NTHU 112 Fall Semester

系統晶片設計

SOC Design Laboratory

Lab4-1

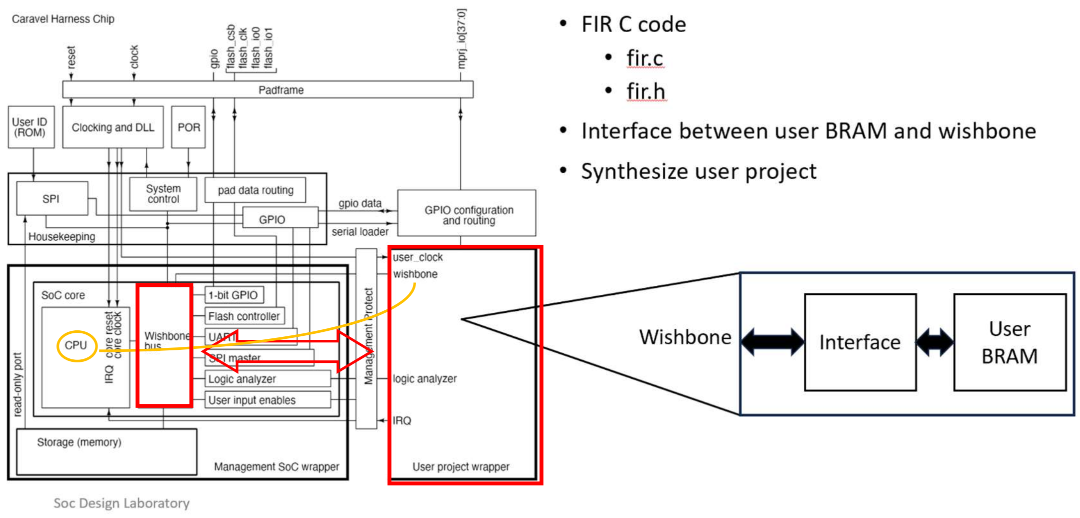
組別:第 18 組

學號: 111061647, 111064537, 112061576

姓名: 盧人豪，林亭君，莊家政

Introduction

In Lab4-0, we set up the Caravel SoC environment and ran simulations. During our experiments, we observed the differences between using a logic analyzer interface and a Wishbone interface for implementing a counter and a GCD engine. These interfaces facilitate communication between the logic analyzer interface/Wishbone interface and the user project.

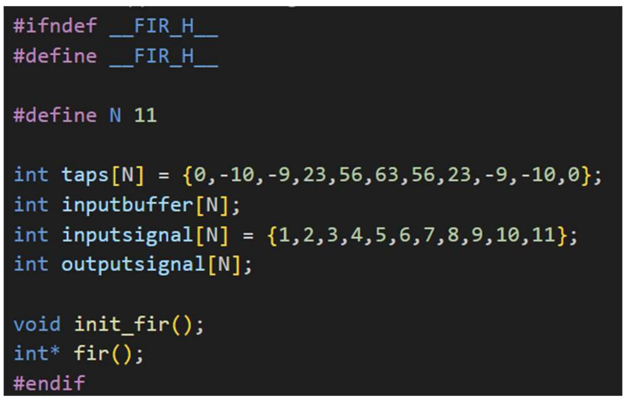


- FIR C code
  - fir.c
  - fir.h
- Interface between user BRAM and wishbone
- Synthesize user project

For the next part of this lab, we wrote firmware code (fir.c and fir.h) to implement the FIR engine. Additionally, we needed to establish interfaces for both Wishbone and the user BRAM. The entire workflow can be summarized as follows: The RISCV CPU uses the firmware code to perform FIR filtering, and the results are transmitted to the user project through the Wishbone interface, where the interface helps store the results in the user BRAM.

Explanation of your firmware code

fir.h: This makes the same order of taps[N] from the original file so as inputsignal[N] .



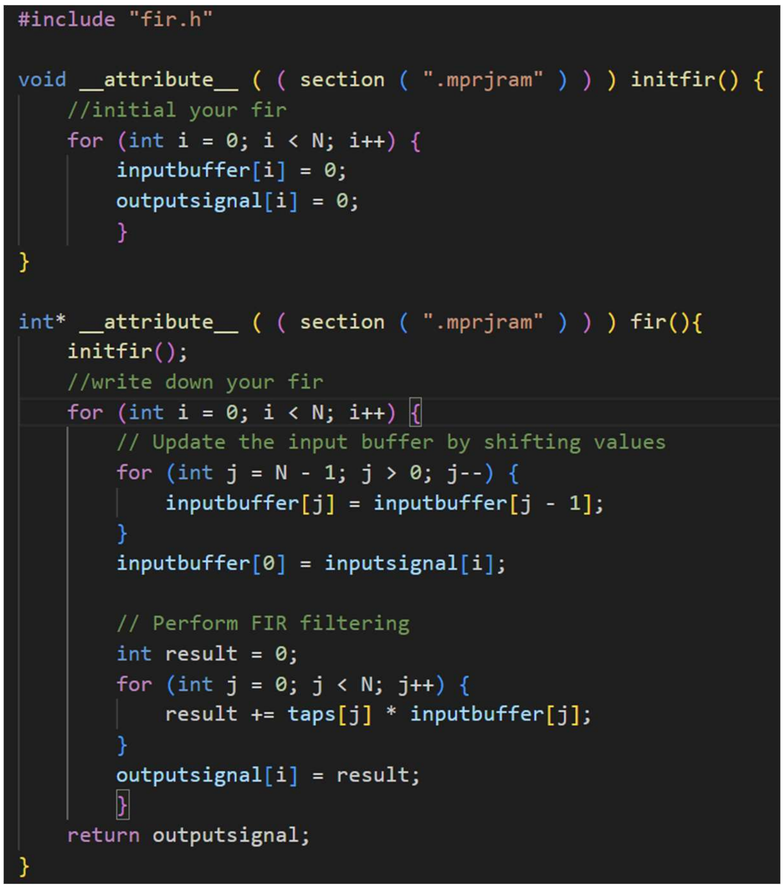
```c
#ifndef __FIR_H__
#define __FIR_H__

#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];

void init_fir();
int* fir();
#endif
```

fir.c:

‘’initfir function’’: This function is marked with` __attribute__((section(".mprjram"))’, indicating that it will be placed in a specific memory section (.mprjram). It is used to initialize the FIR filter by initializing two arrays, inputbuffer and outputsignal, which may be used as buffers to store input and output signals.

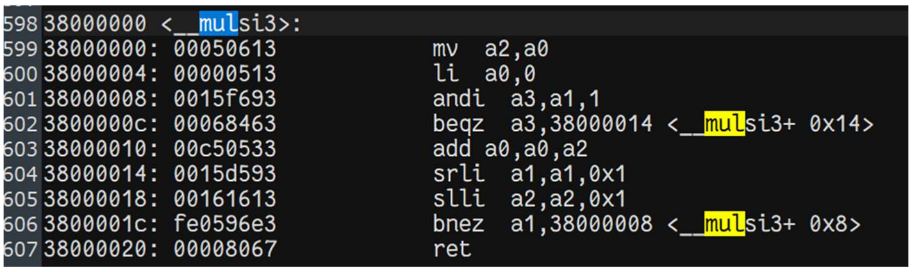
‘’fir function’’: This function is also marked with __attribute__((section(".mprjram")), so it will also reside in the same memory section. It implements the main computation of the FIR filter.

- First, it calls the initfir function to ensure the initial state of the FIR filter is set.
- Then, it iterates through the samples of the input signal using as for loop (in this code, N is a constant representing the number of samples).
- In each sample processing step, it stores the sample of the input signal in the inputbuffer array and also implements the operation of shifting data forward from the end of the inputbuffer, as known as inputbuffer[11]. Then inputbuffer[0] will be replaced by a new data and this is done to simulate a rolling buffer.
- Next, it performs the FIR filtering operation by applying a set of filter coefficients known as taps to the data in the inputbuffer, and stores the result in the outputsignal array.
- Finally, the fir function returns a pointer to the outputsignal array, which contains the output signal after being processed by the FIR filter.

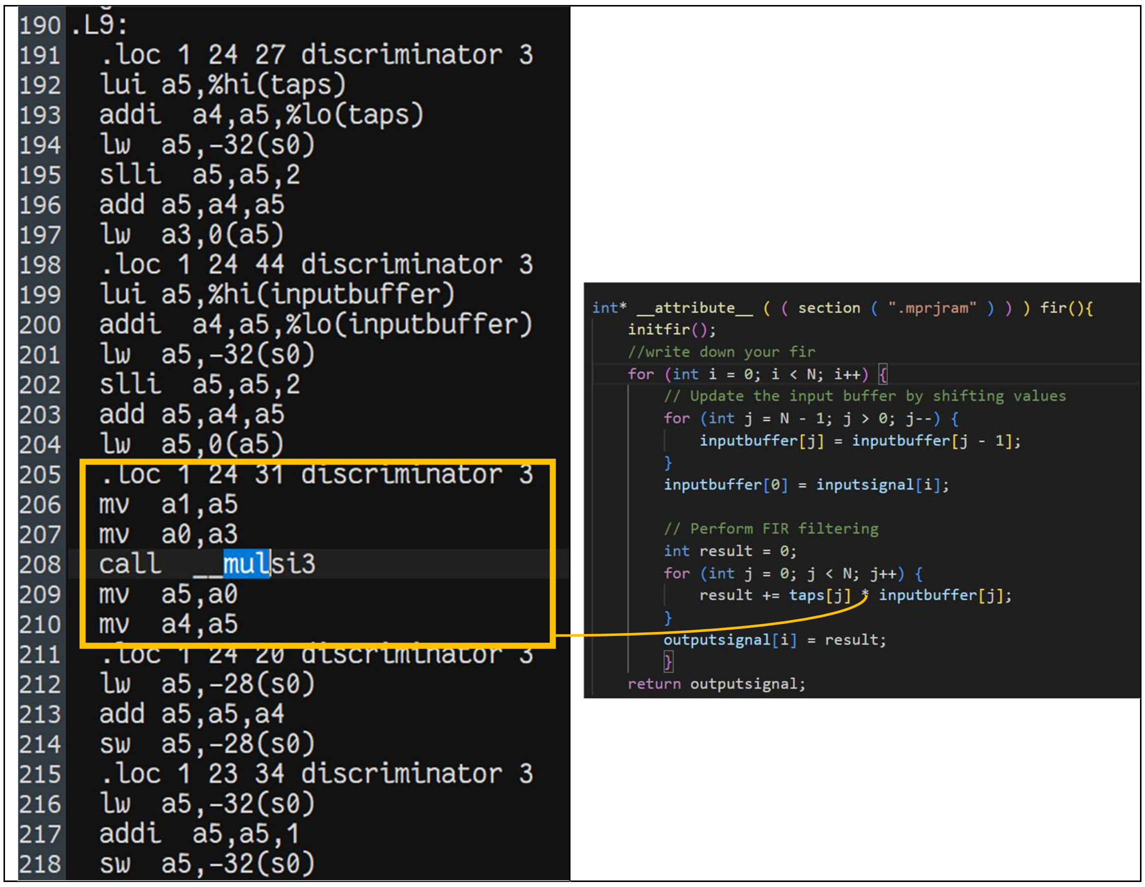

```c
#include "fir.h"

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++) {
        // Update the input buffer by shifting values
        for (int j = N - 1; j > 0; j--) {
            inputbuffer[j] = inputbuffer[j - 1];
        }
        inputbuffer[0] = inputsignal[i];

        // Perform FIR filtering
        int result = 0;
        for (int j = 0; j < N; j++) {
            result += taps[j] * inputbuffer[j];
        }
        outputsignal[i] = result;
        }
    return outputsignal;
}
```

How does it execute a multiplication in assembly code

In counter_la_fir.out ,we can observe the function __mulsi3


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

In counter_la_fir.elf-fir.s ,we can see that it’s doing multiplication.



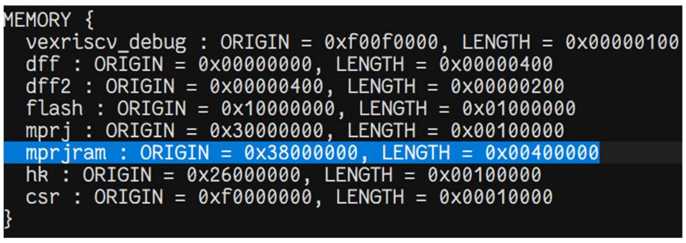
```
.L9:
  .loc 1 24 27 discriminator 3
  lui a5,%hi(taps)
  addi  a4,a5,%lo(taps)
  lw  a5,-32(s0)
  slli  a5,a5,2
  add a5,a4,a5
  lw  a3,0(a5)
  .loc 1 24 44 discriminator 3
  lui a5,%hi(inputbuffer)
  addi  a4,a5,%lo(inputbuffer)
  lw  a5,-32(s0)
  slli  a5,a5,2
  add a5,a4,a5
  lw  a5,0(a5)
  .loc 1 24 31 discriminator 3
  mv  a1,a5
  mv  a0,a3
  call  __mulsi3
  mv  a5,a0
  mv  a4,a5
  .loc 1 24 20 discriminator 3
  lw  a5,-28(s0)
  add a5,a5,a4
  sw  a5,-28(s0)
  .loc 1 23 34 discriminator 3
  lw  a5,-32(s0)
  addi  a5,a5,1
  sw  a5,-32(s0)
```

```c
int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir(){
    initfir();
    //write down your fir
    for (int i = 0; i < N; i++) {
        // Update the input buffer by shifting values
        for (int j = N - 1; j > 0; j--) {
            inputbuffer[j] = inputbuffer[j - 1];
        }
        inputbuffer[0] = inputsignal[i];

        // Perform FIR filtering
        int result = 0;
        for (int j = 0; j < N; j++) {
            result += taps[j] * inputbuffer[j];
        }
        outputsignal[i] = result;
    }
    return outputsignal;
```

Combining two photos above, it can be seen that ‘’__mulsi3” is performing multiplication of two integers a0 and a1, with the result stored in a0. The specific calculation process involves bitwise processing of the digits in a1 using left shift (slli) and right shift (srli) operations to achieve integer multiplication.

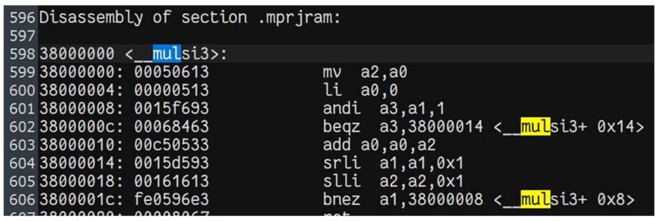
What address allocate for user project and how many space is required to allocate to firmware code

In section.lds , we can find address allocated for the user project is 0x38000000.

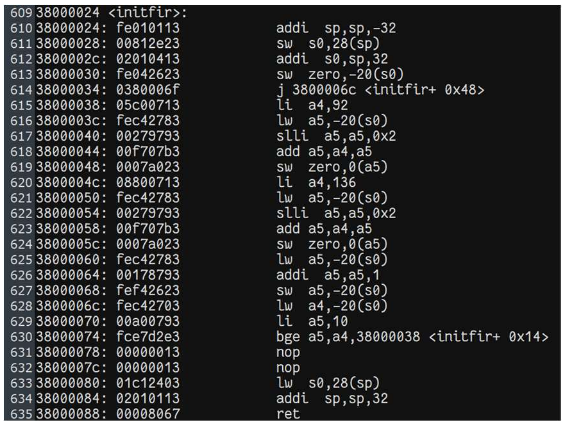


```
MEMORY {
  vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
  dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
  dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
  flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
  mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
  mprjram : ORIGIN = 0x38000000, LENGTH = 0x00400000
  hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
  csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
}
```

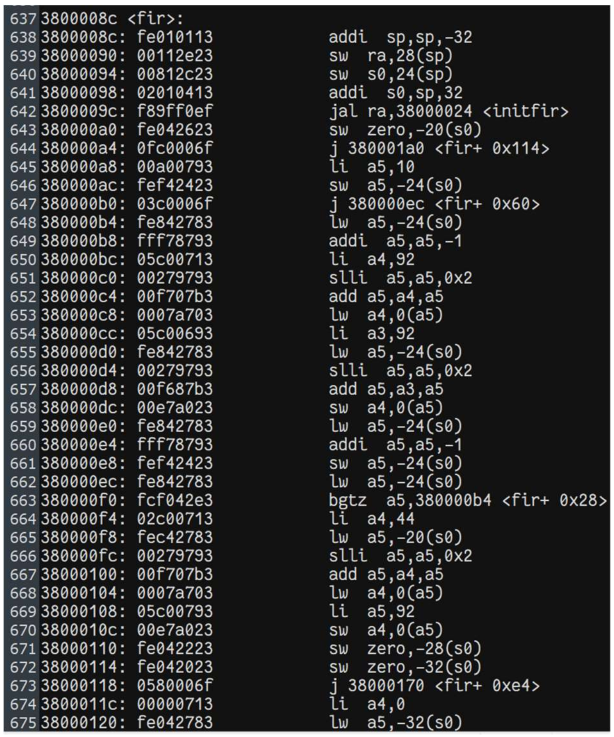
By examining the counter_la_fir.out file, we can determine that the entire mprjram requires 448 bytes, which is represented in hexadecimal as 0x1c0.



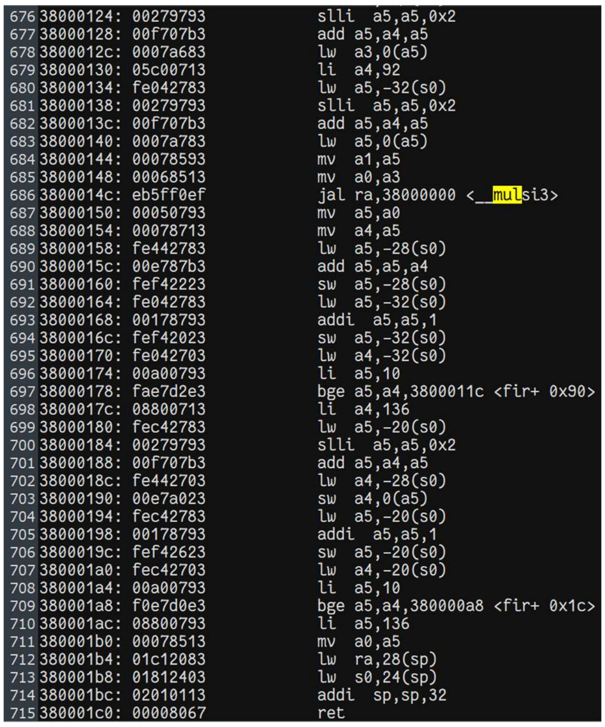
```
Disassembly of section .mprjram:

38000000 <__mulsi3>:
38000000: 00050613              mv   a2,a0
38000004: 00000513              li   a0,0
38000008: 0015f693              andi  a3,a1,1
3800000c: 00068463              beqz  a3,38000014 <__mulsi3+ 0x14>
38000010: 00c50533              add a0,a0,a2
38000014: 0015d593              srli  a1,a1,0x1
38000018: 00161613              slli  a2,a2,0x1
3800001c: fe0596e3              bnez  a1,38000008 <__mulsi3+ 0x8>
```



```
38000024 <initfir>:
38000024: fe010113              addi  sp,sp,-32
38000028: 00812e23              sw  s0,28(sp)
3800002c: 02010413              addi  s0,sp,32
38000030: fe042623              sw  zero,-20(s0)
38000034: 0380006f              j 3800006c <initfir+ 0x48>
38000038: 05c00713              li  a4,92
3800003c: fec42783              lw  a5,-20(s0)
38000040: 00279793              slli  a5,a5,0x2
38000044: 00f707b3              add a5,a4,a5
38000048: 0007a023              sw  zero,0(a5)
3800004c: 08800713              li  a4,136
38000050: fec42783              lw  a5,-20(s0)
38000054: 00279793              slli  a5,a5,0x2
38000058: 00f707b3              add a5,a4,a5
3800005c: 0007a023              sw  zero,0(a5)
38000060: fec42783              lw  a5,-20(s0)
38000064: 00178793              addi  a5,a5,1
38000068: fef42623              sw  a5,-20(s0)
3800006c: fec42703              lw  a4,-20(s0)
38000070: 00a00793              li  a5,10
38000074: fce7d2e3              bge a5,a4,38000038 <initfir+ 0x14>
38000078: 00000013              nop
3800007c: 00000013              nop
38000080: 01c12403              lw  s0,28(sp)
38000084: 02010113              addi  sp,sp,32
38000088: 00008067              ret
```



```
3800008c <fir>:
3800008c: fe010113              addi  sp,sp,-32
38000090: 00112e23              sw  ra,28(sp)
38000094: 00812c23              sw  s0,24(sp)
38000098: 02010413              addi  s0,sp,32
3800009c: f89ff0ef              jal ra,38000024 <initfir>
380000a0: fe042623              sw  zero,-20(s0)
380000a4: 0fc0006f              j 380001a0 <fir+ 0x114>
380000a8: 00a00793              li  a5,10
380000ac: fef42423              sw  a5,-24(s0)
380000b0: 03c0006f              j 380000ec <fir+ 0x60>
380000b4: fe842783              lw  a5,-24(s0)
380000b8: fff78793              addi  a5,a5,-1
380000bc: 05c00713              li  a4,92
380000c0: 00279793              slli  a5,a5,0x2
380000c4: 00f707b3              add a5,a4,a5
380000c8: 0007a703              lw  a4,0(a5)
380000cc: 05c00693              li  a3,92
380000d0: fe842783              lw  a5,-24(s0)
380000d4: 00279793              slli  a5,a5,0x2
380000d8: 00f687b3              add a5,a3,a5
380000dc: 00e7a023              sw  a4,0(a5)
380000e0: fe842783              lw  a5,-24(s0)
380000e4: fff78793              addi  a5,a5,-1
380000e8: fef42423              sw  a5,-24(s0)
380000ec: fe842783              lw  a5,-24(s0)
380000f0: fcf042e3              bgtz  a5,380000b4 <fir+ 0x28>
380000f4: 02c00713              li  a4,44
380000f8: fec42783              lw  a5,-20(s0)
380000fc: 00279793              slli  a5,a5,0x2
38000100: 00f707b3              add a5,a4,a5
38000104: 0007a703              lw  a4,0(a5)
38000108: 05c00793              li  a5,92
3800010c: 00e7a023              sw  a4,0(a5)
38000110: fe042223              sw  zero,-28(s0)
38000114: fe042023              sw  zero,-32(s0)
38000118: 0580006f              j 38000170 <fir+ 0xe4>
3800011c: 00000713              li  a4,0
38000120: fe042783              lw  a5,-32(s0)
```

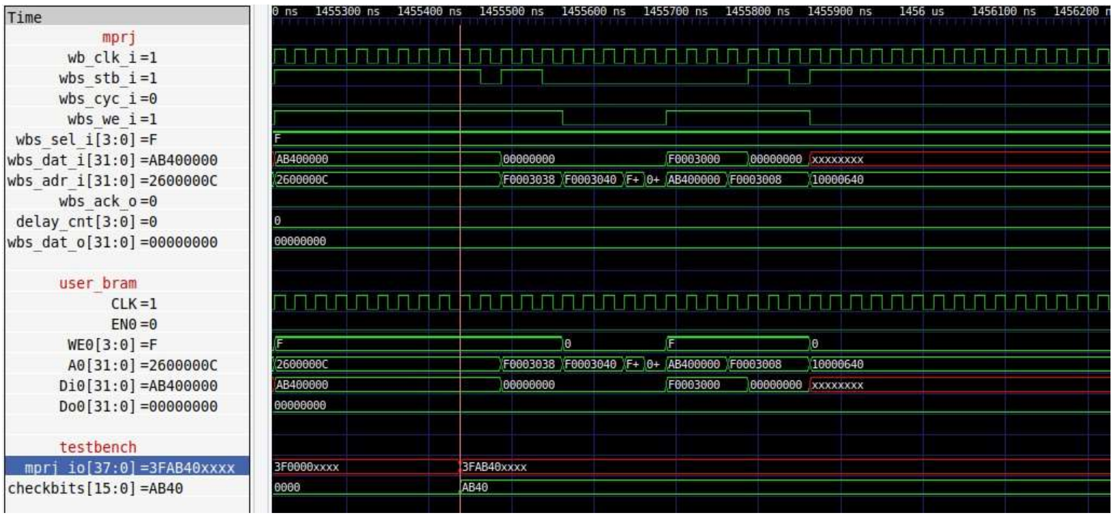


```
38000124: 00279793          slli  a5,a5,0x2
38000128: 00f707b3          add a5,a4,a5
3800012c: 0007a683          lw  a3,0(a5)
38000130: 05c00713          li  a4,92
38000134: fe042783          lw  a5,-32(s0)
38000138: 00279793          slli  a5,a5,0x2
3800013c: 00f707b3          add a5,a4,a5
38000140: 0007a783          lw  a5,0(a5)
38000144: 00078593          mv  a1,a5
38000148: 00068513          mv  a0,a3
3800014c: eb5ff0ef          jal ra,38000000 <__mulsi3>
38000150: 00050793          mv  a5,a0
38000154: 00078713          mv  a4,a5
38000158: fe442783          lw  a5,-28(s0)
3800015c: 00e787b3          add a5,a5,a4
38000160: fef42223          sw  a5,-28(s0)
38000164: fe042783          lw  a5,-32(s0)
38000168: 00178793          addi  a5,a5,1
3800016c: fef42023          sw  a5,-32(s0)
38000170: fe042703          lw  a4,-32(s0)
38000174: 00a00793          li  a5,10
38000178: fae7d2e3          bge a5,a4,3800011c <fir+ 0x90>
3800017c: 08800713          li  a4,136
38000180: fec42783          lw  a5,-20(s0)
38000184: 00279793          slli  a5,a5,0x2
38000188: 00f707b3          add a5,a4,a5
3800018c: fe442703          lw  a4,-28(s0)
38000190: 00e7a023          sw  a4,0(a5)
38000194: fec42783          lw  a5,-20(s0)
38000198: 00178793          addi  a5,a5,1
3800019c: fef42623          sw  a5,-20(s0)
380001a0: fec42703          lw  a4,-20(s0)
380001a4: 00a00793          li  a5,10
380001a8: f0e7d0e3          bge a5,a4,380000a8 <fir+ 0x1c>
380001ac: 08800793          li  a5,136
380001b0: 00078513          mv  a0,a5
380001b4: 01c12083          lw  ra,28(sp)
380001b8: 01812403          lw  s0,24(sp)
380001bc: 02010113          addi  sp,sp,32
380001c0: 00008067          ret
```

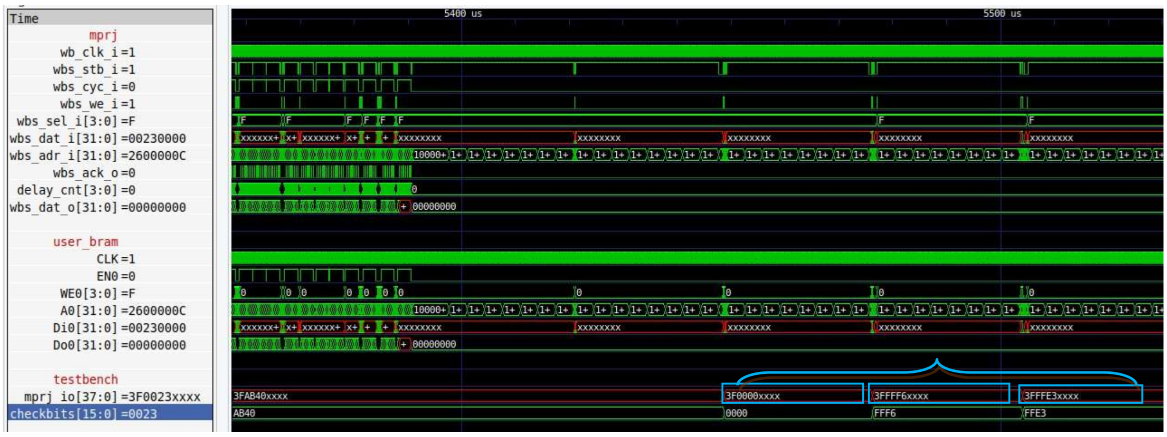
Interface between BRAM and wishbone

Waveform from xsim:

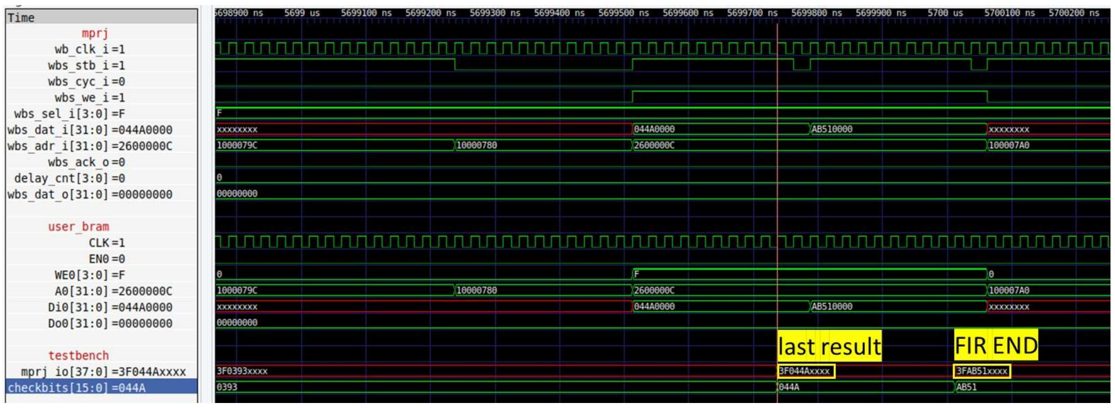
When mprj_io[37:0] =16'hAB40, the CPU is starting to execute the FIR.



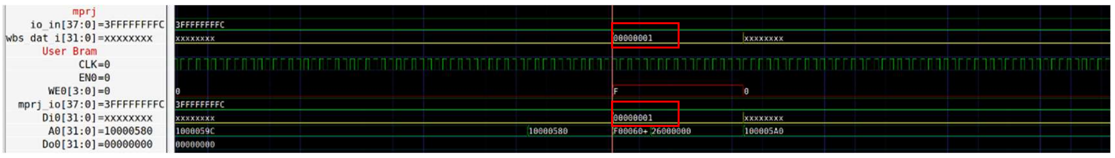
Wishbone send the results of the data calculations to the interface

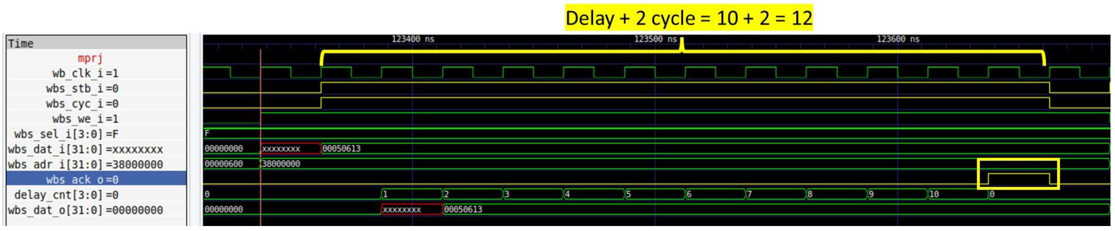


Check_bits = ‘hAB51 end of FIR engine

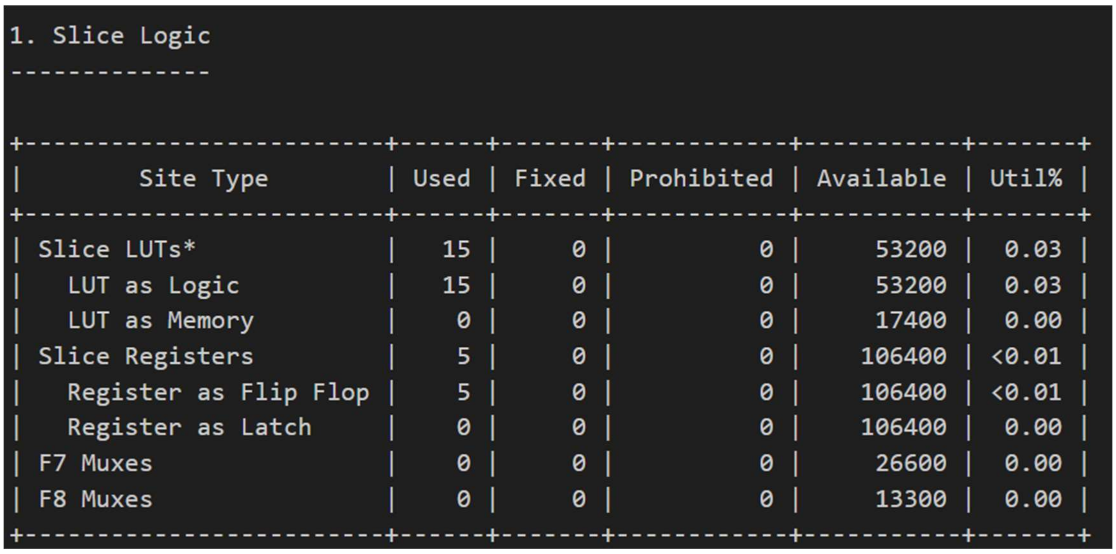


Then write the result to BRAM.

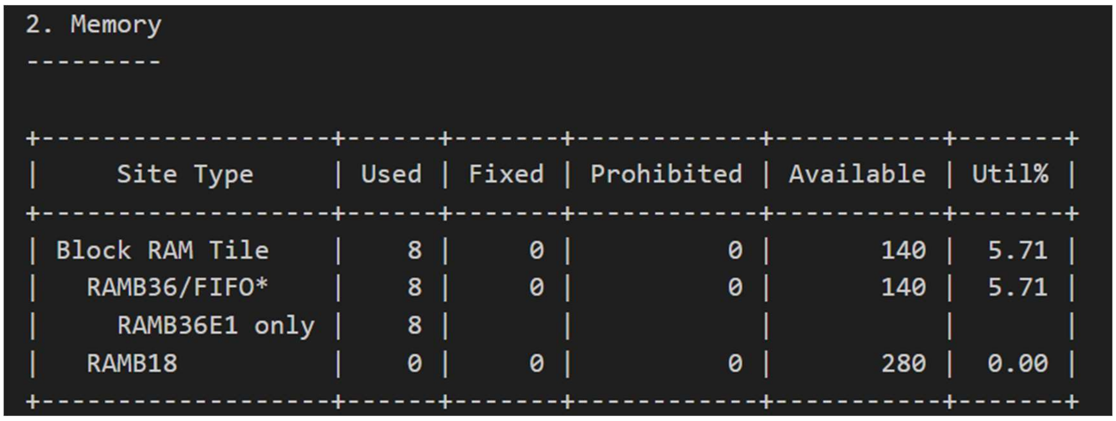




Synthesis report



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

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |   15 |     0 |          0 |     53200 |  0.03 |
|   LUT as Logic          |   15 |     0 |          0 |     53200 |  0.03 |
|   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 |
+-------------------------+------+-------+------------+-----------+-------+
```



```
2. Memory
---------

+-------------------+------+-------+------------+-----------+-------+
|     Site Type     | Used | Fixed | Prohibited | Available | Util% |
+-------------------+------+-------+------------+-----------+-------+
| Block RAM Tile    |    8 |     0 |          0 |       140 |  5.71 |
|   RAMB36/FIFO*    |    8 |     0 |          0 |       140 |  5.71 |
|     RAMB36E1 only |    8 |       |            |           |       |
|   RAMB18          |    0 |     0 |          0 |       280 |  0.00 |
+-------------------+------+-------+------------+-----------+-------+
```

Github link https://github.com/lkl110137918218/SoC-design_Lab4-1