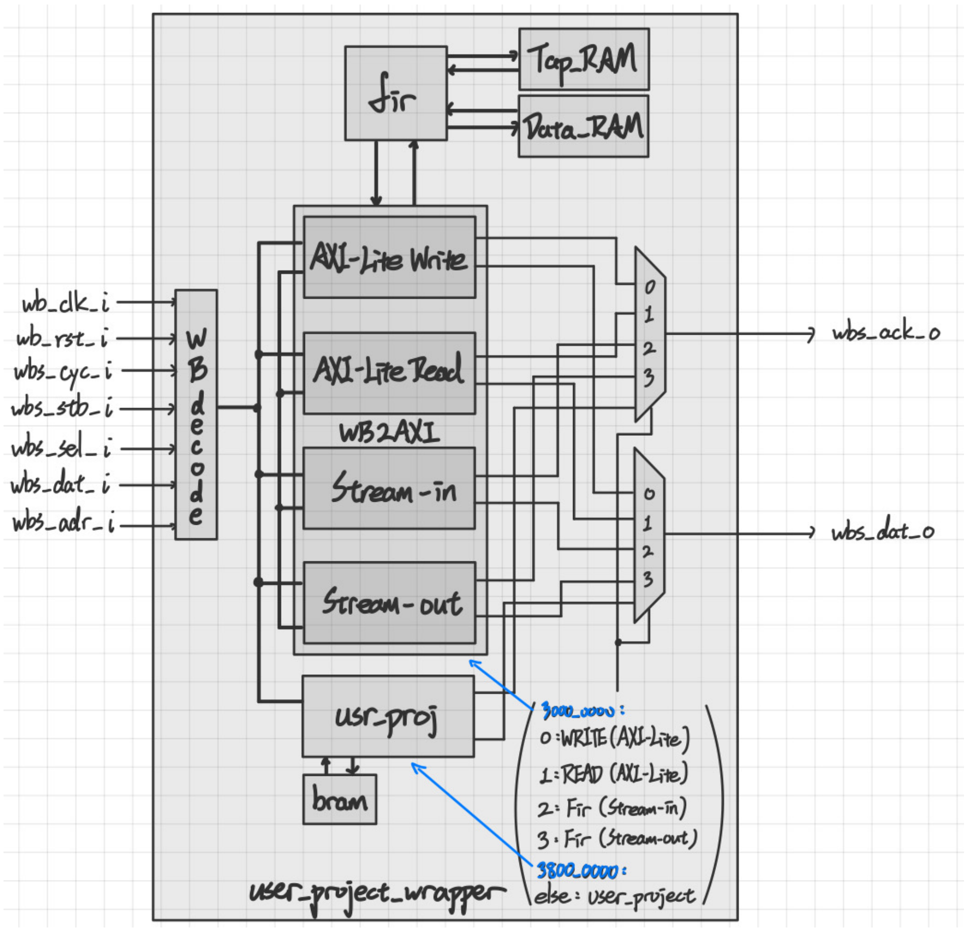
SOC course_lab4-2 Report

110590022 陳冠晰 110000107 陳柏翰 110061217 王彥智

1. **Design block diagram – Datapath, control-path**

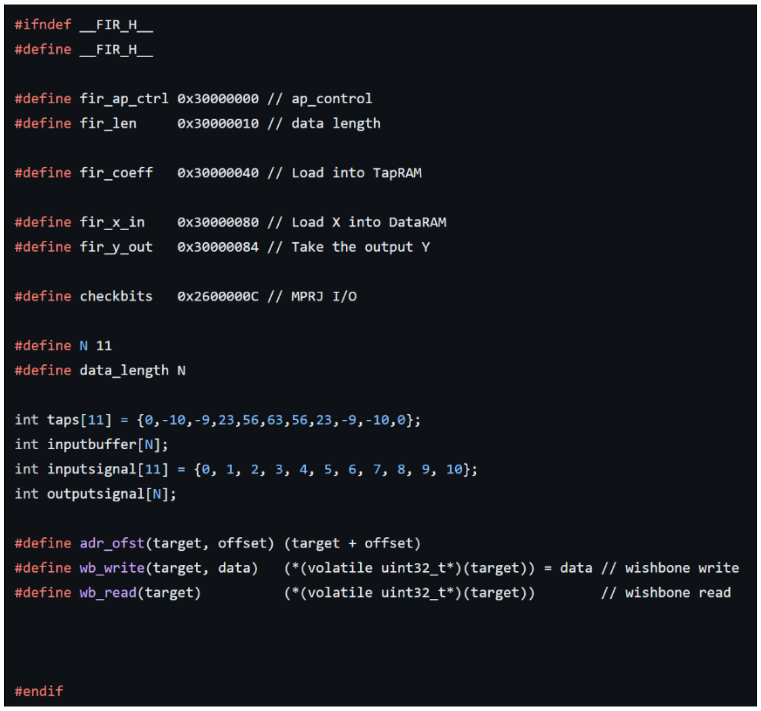
   Datapath



2. **The interface protocol between firmware, user project and testbench**

   a. **Firmware**

i. **fir.h**



```c
#ifndef __FIR_H__
#define __FIR_H__

#define fir_ap_ctrl 0x30000000 // ap_control
#define fir_len     0x30000010 // data length

#define fir_coeff   0x30000040 // Load into TapRAM

#define fir_x_in    0x30000080 // Load X into DataRAM
#define fir_y_out   0x30000084 // Take the output Y

#define checkbits   0x2600000C // MPRJ I/O

#define N 11
#define data_length N

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

#define adr_ofst(target, offset) (target + offset)
#define wb_write(target, data)   (*(volatile uint32_t*)(target)) = data // wishbone write
#define wb_read(target)          (*(volatile uint32_t*)(target))        // wishbone read



#endif
```

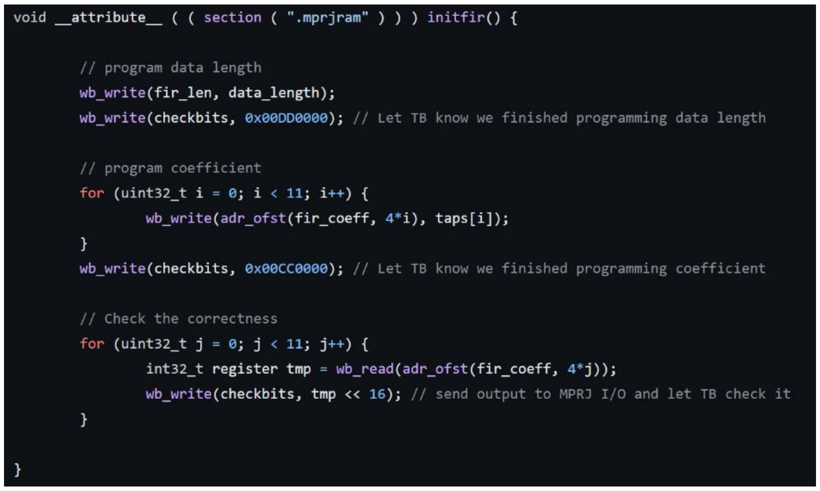
In the fir.h, we define some parameters, address, and functions for the firmware code to have more readability.

Functions:

1. adr_ofst: to calculate the offset address
2. wb_write: wishbone write the data to the target address
3. wb_read: read the data from the target address

ii. **fir.c**

In the firmware code, there is our implementation flow

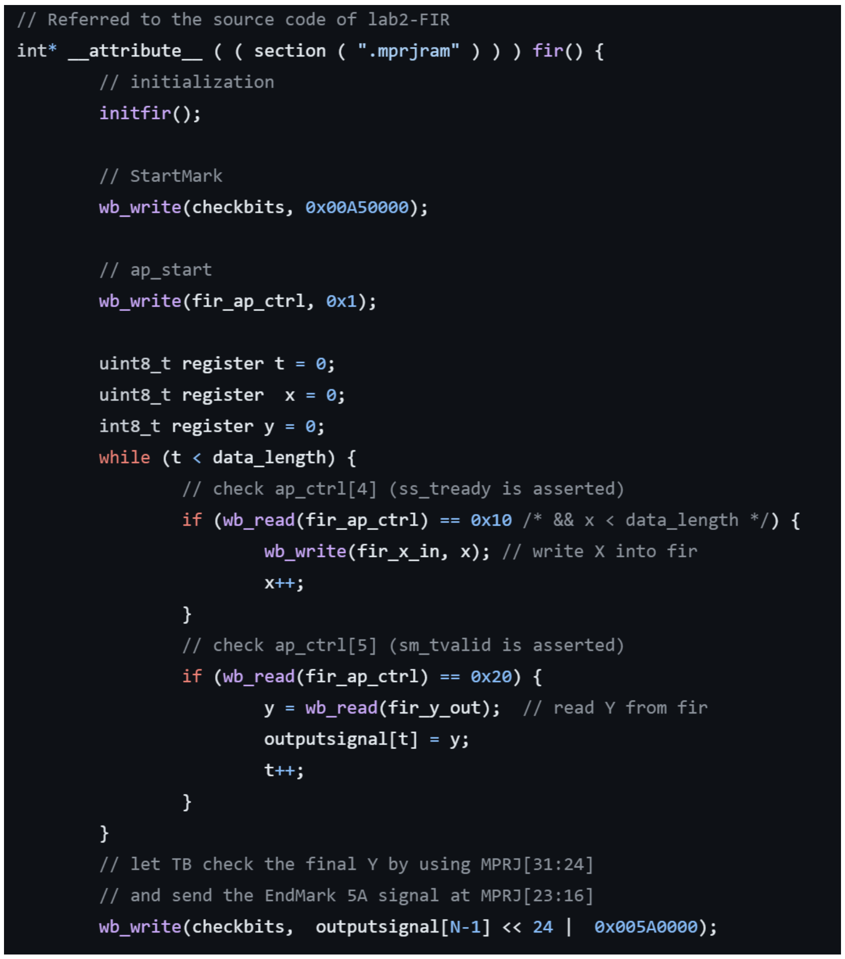


```c
void __attribute__ ( ( section ( ".mprjram" ) ) ) initfir() {

        // program data length
        wb_write(fir_len, data_length);
        wb_write(checkbits, 0x00DD0000); // Let TB know we finished programming data length

        // program coefficient
        for (uint32_t i = 0; i < 11; i++) {
                wb_write(adr_ofst(fir_coeff, 4*i), taps[i]);
        }
        wb_write(checkbits, 0x00CC0000); // Let TB know we finished programming coefficient

        // Check the correctness
        for (uint32_t j = 0; j < 11; j++) {
                int32_t register tmp = wb_read(adr_ofst(fir_coeff, 4*j));
                wb_write(checkbits, tmp << 16); // send output to MPRJ I/O and let TB check it
        }

}
```

1. Initialization:
    a. Program the data length (address and data are already defined in the .h file).
    b. Use for loop to write the coefficient. As for the address offset, we use the function defined in the .h file.

c. Write the 0x00CC0000 to checkbits address to let the testbench know that we the coefficients are all written.
d. Check the coefficients: read the coefficients then write in to the checkbits address to let the testbench check.
e. Fir implementation

2. FIR implementation:



```
// Referred to the source code of lab2-FIR
int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir() {
        // initialization
        initfir();

        // StartMark
        wb_write(checkbits, 0x00A50000);

        // ap_start
        wb_write(fir_ap_ctrl, 0x1);

        uint8_t register t = 0;
        uint8_t register  x = 0;
        int8_t register y = 0;
        while (t < data_length) {
                // check ap_ctrl[4] (ss_tready is asserted)
                if (wb_read(fir_ap_ctrl) == 0x10 /* && x < data_length */) {
                        wb_write(fir_x_in, x); // write X into fir
                        x++;
                }
                // check ap_ctrl[5] (sm_tvalid is asserted)
                if (wb_read(fir_ap_ctrl) == 0x20) {
                        y = wb_read(fir_y_out);  // read Y from fir
                        outputsignal[t] = y;
                        t++;
                }
        }
        // let TB check the final Y by using MPRJ[31:24]
        // and send the EndMark 5A signal at MPRJ[23:16]
        wb_write(checkbits,  outputsignal[N-1] << 24 |  0x005A0000);
```

a. Write 0x00A50000 to checkbits address as start mark.
b. Write ap_start.
c. Write data (we set data[x] = x, which is 0-64). The control signal is when ss_tready is asserted.
d. If sm_tvalid is asserted, which means y[t] is valid, we keep y to outputsignal[t] array.
e. Wishbone write the final outputsignal[N-1] and end mark at MPRJ[31:0] to testbench.
f. Do the calculation again.
g. Do it again.

b. Testbench (counter_la_fir_tb.v)

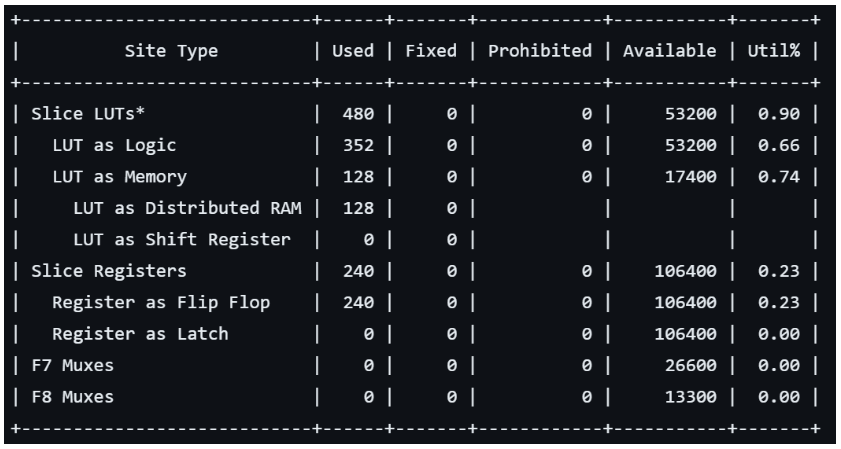
Testbench implementation flow:

i. Check if data length is written.
ii. Check if coefficients are written.

iii. Check if coefficients are right
iv. Check if the ap_start is asserted. If do, start to count the cycles till the final y is valid.
v. Display the total execution time, cycle, and Y.

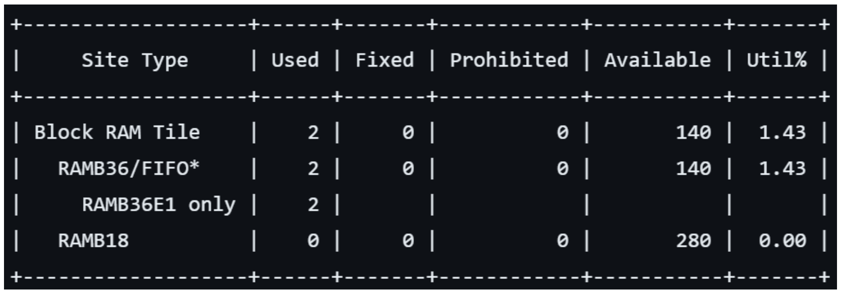
c. **User project**
Implement the wishbone interface.

3. **Waveform and analysis of the hardware/software behavior.**
   a. **hardware behavior:**
      i. **Synthesis**
         1. **Slice logic**



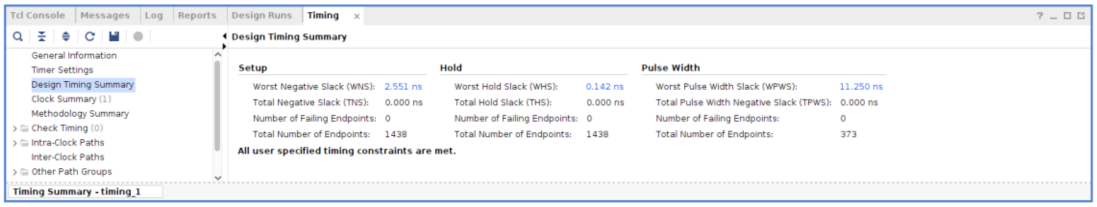
```
+--------------------------+------+-------+------------+-----------+-------+
|        Site Type         | Used | Fixed | Prohibited | Available | Util% |
+--------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*              |  480 |     0 |          0 |     53200 |  0.90 |
|   LUT as Logic           |  352 |     0 |          0 |     53200 |  0.66 |
|   LUT as Memory          |  128 |     0 |          0 |     17400 |  0.74 |
|     LUT as Distributed RAM |  128 |     0 |            |           |       |
|     LUT as Shift Register  |    0 |     0 |            |           |       |
| Slice Registers          |  240 |     0 |          0 |    106400 |  0.23 |
|   Register as Flip Flop  |  240 |     0 |          0 |    106400 |  0.23 |
|   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 |
+--------------------------+------+-------+------------+-----------+-------+
```

         2. **Memory(BRAM11, BRAM)**



```
+-------------------+------+-------+------------+-----------+-------+
|     Site Type     | Used | Fixed | Prohibited | Available | Util% |
+-------------------+------+-------+------------+-----------+-------+
| Block RAM Tile    |    2 |     0 |          0 |       140 |  1.43 |
|   RAMB36/FIFO*    |    2 |     0 |          0 |       140 |  1.43 |
|     RAMB36E1 only |    2 |       |            |           |       |
|   RAMB18          |    0 |     0 |          0 |       280 |  0.00 |
+-------------------+------+-------+------------+-----------+-------+
```

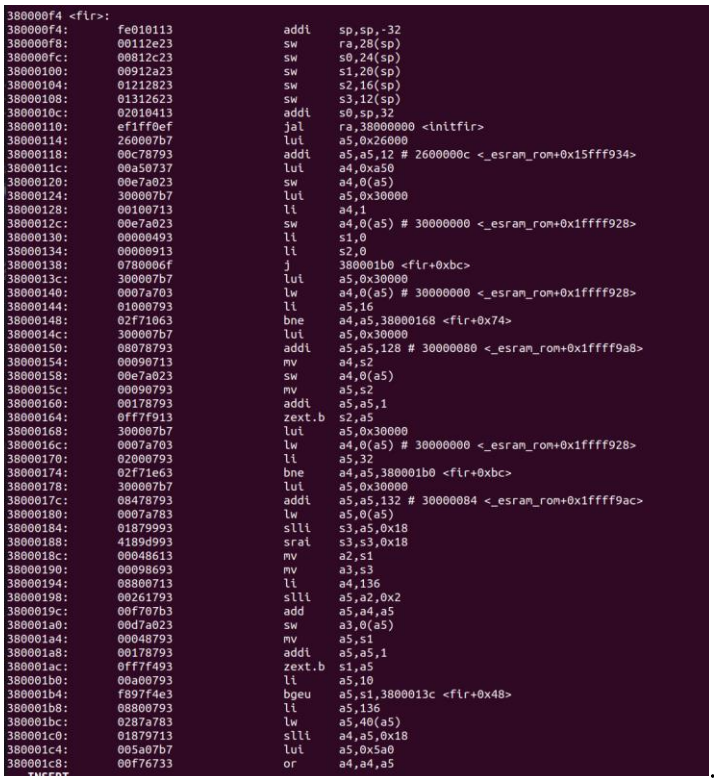
      ii. **timing summary**
      clock period: 25ns (wb_clk)



Design Timing Summary

```
Setup                                        Hold                                        Pulse Width
Worst Negative Slack (WNS):    2.551 ns      Worst Hold Slack (WHS):       0.142 ns      Worst Pulse Width Slack (WPWS):            11.250 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:     1438          Total Number of Endpoints:    1438          Total Number of Endpoints:                 373
All user specified timing constraints are met.
```

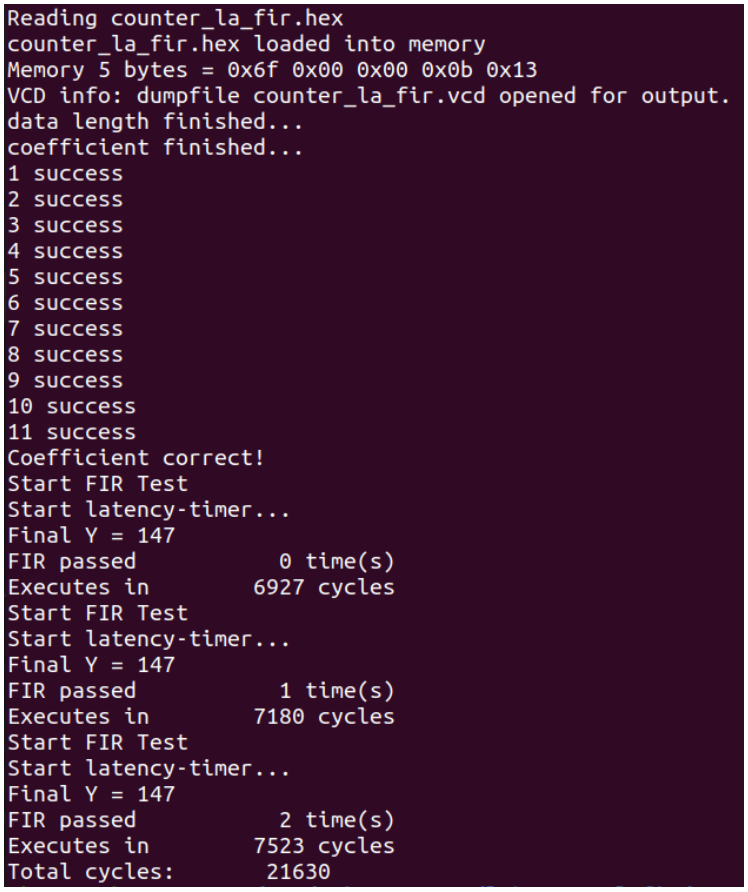
   b. **software behavior:**
   In counter_la_fir.out, we can see the risc-v assembly codes.



```
380000f4 <fir>:
380000f4:       fe010113                addi    sp,sp,-32
380000f8:       00112e23                sw      ra,28(sp)
380000fc:       00812c23                sw      s0,24(sp)
38000100:       00912a23                sw      s1,20(sp)
38000104:       01212823                sw      s2,16(sp)
38000108:       01312623                sw      s3,12(sp)
3800010c:       02010413                addi    s0,sp,32
38000110:       ef1ff0ef                jal     ra,38000000 <initfir>
38000114:       260007b7                lui     a5,0x26000
38000118:       00c78793                addi    a5,a5,12 # 2600000c <_esram_rom+0x15fff934>
3800011c:       00a50737                lui     a4,0xa50
38000120:       00e7a023                sw      a4,0(a5)
38000124:       300007b7                lui     a5,0x30000
38000128:       00100713                li      a4,1
3800012c:       00e7a023                sw      a4,0(a5) # 30000000 <_esram_rom+0x1ffff928>
38000130:       00000493                li      s1,0
38000134:       00000913                li      s2,0
38000138:       0780006f                j       380001b0 <fir+0xbc>
3800013c:       300007b7                lui     a5,0x30000
38000140:       0007a703                lw      a4,0(a5) # 30000000 <_esram_rom+0x1ffff928>
38000144:       01000793                li      a5,16
38000148:       02f71063                bne     a4,a5,38000168 <fir+0x74>
3800014c:       300007b7                lui     a5,0x30000
38000150:       08078793                addi    a5,a5,128 # 30000080 <_esram_rom+0x1ffff9a8>
38000154:       00090713                mv      a4,s2
38000158:       00e7a023                sw      a4,0(a5)
3800015c:       00090793                mv      a5,s2
38000160:       00178793                addi    a5,a5,1
38000164:       0ff7f913                zext.b  s2,a5
38000168:       300007b7                lui     a5,0x30000
3800016c:       0007a703                lw      a4,0(a5) # 30000000 <_esram_rom+0x1ffff928>
38000170:       02000793                li      a5,32
38000174:       02f71e63                bne     a4,a5,380001b0 <fir+0xbc>
38000178:       300007b7                lui     a5,0x30000
3800017c:       08478793                addi    a5,a5,132 # 30000084 <_esram_rom+0x1ffff9ac>
38000180:       0007a783                lw      a5,0(a5)
38000184:       01879993                slli    s3,a5,0x18
38000188:       4189d993                srai    s3,s3,0x18
3800018c:       00048613                mv      a2,s1
38000190:       00098693                mv      a3,s3
38000194:       08800713                li      a4,136
38000198:       00261793                slli    a5,a2,0x2
3800019c:       00f707b3                add     a5,a4,a5
380001a0:       00d7a023                sw      a3,0(a5)
380001a4:       00048793                mv      a5,s1
380001a8:       00178793                addi    a5,a5,1
380001ac:       0ff7f493                zext.b  s1,a5
380001b0:       00a00793                li      a5,10
380001b4:       f897f4e3                bgeu    a5,s1,3800013c <fir+0x48>
380001b8:       08800793                li      a5,136
380001bc:       0287a783                lw      a5,40(a5)
380001c0:       01879713                slli    a4,a5,0x18
380001c4:       005a07b7                lui     a5,0x5a0
380001c8:       00f76733                or      a4,a4,a5
```
……

Different from the fir code in lab4-1, this time the firmware C code isn’t executing the FIR calculation. Instead, it is implementing data and control signal processing.

4. **What is the FIR engine theoretical throughput, i.e. data rate? Actual measured throughput?**

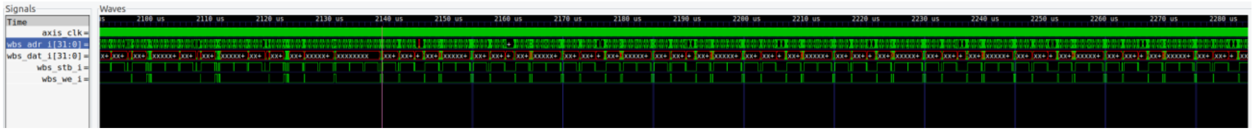


```
Reading counter_la_fir.hex
counter_la_fir.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_fir.vcd opened for output.
data length finished...
coefficient finished...
1 success
2 success
3 success
4 success
5 success
6 success
7 success
8 success
9 success
10 success
11 success
Coefficient correct!
Start FIR Test
Start latency-timer...
Final Y = 147
FIR passed           0 time(s)
Executes in        6927 cycles
Start FIR Test
Start latency-timer...
Final Y = 147
FIR passed           1 time(s)
Executes in        7180 cycles
Start FIR Test
Start latency-timer...
Final Y = 147
FIR passed           2 time(s)
Executes in        7523 cycles
Total cycles:        21630
```

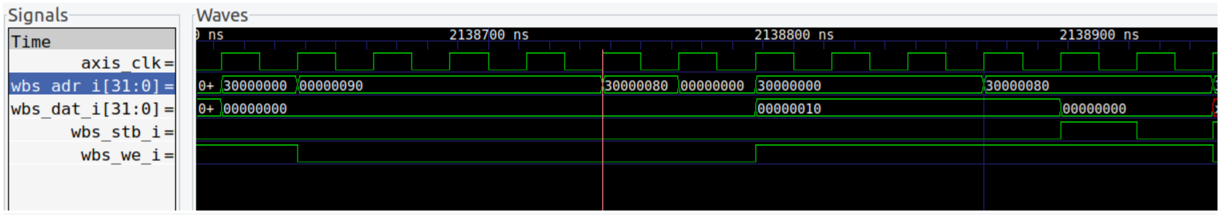
Theoretical throughput = 1

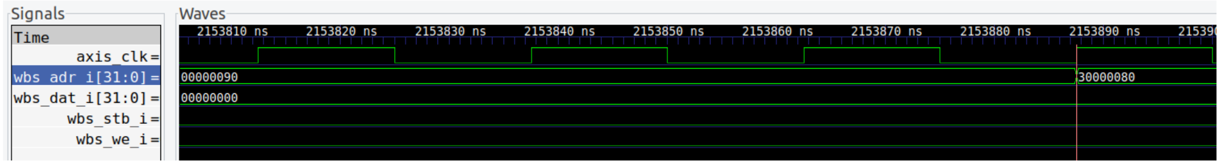
Actual measured throughput = 1 if we see the data calculation part. However, we have many cycles delay due to the firmware code instructions and memory access.

5. **What is latency for firmware to feed data?**



blue lines mark the wbs_adr_i = 0x30000080 (x[n])





Therefore, we can calculate the latency of firmware to feed the data ≈ 2153880 - 2138700

15180 cycles.

6. **What techniques used to improve the throughput?**
    a. We used the pipeline technique to improve the throughput. (just like what we did in the lab3)
    b. Use unsigned integer instead of signed integer (double performance!!)
    c. Don't use x++ in the function. (just reduce a small number of cycle)

7. **Does bram12 give better performance, in what way?**
   We used the old bram11 provided in lab3, which can only be read or written in one time. However, if we use bram12, which can be read and written in the same time, the performance is definitely going to improve since the every time we have to wait the read operation then we can write and vice versa. Therefore, it will definitely have better performance.

8. **Can you suggest other methods to improve the performance?**
    a. Use unsigned integer instead of signed integer (double performance!!)
    b. Don't use x++ in the function. (just reduce a small number of cycle)