# 112-1 SoC Design Laboratory

# Lab4-1

Group 10 王語卉 吳至凌 高宇勝

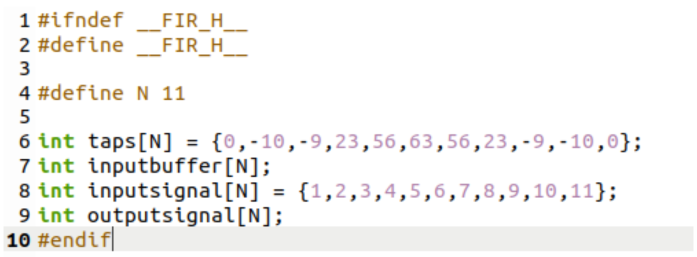
## Introduction

In lab4-0, we build up the Caravel SoC environment and initiated simulations. We observed the difference between using logic analyzer interface and wishbone interface for the implementation of both a counter and a GCD engine. These interfaces facilitate communication between the logic analyzer interface/wishbone interface and the user project.

In this lab, we write our firmware code to implement the FIR engine. Additionally, we build up the interface of wishbone and user bram. The entire workflow can be outlined as follows: (1) The RISCV CPU use the firmware code to do FIR filtering, (2) the result is transmitted to the user project via the wishbone, (3) the interface helps to store the result to the user bram.

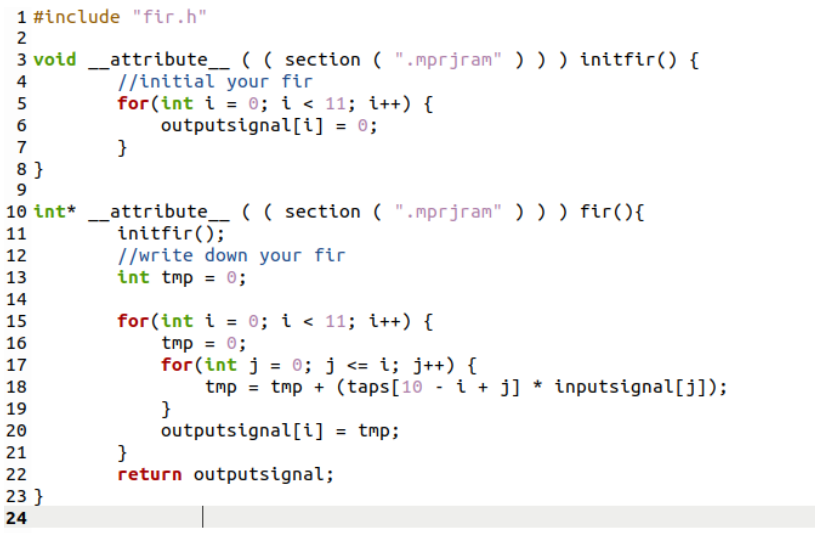
## 1. Explanation of your firmware code

In fir.h, we design the order of taps[N] and inputsignal[N] are the same as the original file.



```
#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
```

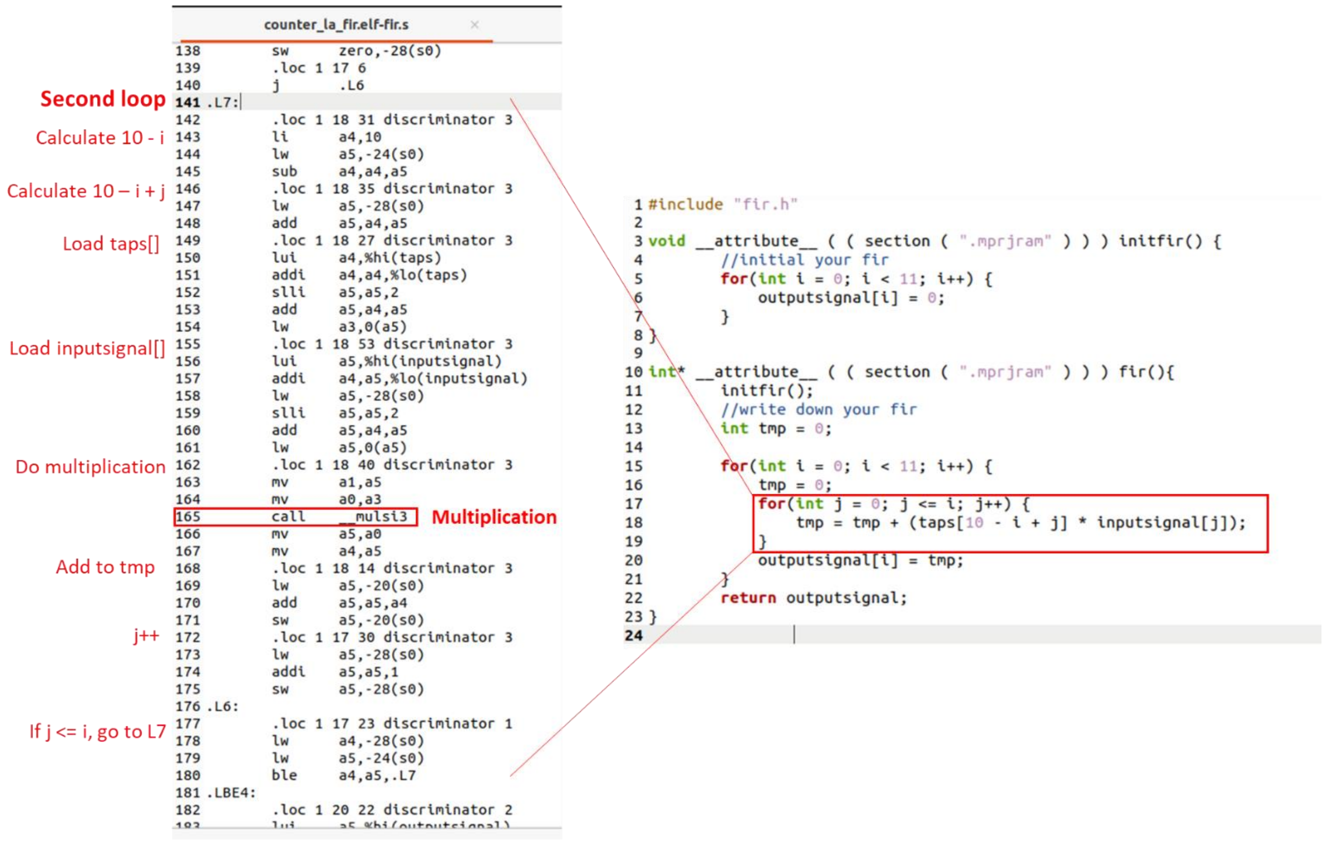
In fir.c, we use a for loop to initialize outputsignal[N] to zero. Then, we use double loop to calculate FIR with a temporary variable for accumulation. The result of each iteration is stored to outputsignal[i], which will be return to counter_la_fir.c who called the fir() function.



```c
#include "fir.h"

void __attribute__ ( ( section ( ".mprjram" ) ) ) initfir() {
	//initial your fir
	for(int i = 0; i < 11; i++) {
		outputsignal[i] = 0;
	}
}

int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir(){
	initfir();
	//write down your fir
	int tmp = 0;

	for(int i = 0; i < 11; i++) {
		tmp = 0;
		for(int j = 0; j <= i; j++) {
			tmp = tmp + (taps[10 - i + j] * inputsignal[j]);
		}
		outputsignal[i] = tmp;
	}
	return outputsignal;
}
```

## A. How does it execute a multiplication in assembly code

In assembly code file(counter_la_fir.elf-fir.s), the multiplication is done in the second loop, i.e., L7 part.

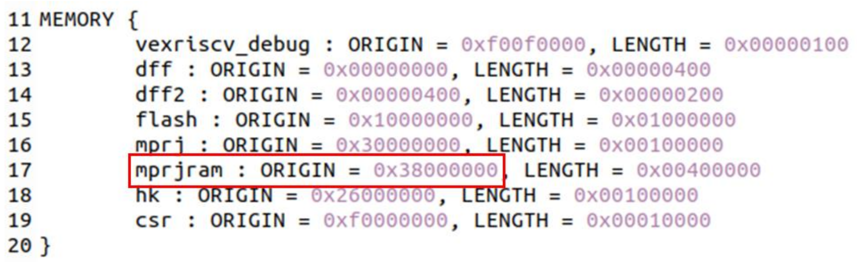


```
		sw	zero,-28(s0)
		.loc 1 17 6
		j	.L6
.L7:
		.loc 1 18 31 discriminator 3
		li	a4,10
		lw	a5,-24(s0)
		sub	a4,a4,a5
		.loc 1 18 35 discriminator 3
		lw	a5,-28(s0)
		add	a5,a4,a5
		.loc 1 18 27 discriminator 3
		lui	a4,%hi(taps)
		addi	a4,a4,%lo(taps)
		slli	a5,a5,2
		add	a5,a4,a5
		lw	a3,0(a5)
		.loc 1 18 53 discriminator 3
		lui	a5,%hi(inputsignal)
		addi	a4,a5,%lo(inputsignal)
		lw	a5,-28(s0)
		slli	a5,a5,2
		add	a5,a4,a5
		lw	a5,0(a5)
		.loc 1 18 40 discriminator 3
		mv	a1,a5
		mv	a0,a3
		call	__mulsi3
		mv	a5,a0
		mv	a4,a5
		.loc 1 18 14 discriminator 3
		lw	a5,-20(s0)
		add	a5,a5,a4
		sw	a5,-20(s0)
		.loc 1 17 30 discriminator 3
		lw	a5,-28(s0)
		addi	a5,a5,1
		sw	a5,-28(s0)
.L6:
		.loc 1 17 23 discriminator 1
		lw	a4,-28(s0)
		lw	a5,-24(s0)
		ble	a4,a5,.L7
.LBE4:
		.loc 1 20 22 discriminator 2
```

```c
#include "fir.h"

void __attribute__ ( ( section ( ".mprjram" ) ) ) initfir() {
	//initial your fir
	for(int i = 0; i < 11; i++) {
		outputsignal[i] = 0;
	}
}

int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir(){
	initfir();
	//write down your fir
	int tmp = 0;

	for(int i = 0; i < 11; i++) {
		tmp = 0;
		for(int j = 0; j <= i; j++) {
			tmp = tmp + (taps[10 - i + j] * inputsignal[j]);
		}
		outputsignal[i] = tmp;
	}
	return outputsignal;
}
```

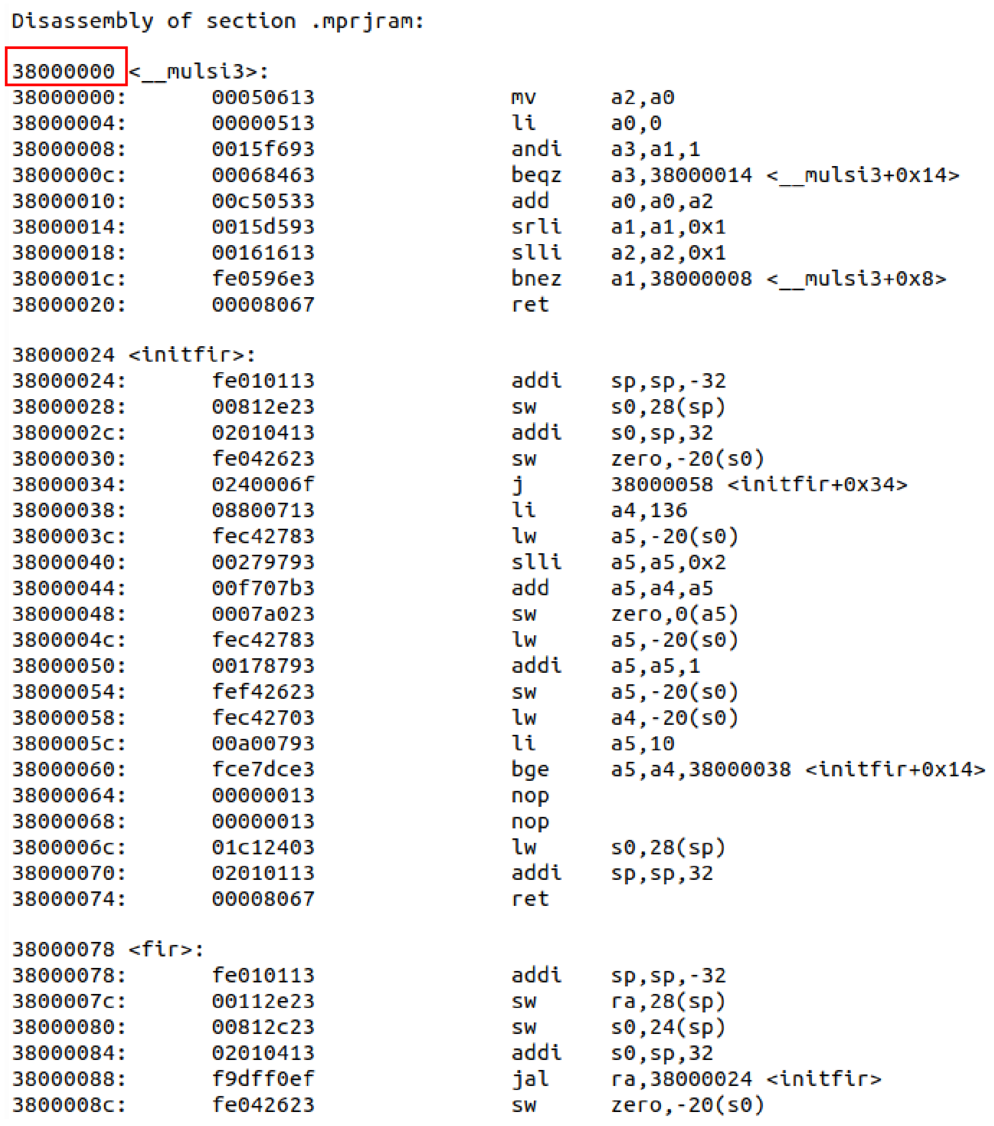
The multiplication is done by calling the "__mulsi3" function. This function performs the multiplication of the value stored in a0, a1.

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

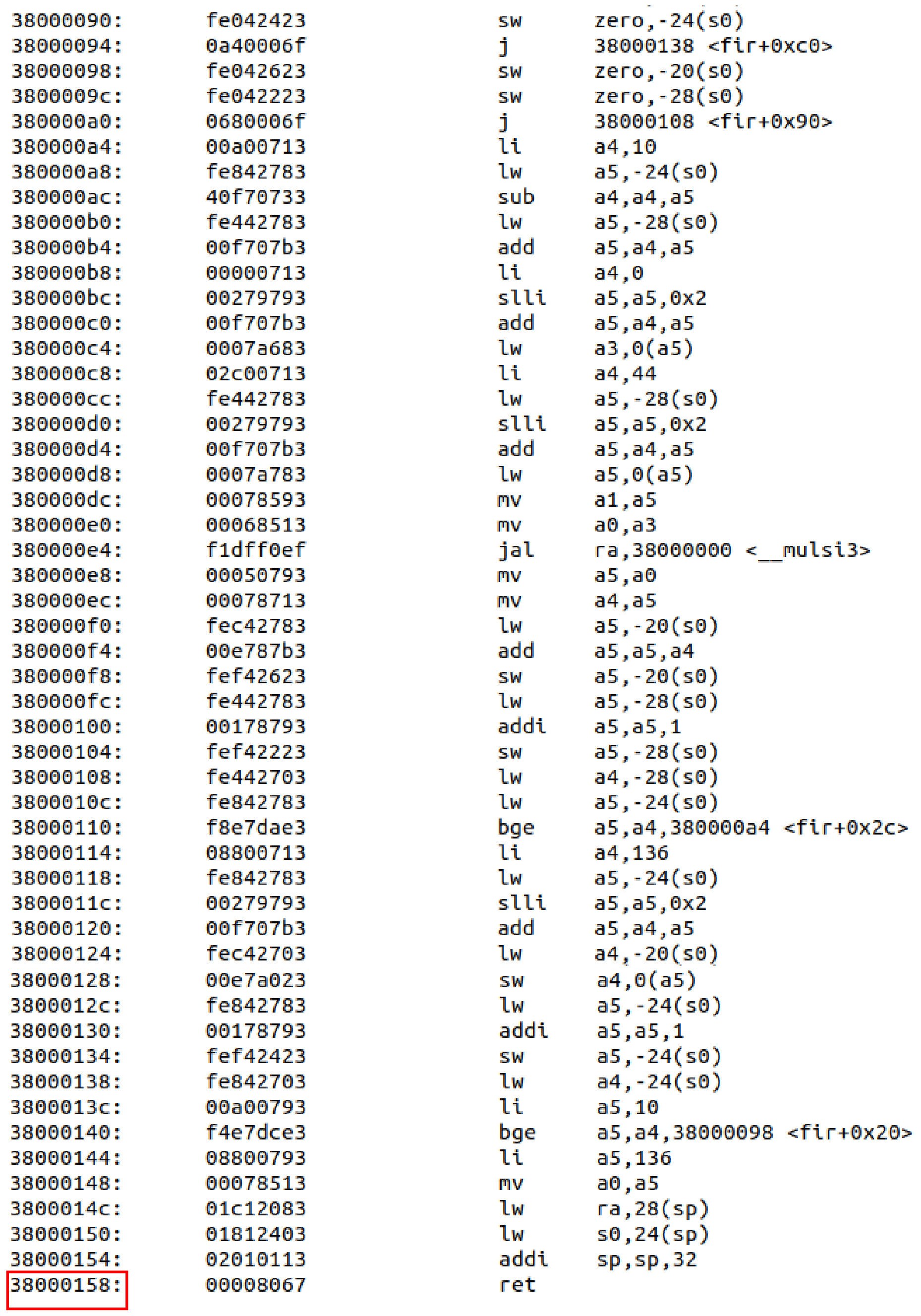


```
MEMORY {
        vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
        dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
        dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
        flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
        mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000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

By the memory arrangement in section.lds, the address allocate for user project is 0x38000000. By counter_la_fir.out, we can find that the whole mprjram requires 344 bytes (hex 158).



```
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>
38000020:       00008067                ret

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:       0240006f                j       38000058 <initfir+0x34>
38000038:       08800713                li      a4,136
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:       fec42783                lw      a5,-20(s0)
38000050:       00178793                addi    a5,a5,1
38000054:       fef42623                sw      a5,-20(s0)
38000058:       fec42703                lw      a4,-20(s0)
3800005c:       00a00793                li      a5,10
38000060:       fce7dce3                bge     a5,a4,38000038 <initfir+0x14>
38000064:       00000013                nop
38000068:       00000013                nop
3800006c:       01c12403                lw      s0,28(sp)
38000070:       02010113                addi    sp,sp,32
38000074:       00008067                ret

38000078 <fir>:
38000078:       fe010113                addi    sp,sp,-32
3800007c:       00112e23                sw      ra,28(sp)
38000080:       00812c23                sw      s0,24(sp)
38000084:       02010413                addi    s0,sp,32
38000088:       f9dff0ef                jal     ra,38000024 <initfir>
3800008c:       fe042623                sw      zero,-20(s0)
```

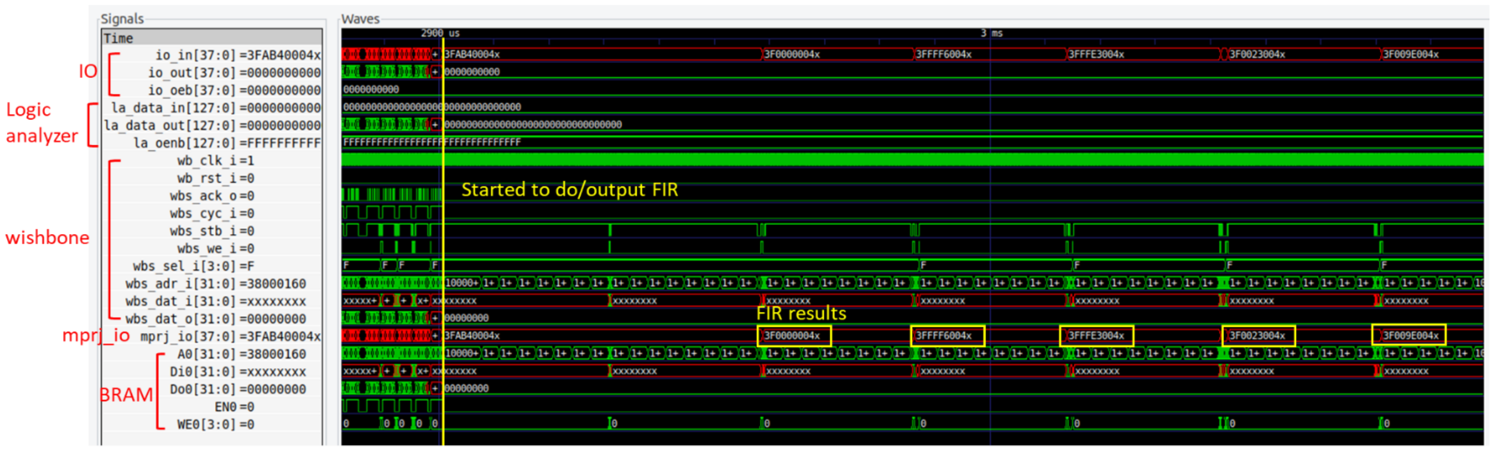


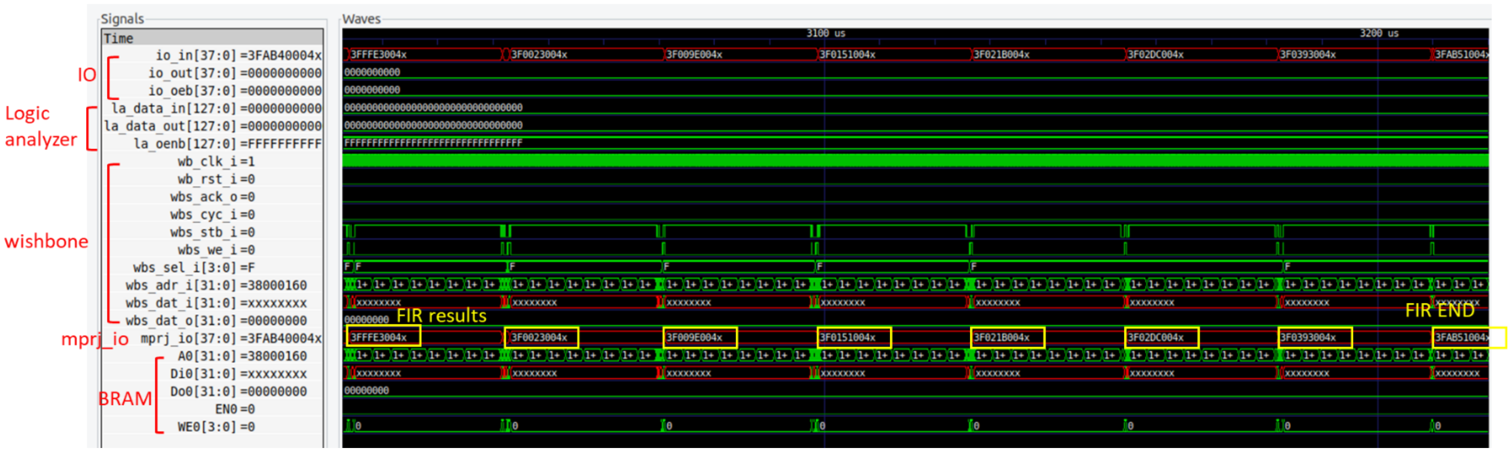
```
38000090:	fe042423	sw	zero,-24(s0)
38000094:	0a40006f	j	38000138 <fir+0xc0>
38000098:	fe042623	sw	zero,-20(s0)
3800009c:	fe042223	sw	zero,-28(s0)
380000a0:	0680006f	j	38000108 <fir+0x90>
380000a4:	00a00713	li	a4,10
380000a8:	fe842783	lw	a5,-24(s0)
380000ac:	40f70733	sub	a4,a4,a5
380000b0:	fe442783	lw	a5,-28(s0)
380000b4:	00f707b3	add	a5,a4,a5
380000b8:	00000713	li	a4,0
380000bc:	00279793	slli	a5,a5,0x2
380000c0:	00f707b3	add	a5,a4,a5
380000c4:	0007a683	lw	a3,0(a5)
380000c8:	02c00713	li	a4,44
380000cc:	fe442783	lw	a5,-28(s0)
380000d0:	00279793	slli	a5,a5,0x2
380000d4:	00f707b3	add	a5,a4,a5
380000d8:	0007a783	lw	a5,0(a5)
380000dc:	00078593	mv	a1,a5
380000e0:	00068513	mv	a0,a3
380000e4:	f1dff0ef	jal	ra,38000000 <__mulsi3>
380000e8:	00050793	mv	a5,a0
380000ec:	00078713	mv	a4,a5
380000f0:	fec42783	lw	a5,-20(s0)
380000f4:	00e787b3	add	a5,a5,a4
380000f8:	fef42623	sw	a5,-20(s0)
380000fc:	fe442783	lw	a5,-28(s0)
38000100:	00178793	addi	a5,a5,1
38000104:	fef42223	sw	a5,-28(s0)
38000108:	fe442703	lw	a4,-28(s0)
3800010c:	fe842783	lw	a5,-24(s0)
38000110:	f8e7dae3	bge	a5,a4,380000a4 <fir+0x2c>
38000114:	08800713	li	a4,136
38000118:	fe842783	lw	a5,-24(s0)
3800011c:	00279793	slli	a5,a5,0x2
38000120:	00f707b3	add	a5,a4,a5
38000124:	fec42703	lw	a4,-20(s0)
38000128:	00e7a023	sw	a4,0(a5)
3800012c:	fe842783	lw	a5,-24(s0)
38000130:	00178793	addi	a5,a5,1
38000134:	fef42423	sw	a5,-24(s0)
38000138:	fe842703	lw	a4,-24(s0)
3800013c:	00a00793	li	a5,10
38000140:	f4e7dce3	bge	a5,a4,38000098 <fir+0x20>
38000144:	08800793	li	a5,136
38000148:	00078513	mv	a0,a5
3800014c:	01c12083	lw	ra,28(sp)
38000150:	01812403	lw	s0,24(sp)
38000154:	02010113	addi	sp,sp,32
38000158:	00008067	ret
```

## 2. Interface between BRAM and wishbone

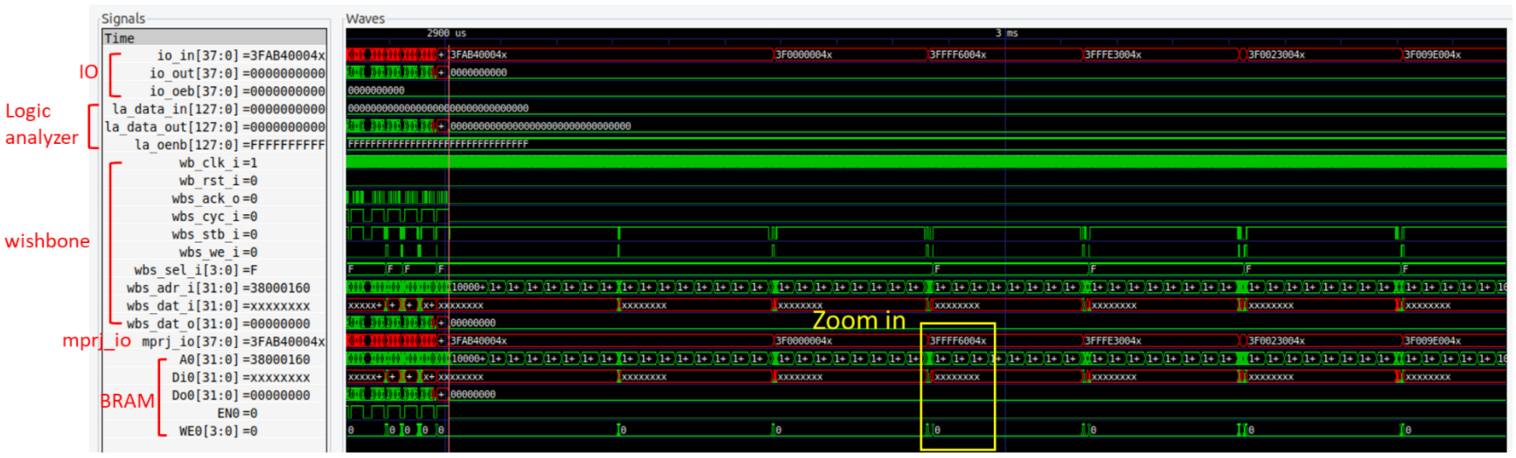
### A. Waveform from xsim

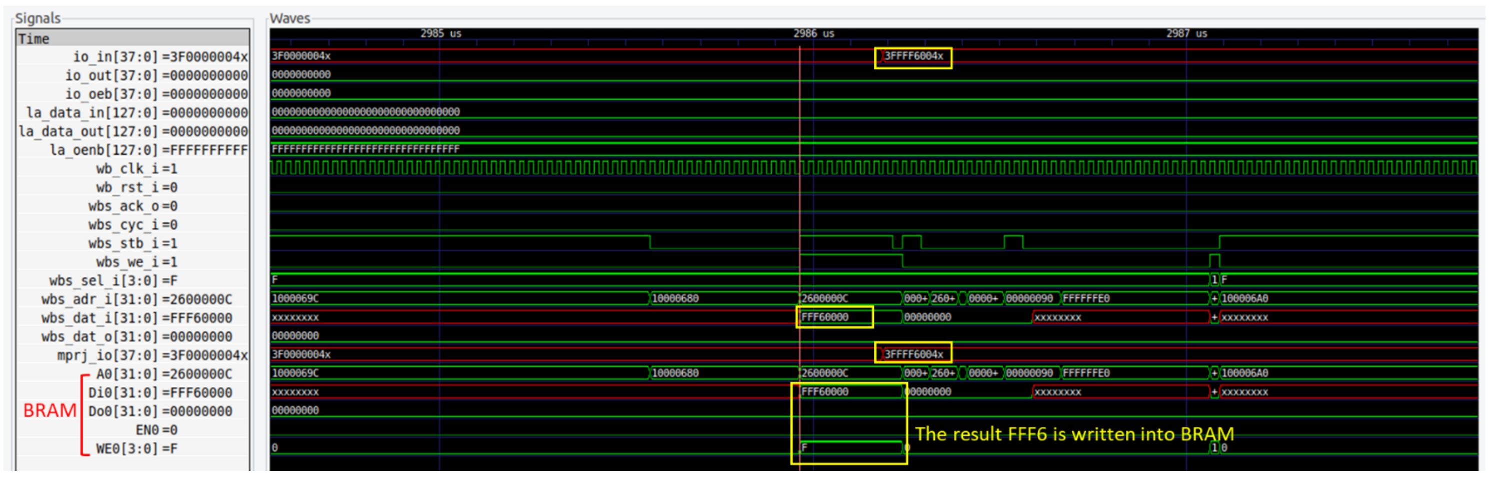
If mprj_io[31:16] == 16’hAB40, it indicate that the CPU is starting to do FIR. The wishbone (wbs_dat_i) will send the FIR result (i.e., outputsignal[N]) to this interface. After writing the result to the BRAM, the result will output to testbench via mprj_io[31:16].





Zoom in to show the BRAM write operation:





The received data FFF6 is written to BRAM, and output to testbench via mprj_io[31:16].

## B. Design

- Interface clk & rst

  Within this interface block, the clock and reset signals are defined by la_oneb[64] and la_oneb[65]. If la_oneb[64]/la_oneb[65] is 0, then clk/rst corresponds to la_data_in[64]/ la_data_in[65]. Otherwise, clk/rst is mapped to wb_clk_i/wb_rst_i.

- Wishbone and BRAM

  Wishbone transmits the data to be stored in BRAM via the signal wbs_dat_i with the corresponding address wbs_adr_i. The signal wbs_dat_o is the output of user BRAM which must be sent back to wishbone. To determine whether the BRAM fetch operation is neccessary, the EN signal of the BRAM corresponds to wbs_cyc_i and wbs_stb_i, which we refer to as "valid" in our design. The WE is defined as wbs_sel_i & {4{wbs_we_i}}.

  The signal wbs_ack_o needs to be delayed 10 cycles after fetching the BRAM, i.e., after the EN of BRAM goes high. We employ a 10-stage FIFO to delay the valid signal. After sending the activated valid signal, the FIFO is initialized with all 0s.

- Logic analyzer

  The output to logic analyzer la_data_out is connected to the output of BRAM.

- IO

  The output to IO io_out is also connected to the output of BRAM. The IO interface's signal io_oeb is the reset signal in this block.

- IRQ

  The IRQ signal is unused and is consistently set to 0.

## 3. Synthesis report

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

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |   13 |     0 |          0 |     53200 |  0.02 |
|   LUT as Logic          |   13 |     0 |          0 |     53200 |  0.02 |
|   LUT as Memory         |    0 |     0 |          0 |     17400 |  0.00 |
| Slice Registers         |   10 |     0 |          0 |    106400 | <0.01 |
|   Register as Flip Flop |   10 |     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    |    2 |     0 |          0 |       140 |  1.43 |
|   RAMB36/FIFO*    |    2 |     0 |          0 |       140 |  1.43 |
|     RAMB36E1 only |    2 |       |            |           |       |
|   RAMB18          |    0 |     0 |          0 |       280 |  0.00 |
+-------------------+------+-------+------------+-----------+-------+
```