Final Project Report

SOC Design

系統晶片設計

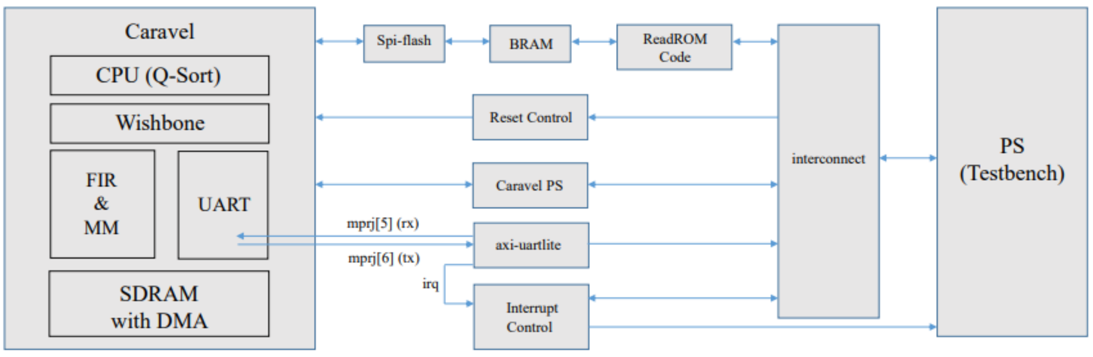
Group 7

112061603 李柏叡

112061620 郭邑哲

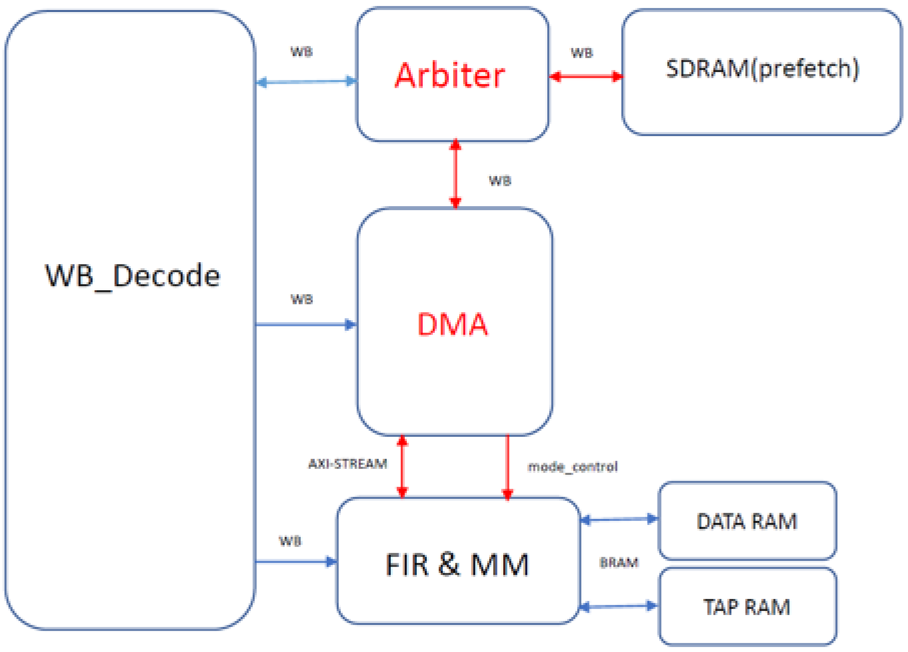
112061621 貢暐家

一、Block diagram :



二、DMA&Arbiter :

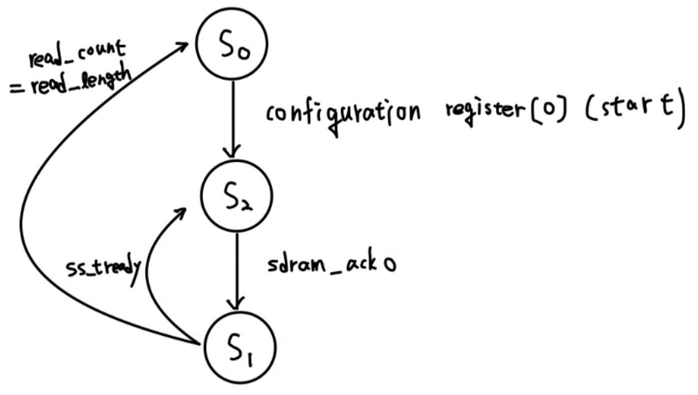
- DMA&Arbiter diagram:



- 1 input buffer and 1 output buffer:

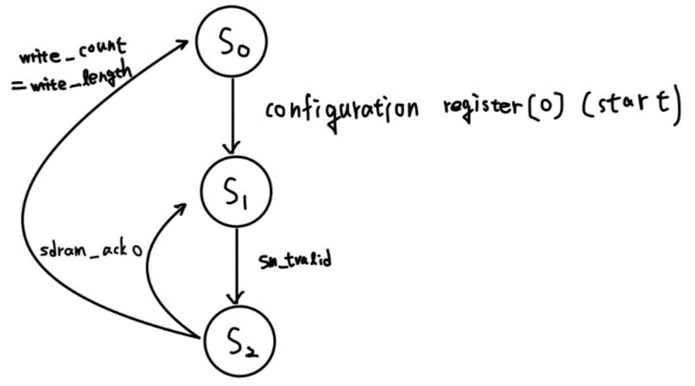
  Input buffer:

S_0 : Idle
S_1 : wait Fir_MM read (ss_tdata = Inputbuffer)
S_2 : pull read request to SDRAM (Input buffer = SDRAM_data_o)



  Output buffer:

S_0 : Idle
S_1 : wait Fir_MM write (Outputbuffer = sm_tdata)
S_2 : pull write request to SDRAM (SDRAM_data_i = Output buffer)



- DMA Configuration register map:

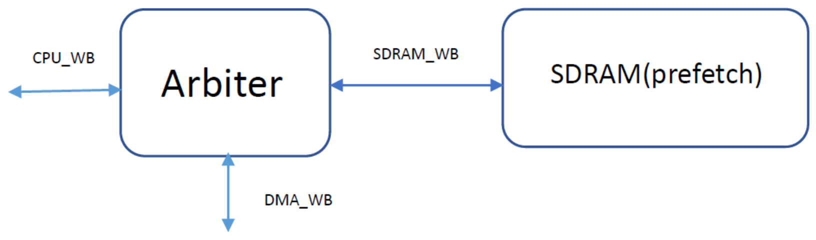
When the fir_mm hardware requires data, it does not rely on the CPU for data read or write operations. Instead, it can directly access memory through DMA, thereby accelerating computation speed and enhancing memory access efficiency.

address	bit	
0X380002AC	[0] [1]	Start Idle
0X380002B0	[31:0]	Read starting address
0X380002B4	[31:0]	Read length
0X380002B8	[31:0]	Write starting address
0X380002BC	[31:0]	Write length

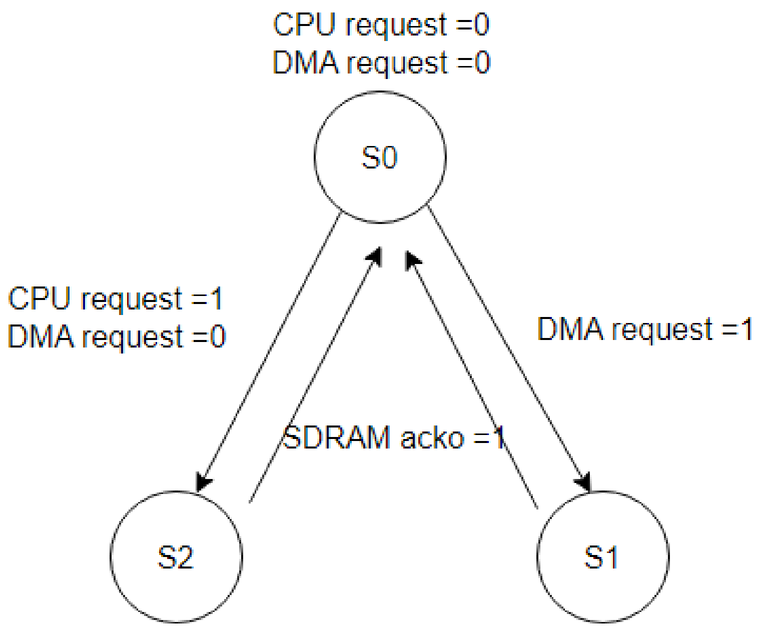
- Arbiter:

The main function of the arbiter is to determine whether DMA or the CPU should access memory at a given moment, enabling concurrent execution of both.

When DMA and CPU both have requests to memory at the same time, arbiter prioritized handling requests from DMA.



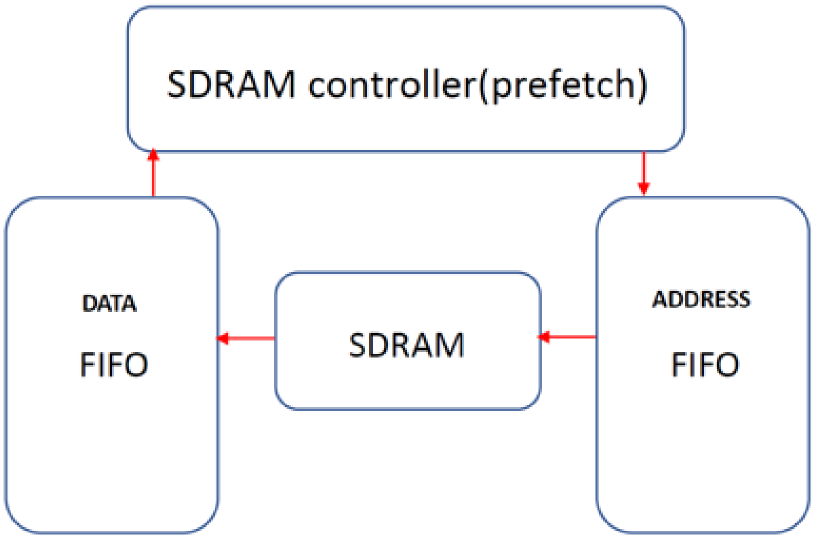
State diagram:



三、SDRAM with Pre-fetch :

User Project SDRAM address map	
Bank0	Qsort instruction
Bank1	Fir & MM data
Bank2	Fir & MM data
Bank3	Qsort data

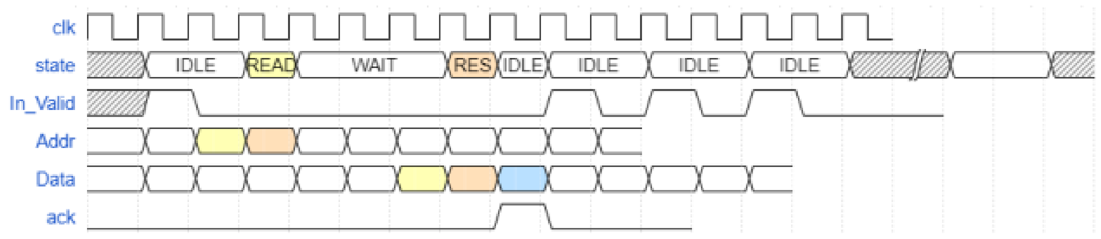
- SDRAM with Pre-fetch diagram:



- Waveform view:

In order to implement the prefetch functionality, three prefetch buffers have been introduced. These prefetch buffers are designed to anticipate the next instruction, utilizing the three WAIT states following the READ state. If the prediction is successful, the instruction can be read out after only two clock cycles. However, in case of prediction failure, it still requires 8 cycles, and the system proceeds to the next prefetch.

Without Pre-fetch : 32 clocks are required to execute 4 instructions

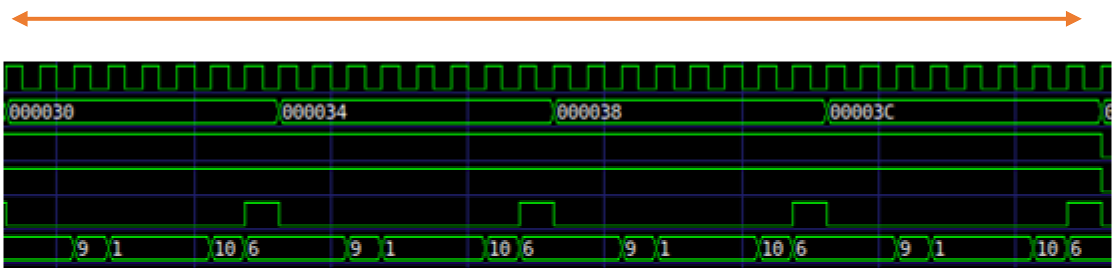


🟨 Addr : Access address
Data : get data

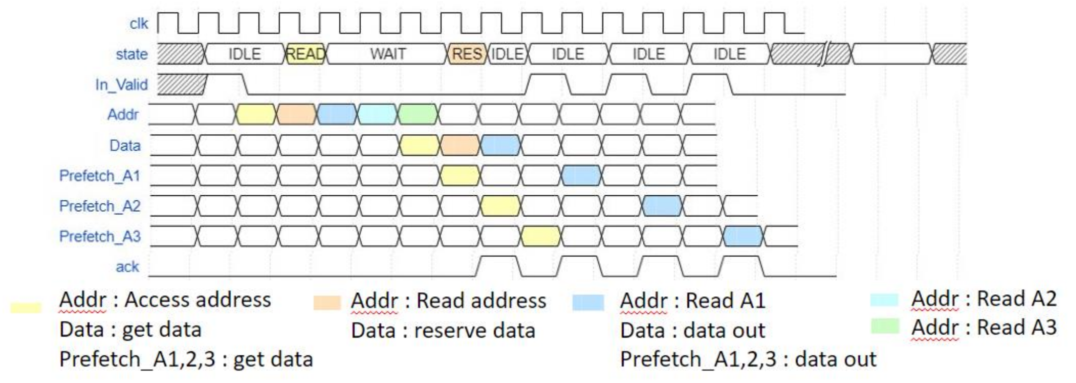
🟧 Addr : Read address
Data : reserve data

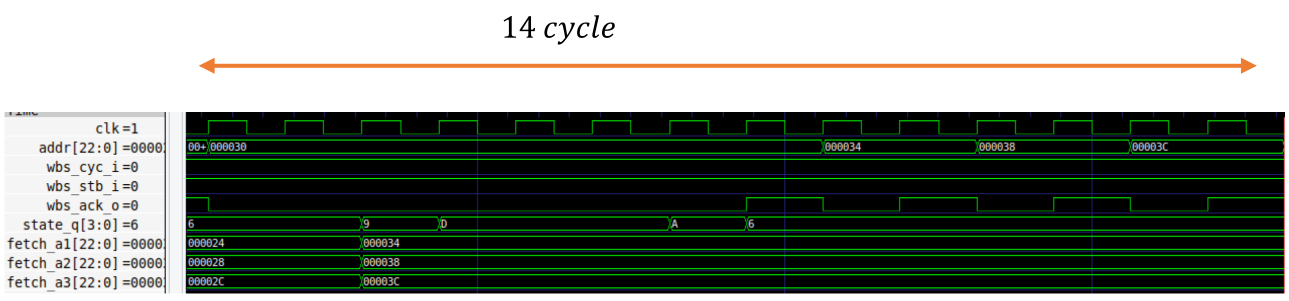
🟦 Data : data out

$32cycle$



With Pre-fetch : 14 clocks are required to execute 4 instructions

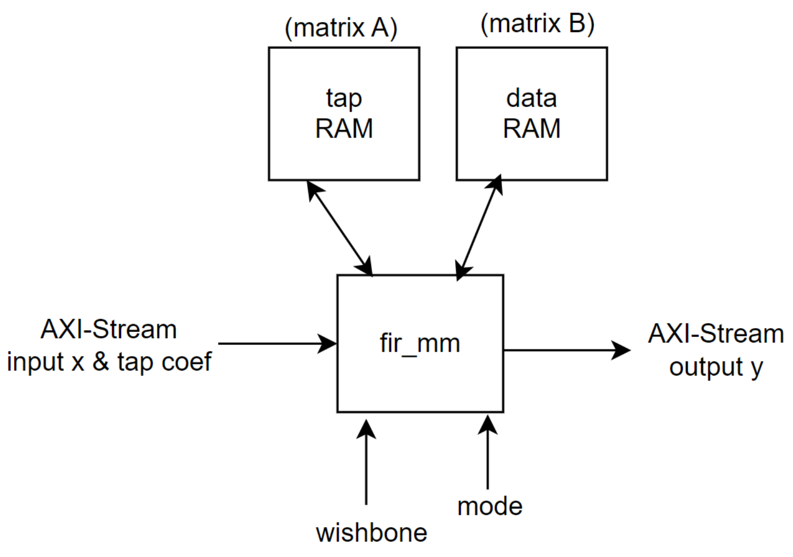




四、Fir & Matrix multiplication

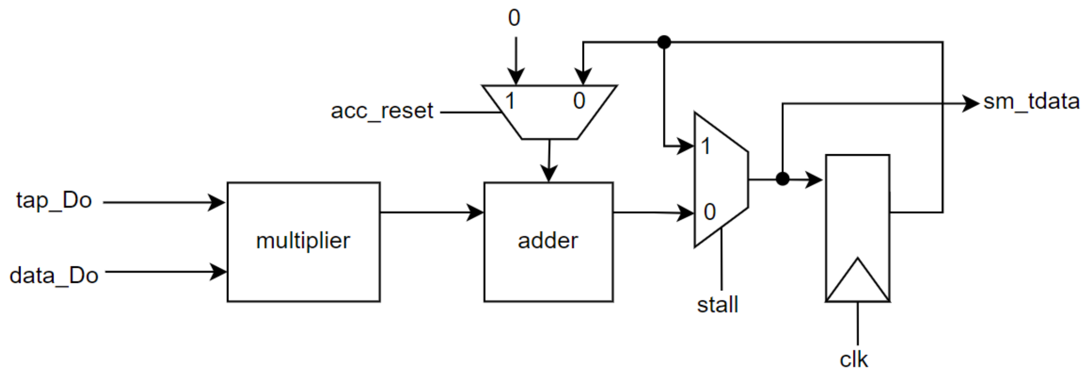
Both Fir and Matrix multiplication need multiply-add operation. So, we decided to make an accelerator to do fir and matrix multiplication.
Since we want to store matrix A and matrix B in tap ram and data ram, we set the size of these two RAMs to 16 DW.

- Fir & Matrix multiplication diagram:



- Fir & Matrix multiplication architecture:

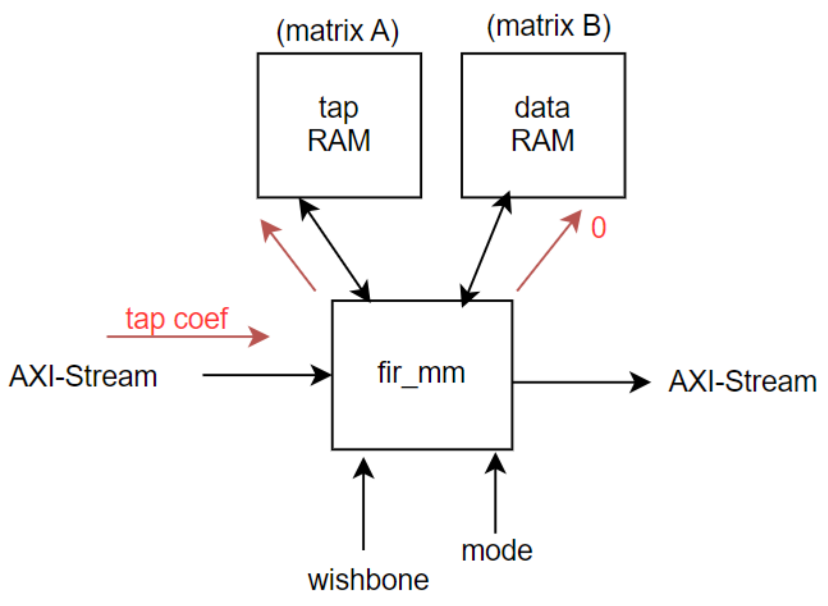
Using 1 multiplier and 1 adder



- Fir & Matrix multiplication mode:

I. Set tap mode:

Input tap coefficient by using AXI-Stream and store the coefficient in tap RAM . Use a counter to count from 0 to 10 (the counter increases when Fir_mm receives 1 coefficient), then Fir_mm returns to IDLE state.



II. FIR mode:

--Do Fir (output a data needs 11 cycles)

-- Use three counters:

- Tap_idx: counting the 11 cycles needed to produce 1 output
- Data_idx: count how many input x received, when it reaches data_length, fir_mm return to idle state
- Data_A_shift: used to calculate data_RADDR

III. Matrix multiplication mode:

--Input matrix A and matrix B using AXI-Stream. (max speed: 32 cycles)

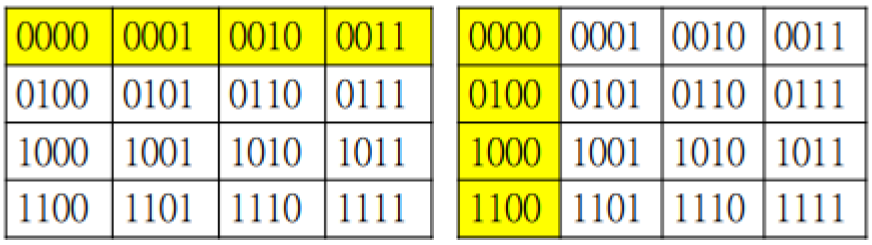
--Store matrix A in tap RAM and store matrix B in data RAM.

--After both matrixes are stored in RAM, start the computation.(max speed: 64 cycles)

-- Use two counters to determine read address:

Tap_idx : count from 0 to 15, then goes back to 0

Data_idx : starts from 0x010, increase 1 every time tap_idx return to 0



0000	0001	0010	0011
0100	0101	0110	0111
1000	1001	1010	1011
1100	1101	1110	1111

MATRIX_A
(tap RAM)

0000	0001	0010	0011
0100	0101	0110	0111
1000	1001	1010	1011
1100	1101	1110	1111

MATRIX_B
(data_RAM)

```
tap_RADDR = {data_idx[2],data_idx[0],tap_idx[1:0]};
            data_RADDR = {tap_idx[1:0],tap_idx[3:2]};
```

IV. Simulation State cycles:

With qsort concurrently executing:

State 1(SET TAP Mode):

92 cycle

State 2(FIR Mode):

916 cycle

State 3(MM Mode):

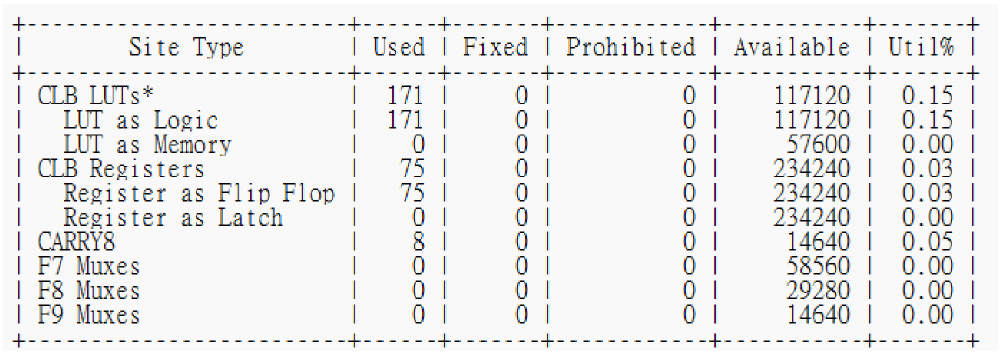
335 cycle

Total (FIR & MM) finish:

1343 cycle

● Fir & Matrix multiplication synthesis result:

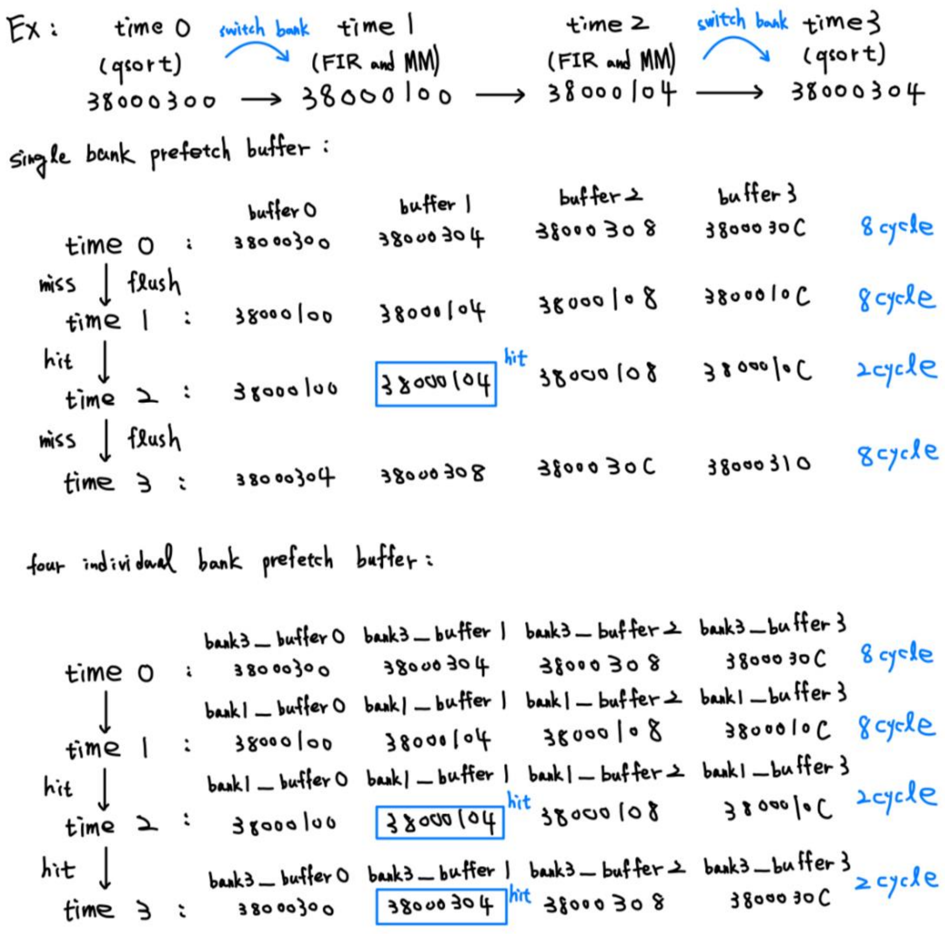
No extra register is needed for the original Fir to support Matrix-multiplication.



```
+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| CLB LUTs*               |  171 |     0 |          0 |    117120 |  0.15 |
|   LUT as Logic          |  171 |     0 |          0 |    117120 |  0.15 |
|   LUT as Memory         |    0 |     0 |          0 |     57600 |  0.00 |
| CLB Registers           |   75 |     0 |          0 |    234240 |  0.03 |
|   Register as Flip Flop |   75 |     0 |          0 |    234240 |  0.03 |
|   Register as Latch     |    0 |     0 |          0 |    234240 |  0.00 |
| CARRY8                  |    8 |     0 |          0 |     14640 |  0.05 |
| F7 Muxes                |    0 |     0 |          0 |     58560 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |     29280 |  0.00 |
| F9 Muxes                |    0 |     0 |          0 |     14640 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

- Observation

Adding prefetch buffers to each individual bank enables successful memory access prefetching even when branching across different banks (Due to parallel execution of software and hardware, there is a probability of continuously switching bank address)



Ex : time 0 (qsort) 38000300 → switch bank → time 1 (FIR and MM) 38000100 → time 2 (FIR and MM) 38000104 → switch bank → time 3 (qsort) 38000304

single bank prefetch buffer :

	buffer 0	buffer 1	buffer 2	buffer 3	
time 0 :	38000300	38000304	38000308	3800030C	8 cycle
miss ↓ flush					
time 1 :	38000100	38000104	38000108	3800010C	8 cycle
hit ↓					
time 2 :	38000100	$\boxed{38000104}$ hit	38000108	3800010C	2 cycle
miss ↓ flush					
time 3 :	38000304	38000308	3800030C	38000310	8 cycle

four individual bank prefetch buffer :

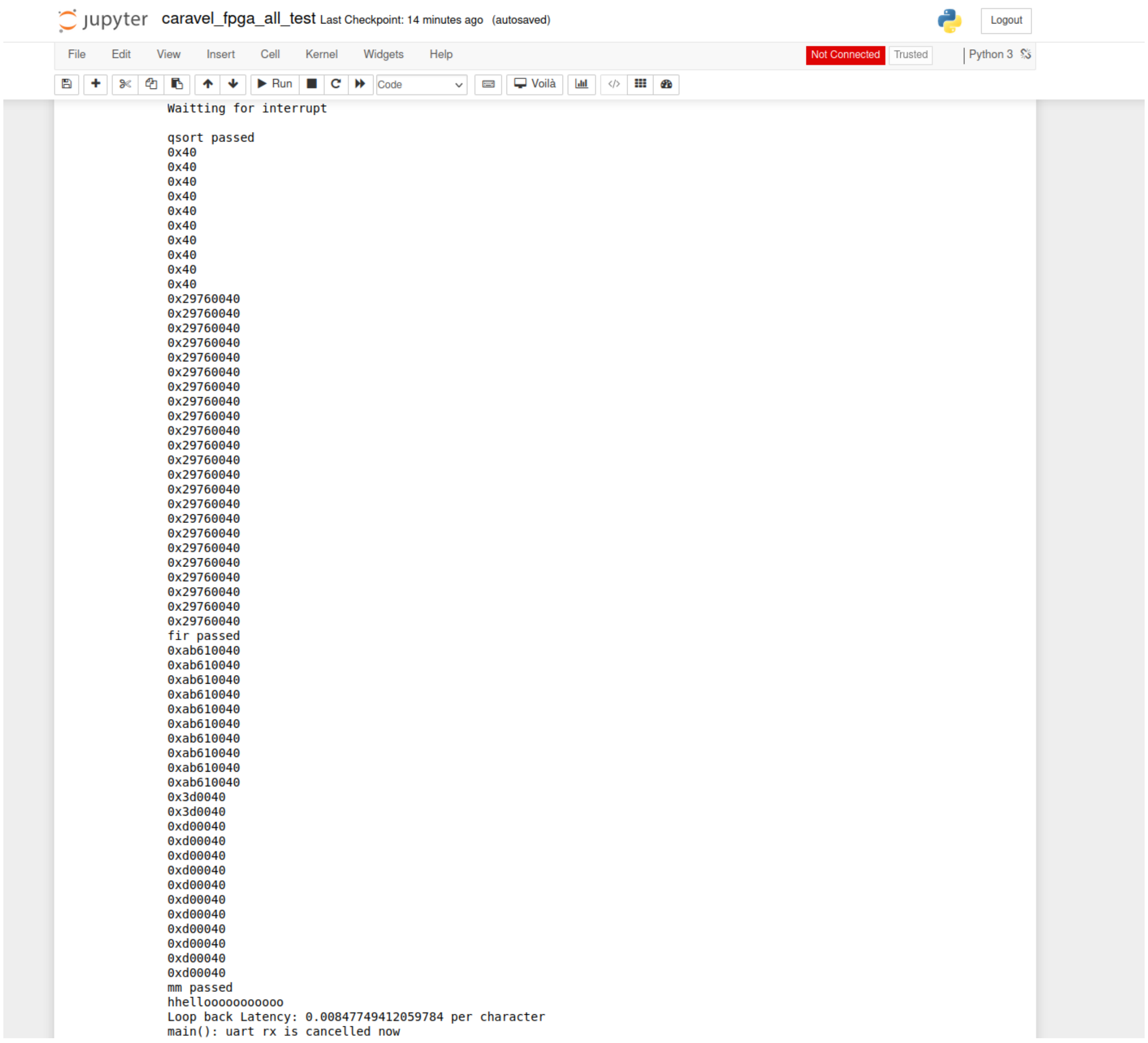
time 0 :	bank3 _ buffer 0 38000300	bank3 _ buffer 1 38000304	bank3 _ buffer 2 38000308	bank3 _ buffer 3 3800030C	8 cycle
↓					
time 1 :	bank1 _ buffer 0 38000100	bank1 _ buffer 1 38000104	bank1 _ buffer 2 38000108	bank1 _ buffer 3 3800010C	8 cycle
hit ↓					
time 2 :	bank1 _ buffer 0 38000100	bank1 _ buffer 1 $\boxed{38000104}$ hit	bank1 _ buffer 2 38000108	bank1 _ buffer 3 3800010C	2 cycle
hit ↓					
time 3 :	bank3 _ buffer 0 38000300	bank3 _ buffer 1 $\boxed{38000304}$ hit	bank3 _ buffer 2 38000308	bank3 _ buffer 3 3800030C	2 cycle

Number of cycles required to finish FIR MM QSORT:

Single bank prefetch buffer : 6652 cycle

Four individual bank prefetch buffer : 5773 cycle

- FPGA result


jupyter caravel_fpga_all_test Last Checkpoint: 14 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Connected
Trusted
Python 3
Run
Code
Voilà

```
Waitting for interrupt

qsort passed
0x40
0x40
0x40
0x40
0x40
0x40
0x40
0x40
0x40
0x40
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
0x29760040
fir passed
0xab610040
0xab610040
0xab610040
0xab610040
0xab610040
0xab610040
0xab610040
0xab610040
0xab610040
0xab610040
0x3d0040
0x3d0040
0xd00040
0xd00040
0xd00040
0xd00040
0xd00040
0xd00040
0xd00040
0xd00040
0xd00040
0xd00040
0xd00040
mm passed
hhelloooooooooo
Loop back Latency: 0.00847749412059784 per character
main(): uart rx is cancelled now
```

- Github link

https://github.com/ruei7916/SOC-final-project