8(sp)
10000028: fe612a23            sw  t1,-12(sp)
1000002c: fe712823            sw  t2,-16(sp)
10000030: fea12623            sw  a0,-20(sp)
10000034: feb12423            sw  a1,-24(sp)
10000038: fec12223            sw  a2,-28(sp)
1000003c: fed12023            sw  a3,-32(sp)
10000040: fce12e23            sw  a4,-36(sp)
10000044: fcf12c23            sw  a5,-40(sp)
10000048: fd012a23            sw  a6,-44(sp)
1000004c: fd112823            sw  a7,-48(sp)
10000050: fdc12623            sw  t3,-52(sp)
10000054: fdd12423            sw  t4,-56(sp)
10000058: fde12223            sw  t5,-60(sp)
1000005c: fdf12023            sw  t6,-64(sp)
10000060: fc010113            addi  sp,sp,-64
10000064: 140000ef            jal ra,100001a4 <isr>
10000068: 03c12083            lw  ra,60(sp)
1000006c: 03812283            lw  t0,56(sp)
10000070: 03412303            lw  t1,52(sp)
10000074: 03012383            lw  t2,48(sp)
10000078: 02c12503            lw  a0,44(sp)
1000007c: 02812583            lw  a1,40(sp)
10000080: 02412603            lw  a2,36(sp)
10000084: 02012683            lw  a3,32(sp)
10000088: 01c12703            lw  a4,28(sp)
1000008c: 01812783            lw  a5,24(sp)
10000090: 01412803            lw  a6,20(sp)
10000094: 01012883            lw  a7,16(sp)
10000098: 00c12e03            lw  t3,12(sp)
1000009c: 00812e83            lw  t4,8(sp)
100000a0: 00412f03            lw  t5,4(sp)
100000a4: 00012f83            lw  t6,0(sp)
100000a8: 04010113            addi  sp,sp,64
100000ac: 30200073            mret
```

這段是把暫存器給存入 data 作為 init。

```
100002ec <main>:
100002ec: fe010113          addi  sp,sp,-32
100002f0: 00112e23          sw  ra,28(sp)
100002f4: 00812c23          sw  s0,24(sp)
100002f8: 02010413          addi  s0,sp,32
100002fc: 260007b7          lui a5,0x26000
10000300: 0a078793          addi  a5,a5,160 # 260000a0 <_esram_rom+0x15fff898>
10000304: 00002737          lui a4,0x2
10000308: 80970713          addi  a4,a4,-2039 # 1809 <_fstack+0x1209>
1000030c: 00e7a023          sw  a4,0(a5)
10000310: 260007b7          lui a5,0x26000
10000314: 09c78793          addi  a5,a5,156 # 2600009c <_esram_rom+0x15fff894>
10000318: 00002737          lui a4,0x2
1000031c: 80970713          addi  a4,a4,-2039 # 1809 <_fstack+0x1209>
10000320: 00e7a023          sw  a4,0(a5)
10000324: 260007b7          lui a5,0x26000
10000328: 09878793          addi  a5,a5,152 # 26000098 <_esram_rom+0x15fff890>
1000032c: 00002737          lui a4,0x2
10000330: 80970713          addi  a4,a4,-2039 # 1809 <_fstack+0x1209>
10000334: 00e7a023          sw  a4,0(a5)
10000338: 260007b7          lui a5,0x26000
1000033c: 09478793          addi  a5,a5,148 # 26000094 <_esram_rom+0x15fff88c>
10000340: 00002737          lui a4,0x2
10000344: 80970713          addi  a4,a4,-2039 # 1809 <_fstack+0x1209>
10000348: 00e7a023          sw  a4,0(a5)
1000034c: 260007b7          lui a5,0x26000
10000350: 09078793          addi  a5,a5,144 # 26000090 <_esram_rom+0x15fff888>
10000354: 00002737          lui a4,0x2
10000358: 80970713          addi  a4,a4,-2039 # 1809 <_fstack+0x1209>
1000035c: 00e7a023          sw  a4,0(a5)
10000360: 260007b7          lui a5,0x26000
10000364: 08c78793          addi  a5,a5,140 # 2600008c <_esram_rom+0x15fff884>
10000368: 00002737          lui a4,0x2
1000036c: 80970713          addi  a4,a4,-2039 # 1809 <_fstack+0x1209>
10000370: 00e7a023          sw  a4,0(a5)
10000374: 260007b7          lui a5,0x26000
10000378: 08878793          addi  a5,a5,136 # 26000088 <_esram_rom+0x15fff880>
1000037c: 00002737          lui a4,0x2
10000380: 80970713          addi  a4,a4,-2039 # 1809 <_fstack+0x1209>
10000384: 00e7a023          sw  a4,0(a5)
10000388: 260007b7          lui a5,0x26000
1000038c: 08478793          addi  a5,a5,132 # 26000084 <_esram_rom+0x15fff87c>
10000390: 00002737          lui a4,0x2
10000394: 80970713          addi  a4,a4,-2039 # 1809 <_fstack+0x1209>
10000398: 00e7a023          sw  a4,0(a5)
1000039c: 260007b7          lui a5,0x26000
```

這段就是遵照 user_counter 的宣告去計算對應 address 並進行讀取，也就是 reg_mprj_io 那幾根 pin 的對應位置。

對應的 C cod e(counter_la_fir.c, firmware code)為

```
// logic analyzer probes.
// 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;
```

```
3800008c <fir>:
3800008c: fe010113          addi  sp,sp,-32
38000090: 00112e23          sw  ra,28(sp)
38000094: 00812c23          sw  s0,24(sp)
38000098: 00912a23          sw  s1,20(sp)
3800009c: 02010413          addi  s0,sp,32
380000a0: f85ff0ef          jal ra,38000024 <initfir>
380000a4: fe042623          sw  zero,-20(s0)
380000a8: 1500006f          j 380001f8 <fir+0x16c>
380000ac: 05c00793          li  a5,92
380000b0: 0247a703          lw  a4,36(a5)
380000b4: 05c00793          li  a5,92
380000b8: 02e7a423          sw  a4,40(a5)
380000bc: 05c00793          li  a5,92
380000c0: 0207a703          lw  a4,32(a5)
380000c4: 05c00793          li  a5,92
380000c8: 02e7a223          sw  a4,36(a5)
380000cc: 05c00793          li  a5,92
380000d0: 01c7a703          lw  a4,28(a5)
380000d4: 05c00793          li  a5,92
380000d8: 02e7a023          sw  a4,32(a5)
380000dc: 05c00793          li  a5,92
380000e0: 0187a703          lw  a4,24(a5)
380000e4: 05c00793          li  a5,92
380000e8: 00e7ae23          sw  a4,28(a5)
380000ec: 05c00793          li  a5,92
380000f0: 0147a703          lw  a4,20(a5)
380000f4: 05c00793          li  a5,92
380000f8: 00e7ac23          sw  a4,24(a5)
380000fc: 05c00793          li  a5,92
38000100: 0107a703          lw  a4,16(a5)
38000104: 05c00793          li  a5,92
38000108: 00e7aa23          sw  a4,20(a5)
3800010c: 05c00793          li  a5,92
38000110: 00c7a703          lw  a4,12(a5)
38000114: 05c00793          li  a5,92
38000118: 00e7a823          sw  a4,16(a5)
3800011c: 05c00793          li  a5,92
38000120: 0087a703          lw  a4,8(a5)
38000124: 05c00793          li  a5,92
38000128: 00e7a623          sw  a4,12(a5)
3800012c: 05c00793          li  a5,92
38000130: 0047a703          lw  a4,4(a5)
38000134: 05c00793          li  a5,92
38000138: 00e7a423          sw  a4,8(a5)
3800013c: 05c00793          li  a5,92
38000140: 0007a703          lw  a4,0(a5)
```

```
684   38000144: 05c00793          li  a5,92
685   38000148: 00e7a223          sw  a4,4(a5)
686   3800014c: 02c00713          li  a4,44
687   38000150: fec42783          lw  a5,-20(s0)
688   38000154: 00279793          slli  a5,a5,0x2
689   38000158: 00f707b3          add a5,a4,a5
690   3800015c: 0007a703          lw  a4,0(a5)
691   38000160: 05c00793          li  a5,92
692   38000164: 00e7a023          sw  a4,0(a5)
693   38000168: fe042423          sw  zero,-24(s0)
694   3800016c: 0740006f          j 380001e0 <fir+0x154>
695   38000170: 08800713          li  a4,136
696   38000174: fec42783          lw  a5,-20(s0)
697   38000178: 00279793          slli  a5,a5,0x2
698   3800017c: 00f707b3          add a5,a4,a5
699   38000180: 0007a483          lw  s1,0(a5)
700   38000184: 05c00713          li  a4,92
701   38000188: fe842783          lw  a5,-24(s0)
702   3800018c: 00279793          slli  a5,a5,0x2
703   38000190: 00f707b3          add a5,a4,a5
704   38000194: 0007a683          lw  a3,0(a5)
705   38000198: 00000713          li  a4,0
706   3800019c: fe842783          lw  a5,-24(s0)
707   380001a0: 00279793          slli  a5,a5,0x2
708   380001a4: 00f707b3          add a5,a4,a5
709   380001a8: 0007a783          lw  a5,0(a5)
710   380001ac: 00078593          mv  a1,a5
711   380001b0: 00068513          mv  a0,a3
712   380001b4: e4dff0ef          jal ra,38000000 <__mulsi3>
713   380001b8: 00050793          mv  a5,a0
714   380001bc: 00f48733          add a4,s1,a5
715   380001c0: 08800693          li  a3,136
716   380001c4: fec42783          lw  a5,-20(s0)
717   380001c8: 00279793          slli  a5,a5,0x2
718   380001cc: 00f687b3          add a5,a3,a5
719   380001d0: 00e7a023          sw  a4,0(a5)
720   380001d4: fe842783          lw  a5,-24(s0)
721   380001d8: 00178793          addi  a5,a5,1
722   380001dc: fef42423          sw  a5,-24(s0)
723   380001e0: fe842703          lw  a4,-24(s0)
724   380001e4: 00a00793          li  a5,10
```

```
730   380001fc: 00a00793          li  a5,10
731   38000200: eae7d6e3          bge a5,a4,380000ac <fir+0x20>
732   38000204: 08800793          li  a5,136
733   38000208: 00078513          mv  a0,a5
734   3800020c: 01c12083          lw  ra,28(sp)
735   38000210: 01812403          lw  s0,24(sp)
736   38000214: 01412483          lw  s1,20(sp)
737   38000218: 02010113          addi  sp,sp,32
738   3800021c: 00008067          ret
```

這塊主要是 fir.c 的 function 實踐 FIR code 他會先將 a5 和 a4 先讀出來然後跳到 380001fc 去比較，基本上就是看算到 data length 了沒。

如果 a5 比 a4 大就回到上面繼續讀寫乘加(上圖 731 行)，這樣就能進行迴圈來達到目的。

How does it execute a multiplication in assembly code:

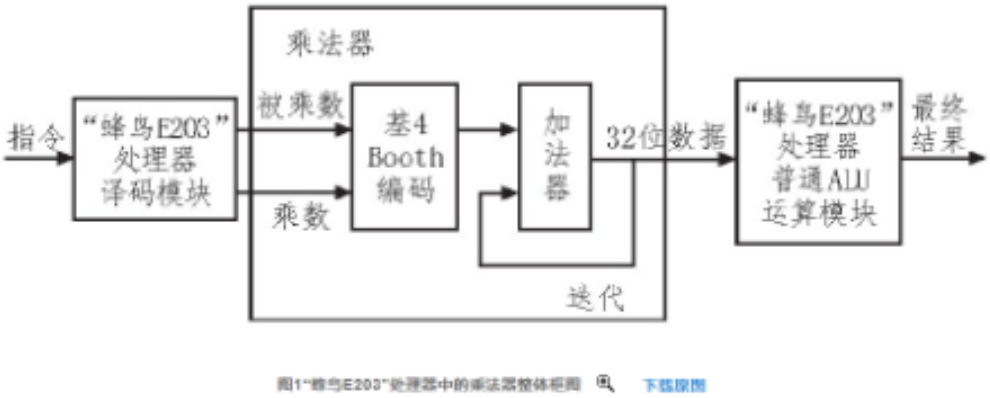


图1“蜂鸟E203”处理器中的乘法器整体框图 下载原图

```
598   38000000 <__mulsi3>:
599   38000000: 00050613          mv  a2,a0
600   38000004: 00000513          li  a0,0
601   38000008: 0015f693          andi  a3,a1,1
602   3800000c: 00068463          beqz  a3,38000014 <__mulsi3+0x14>
603   38000010: 00c50533          add a0,a0,a2
604   38000014: 0015d593          srli  a1,a1,0x1
605   38000018: 00161613          slli  a2,a2,0x1
606   3800001c: fe0596e3          bnez  a1,38000008 <__mulsi3+0x8>
607   38000020: 00008067          ret
```

整篇 code 都沒有使用到 mult 這種乘法指令，mult 改用 function 的方式實踐。是因為 caravel SoC 的 ALU 不支援乘法，那乘法就得加法器進行(慢慢累加)。可以看到 add a0, a0, a2 在做累加的動作。

What address allocate for user project and how many space is required to allocate to firmware code:

根據 assembly code 可以看到 fir()的空間是 0x38000000 到 0x3800021C，
一共 540 byte。

```
596   Disassembly of section .mprjram:
597
598   38000000 <__mulsi3>:
```

```
738   3800021c: 00008067          ret
```

counter_la_fir.c 從 1000_0000 ~ 1000_07a8 總共 1960 byte。

```
536   100007a8: 00008067          ret
```

```
5   Disassembly of section .text:
6
7   10000000 <_ftext>:
```

counter_la_fir.c 的 main() 在 1000_02ec 的位置開始

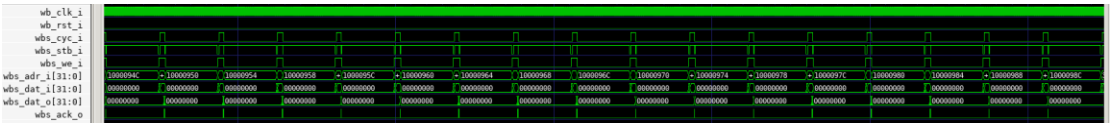
```
232   100002ec <main>:
```

以上主要是 code 所佔的存放空間，不包含 data 的空間。

Interface between BRAM and wishnone:

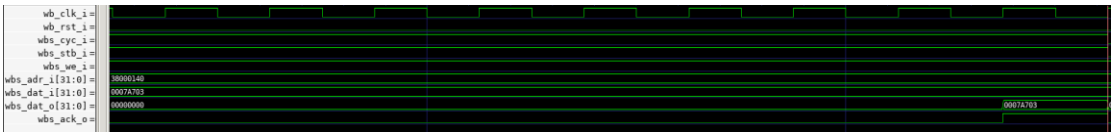
Waveform from xsim:

指定 0x3800000xx 的位址進行讀寫

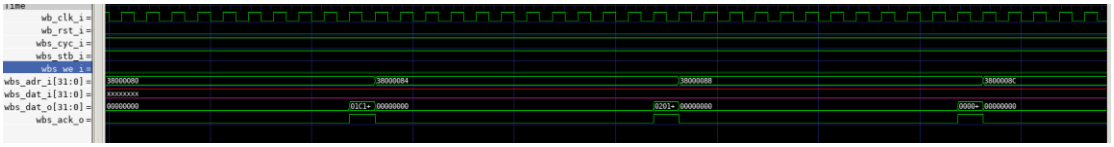


放大就能看得出來是用 cyc、stb、和 ack 進行 handshake。

Write：



Read：



Hardware:

```
wire [BITS-1:0] bram_do;
wire [BITS-1:0] bram_di;
wire [BITS-1:0] bram_adr;
wire bram_valid;
wire [3:0] bram_we;

assign bram_valid   = wbs_stb_i == 1 && wbs_cyc_i == 1 && wbs_adr_i[31:24] == 'h38;
assign bram_we      = {4{wbs_we_i & bram_valid}};
assign bram_adr     = (wbs_adr_i - 'h38000000) >> 2;
assign bram_di      = wbs_dat_i;


bram user_bram (
    .CLK(wb_clk_i),
    .WE0(bram_we),
    .EN0(bram_valid),
    .Di0(bram_di),
    .Do0(bram_do),
    .A0(bram_adr)
);


reg [15:0] counter; // why is this 16 bits

always @(posedge wb_clk_i) begin
    if (wb_rst_i) begin
        counter     <= 0;
        wbs_ack_o   <= 0;
        wbs_dat_o   <= 0;
    end else begin
        counter     <= (counter == DELAYS) ? 0 : (bram_valid == 1 && wbs_ack_o != 1) ? counter + 1 : counter;
        wbs_ack_o   <= (counter == DELAYS) ? 1 : 0;
        wbs_dat_o   <= (counter == DELAYS) ? bram_do : 0;
    end
end
```

單純上圖就可以做到經由 wishbone 送進來的 transaction request 向我們的 bram 拿資料的行為。簡單的 handshake 還不需要用到 fsm。

Synthesis report:

Synthesis report 檔案連結:https://github.com/holyuming/NYCU-2023-SoCLab/blob/master/lab-exmem_fir/synthesis/synthesis.runs/synth_1/user_project_wrapper_utilization_synth.rpt

1. CLB Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
CLB LUTs*	42	0	0	230400	0.02
LUT as Logic	42	0	0	230400	0.02
LUT as Memory	0	0	0	101760	0.00
CLB Registers	49	0	0	460800	0.01
Register as Flip Flop	49	0	0	460800	0.01
Register as Latch	0	0	0	460800	0.00
CARRY8	0	0	0	28800	0.00
F7 Muxes	0	0	0	115200	0.00
F8 Muxes	0	0	0	57600	0.00
F9 Muxes	0	0	0	28800	0.00

BRAM：

```
2. BLOCKRAM
-----------

+-------------------+------+-------+------------+-----------+-------+
|     Site Type     | Used | Fixed | Prohibited | Available | Util% |
+-------------------+------+-------+------------+-----------+-------+
| Block RAM Tile    |  0.5 |     0 |          0 |       312 |  0.16 |
|   RAMB36/FIFO*    |    0 |     0 |          0 |       312 |  0.00 |
|   RAMB18          |    1 |     0 |          0 |       624 |  0.16 |
|     RAMB18E2 only |    1 |       |            |           |       |
| URAM              |    0 |     0 |          0 |        96 |  0.00 |
+-------------------+------+-------+------------+-----------+-------+
```

IO & arithemetic：

```
3. ARITHMETIC
-------------

+-----------+------+-------+------------+-----------+-------+
| Site Type | Used | Fixed | Prohibited | Available | Util% |
+-----------+------+-------+------------+-----------+-------+
| DSPs      |    0 |     0 |          0 |      1728 |  0.00 |
+-----------+------+-------+------------+-----------+-------+


4. I/O
------

+------------+------+-------+------------+-----------+-------+
|  Site Type | Used | Fixed | Prohibited | Available | Util% |
+------------+------+-------+------------+-----------+-------+
| Bonded IOB |  282 |     0 |          0 |       360 | 78.33 |
+------------+------+-------+------------+-----------+-------+
```

會這樣應該是因為 lab4-1 要求合成的是只有作為 delay 存在的 user_counter 和 bram，所以用不到運算單元，固沒有 DSPs 被合成，但 IO 的部分就用了很多 port。

Primitives：

8. Primitives

Ref Name	Used	Functional Category
OBUFT	195	I/O
INBUF	54	I/O
IBUFCTRL	54	Others
FDRE	49	Register
LUT2	37	CLB
OBUF	33	I/O
LUT3	13	CLB
LUT4	7	CLB
LUT6	6	CLB
LUT5	5	CLB
RAMB18E2	1	BLOCKRAM
LUT1	1	CLB
BUFGCE	1	Clock

Timing report 檔案連結: https://github.com/holyuming/NYCU-2023-SoCLab/blob/master/lab-exmem_fir/synthesis/timing_report.txt

Clock	Waveform(ns)	Period(ns)	Frequency(MHz)
wb_clk_i	{0.000 2.500}	5.000	200.000

我們是用 period = 5 合成

From Clock: wb_clk_i
To Clock: wb_clk_i

Setup :	0	Failing Endpoints,	Worst Slack	3.622ns,	Total Violation	0.000ns
Hold :	NA	Failing Endpoints,	Worst Slack	NA ,	Total Violation	NA
PW :	0	Failing Endpoints,	Worst Slack	1.958ns,	Total Violation	0.000ns

Slack MET 並沒有 timing violation 的狀況，畢竟我們的 counter.v 只有對 bram 拿資料的動作而且 data 都是從 bram FF 輸出，所以 delay 很短。這還不是最短的 clock period。應該可以跑更快(counter.v)。