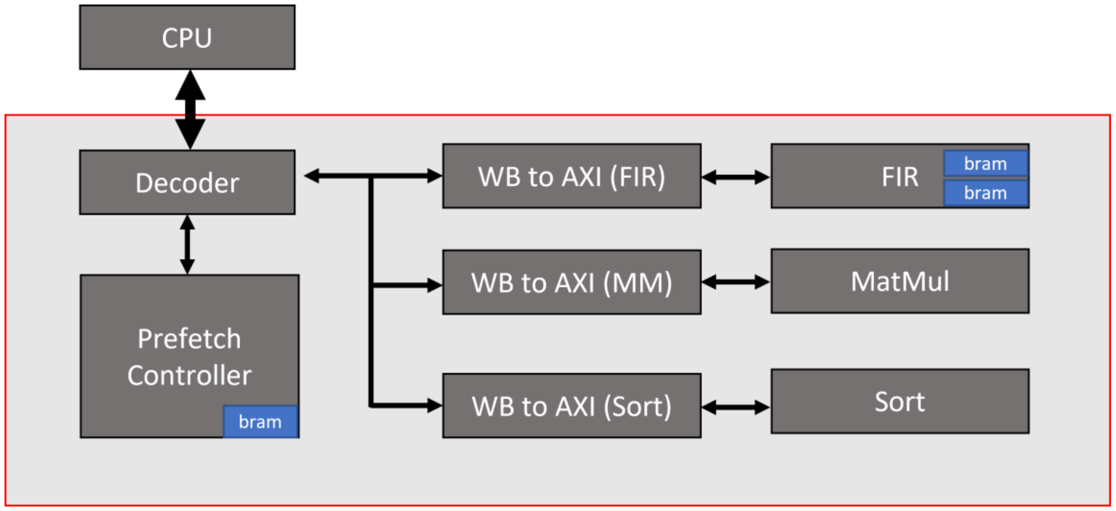
SoC Final Project Group 8

李承澔 梁凱傑 張豐餘

- Overall Hardware Architecture



We designed several modules for the hardware accelerator:

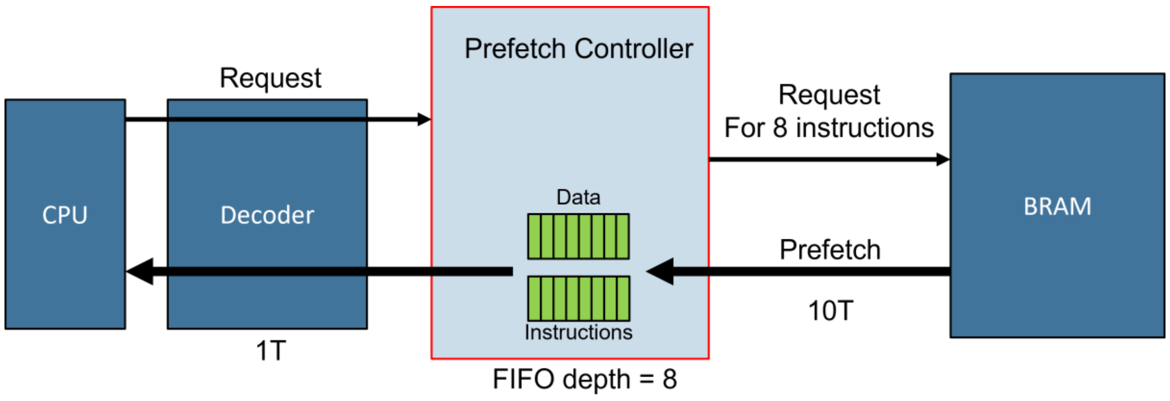
1. Decoder:
It's a "relay station" for the signal communication between the CPU and all the designed hardware.

2. Prefetch Controller:
Note that one instruction need 10T after request, so we implemented a pipelined prefetch controller to make it faster for instructions with consecutive address.

3. WB to AXI:
Since we designed the computational module for AXI protocol at the beginning, so we need to transfer the handshake signals.

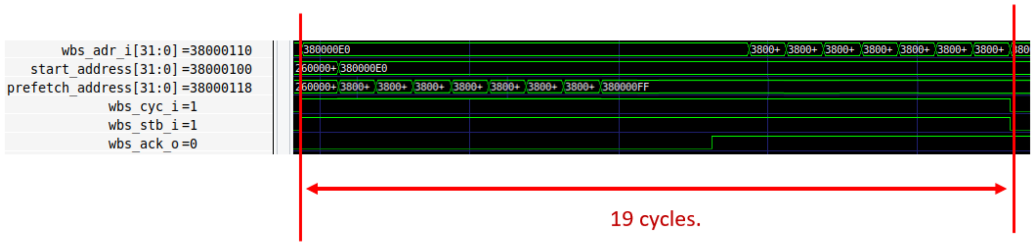
4. FIR, MatMul, Sort:
These modules use hardware resource to achieve faster computation process for FIR, matrix multiplication, and sorting algorithm.

- Prefetch Scheme



We observed that CPU often send at most 8 instructions with consecutive address. Hence, the prefetch buffer is set to 8, and once the CPU sends the request, the prefetch controller will then send 8 request cycle by cycle. After 10 cycles, the controller will store the received instructions and data from the BRAM to the FIFO, and then send back to the CPU. Once the address does not match, the FIFO will be cleared.

Hence, we can read 8 instructions with 19 cycles when all the instructions hit.


wbs_adr_i[31:0] =38000110
start_address[31:0] =38000100
prefetch_address[31:0] =38000118
wbs_cyc_i =1
wbs_stb_i =1
wbs_ack_o =0
380000E0
3800+
260000+
380000E0
380000FF
19 cycles.

- ## Memory Layout

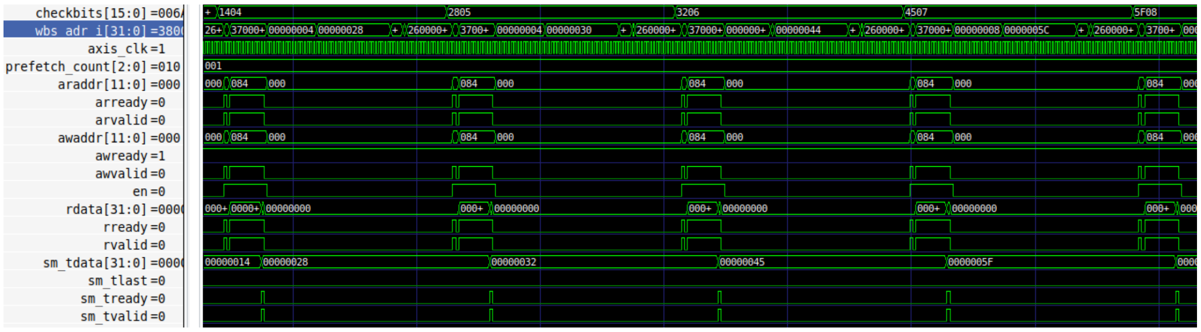
Sections	Base Address
Execution memory	0x3800 0000
UART	0x3000 0000
Data	0x3400 0000
Sort	0x3500 0000
MatMul	0x3600 0000
FIR	0x3700 0000

- ## FIR Register Address

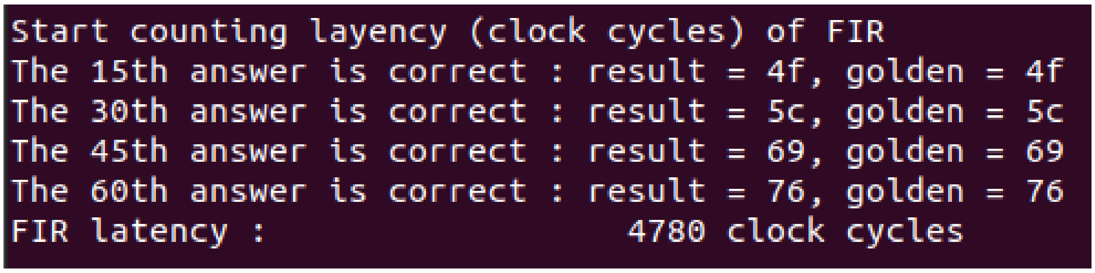
Address		Data
0x00	Bit 0	ap_start
	Bit 1	ap_idle
	Bit 2	ap_done
	Bit 3	Ready for input
	Bit 4	Ready for Output
0x10		Length
0x20 ~ 0x5F		Tap coef.
0x80		Input
0x84		Output

- FIR Improvement

In our hardware accelerator with the optimization, we reduce the cycles for each computation to **74 cycles on average.**



The baseline in lab6 is **287208** clock cycles. While the total cycles here is **4780**. The start flag of FIR is **0x00A5**, while the end flag is **0x005A.** We also check the 15th, 30th ,45th ,and 60th result in the testbench to make sure the computation is correct.



```
Start counting layency (clock cycles) of FIR
The 15th answer is correct : result = 4f, golden = 4f
The 30th answer is correct : result = 5c, golden = 5c
The 45th answer is correct : result = 69, golden = 69
The 60th answer is correct : result = 76, golden = 76
FIR latency :                    4780 clock cycles
```

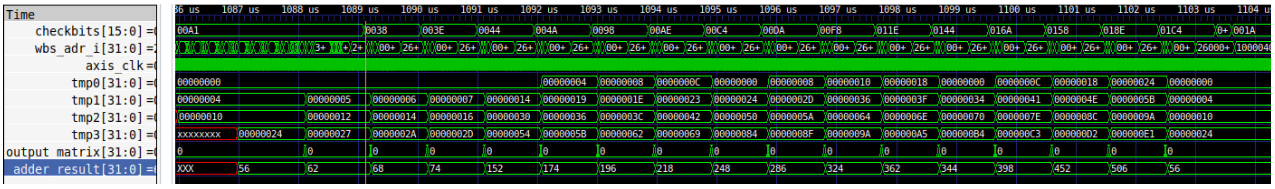
The performance improvement is around **60.1X**

- MatMul Register Address

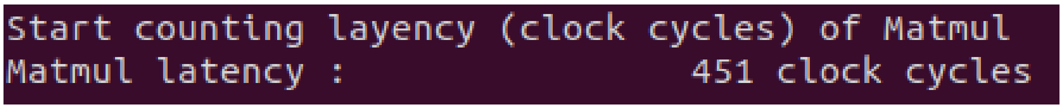
Address		Data
0xF0	Bit 0	ap_start
	Bit 1	ap_idle
	Bit 2	ap_done
	Bit 3	Ready for input
	Bit 4	Ready for Output
0x00 ~ 0x3F		Matrix A
0x40 ~ 0x7F		Matrix B
0x80		Matrix C (stream out)

- MatMul Improvement

The result will put in the checkbits as in the figure, which we can see the computation is correct.



The baseline in lab6 is **74102** clock cycles. While the total cycles used here is **451**. The start flag of MatMul is **0x00A1**, while the end flag is **0x001A**



```
Start counting layency (clock cycles) of Matmul
Matmul latency :                 451 clock cycles
```

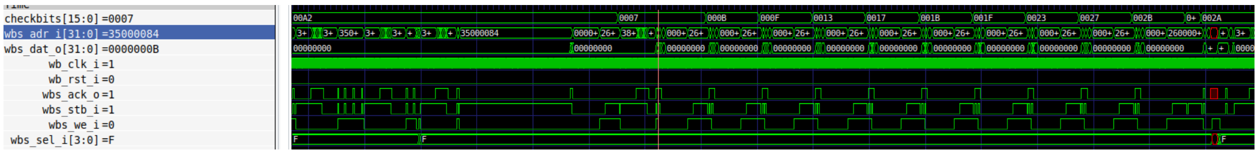
The performance improvement is around **164.3X**

- ## Sort Register Address

Address		Data
0x00	Bit 0	ap_start
	Bit 1	ap_idle
	Bit 2	ap_done
	Bit 3	Ready for input
	Bit 4	Ready for Output
0x80		Input
0x84		Output

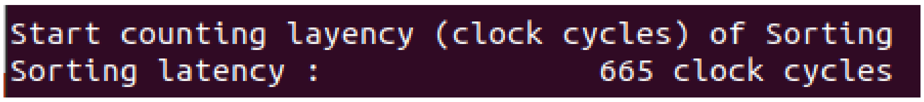
- ## Sort Improvement

The result will put in the checkbits as in the figure, which we can see the computation is correct.
The original data is : {2B, 27, 23, 1D, 1B, 17, 13, 0F, 0B, 07}



```
checkbits[15:0] =0007
wbs_adr_i[31:0] =35000084
wbs_dat_o[31:0] =0000000B
      wb_clk_i =1
      wb_rst_i =0
     wbs_ack_o =1
     wbs_stb_i =1
      wbs_we_i =0
 wbs_sel_i[3:0] =F
```

The baseline in lab6 is **24305** clock cycles. While the total cycles here is **665**. The start flag of Sort is **0x00A2**, while the end flag is **0x002A**

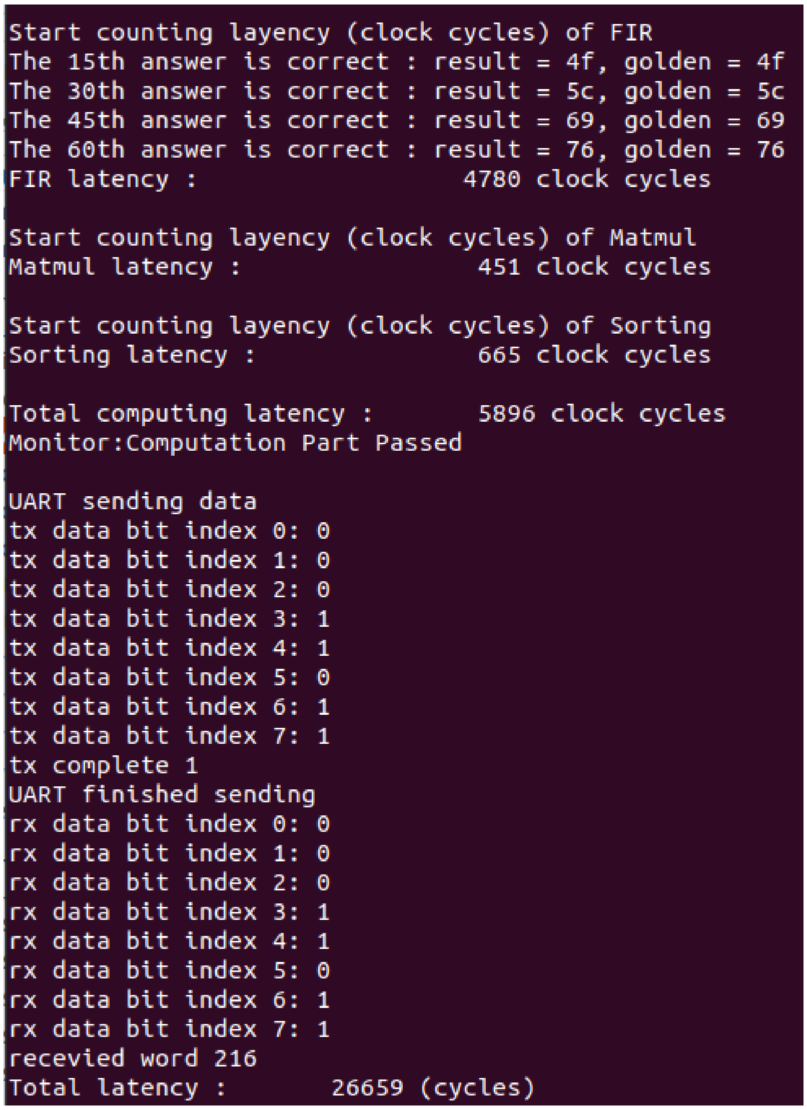


```
Start counting layency (clock cycles) of Sorting
Sorting latency :                665 clock cycles
```

The performance improvement is around **36.5X**

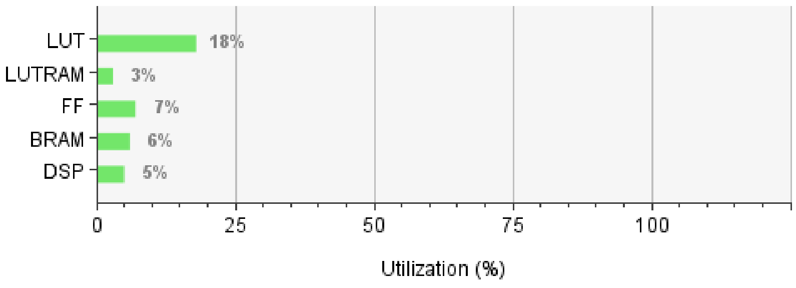
- Overall Result of Testbench

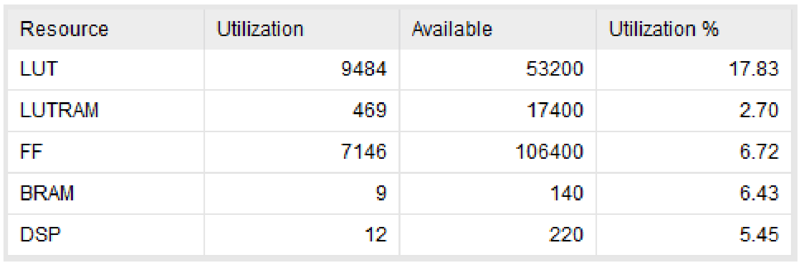
Task	Our work	Improvement
FIR	4780 cycles	60.1
MatMul	451 cycles	164.3
Sorting	665 cycles	36.5



```
Start counting layency (clock cycles) of FIR
The 15th answer is correct : result = 4f, golden = 4f
The 30th answer is correct : result = 5c, golden = 5c
The 45th answer is correct : result = 69, golden = 69
The 60th answer is correct : result = 76, golden = 76
FIR latency :                   4780 clock cycles

Start counting layency (clock cycles) of Matmul
Matmul latency :                 451 clock cycles

Start counting layency (clock cycles) of Sorting
Sorting latency :                665 clock cycles

Total computing latency :        5896 clock cycles
Monitor:Computation Part Passed

UART sending data
tx data bit index 0: 0
tx data bit index 1: 0
tx data bit index 2: 0
tx data bit index 3: 1
tx data bit index 4: 1
tx data bit index 5: 0
tx data bit index 6: 1
tx data bit index 7: 1
tx complete 1
UART finished sending
rx data bit index 0: 0
rx data bit index 1: 0
rx data bit index 2: 0
rx data bit index 3: 1
rx data bit index 4: 1
rx data bit index 5: 0
rx data bit index 6: 1
rx data bit index 7: 1
recevied word 216
Total latency :       26659 (cycles)
```

- ## Utilization Report





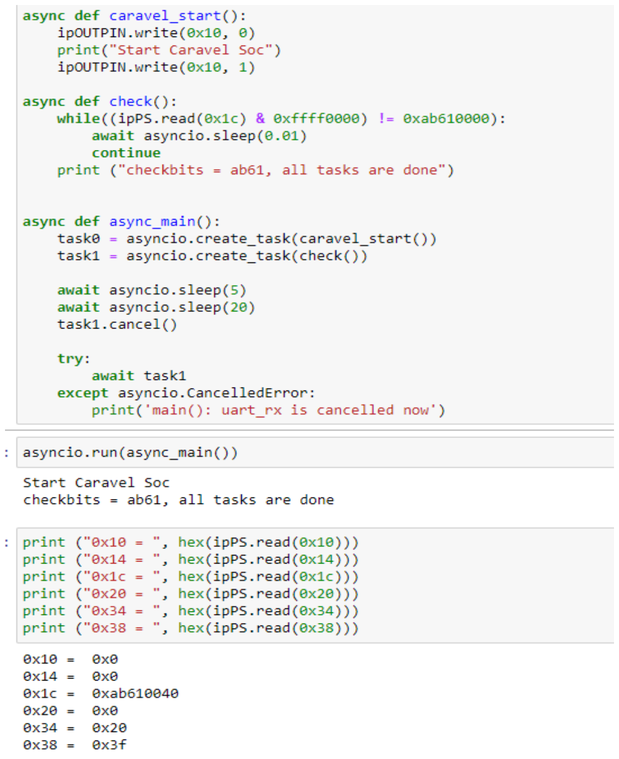
Resource	Utilization	Available	Utilization %
LUT	9484	53200	17.83
LUTRAM	469	17400	2.70
FF	7146	106400	6.72
BRAM	9	140	6.43
DSP	12	220	5.45

We can see the resource is still sufficient after adding hardware accelerators. The resource overhead is not too much

- ## FPGA Test on the PS

We also do the test for FPGA on the Jupyter Notebook.
From the result, we can see the checkbits is correct.

Note that we didn't do anything to optimize the UART, so we skip the test on the UART. (We still have confirmed the function is correct.)



```
async def caravel_start():
    ipOUTPIN.write(0x10, 0)
    print("Start Caravel Soc")
    ipOUTPIN.write(0x10, 1)

async def check():
    while((ipPS.read(0x1c) & 0xffff0000) != 0xab610000):
        await asyncio.sleep(0.01)
        continue
    print ("checkbits = ab61, all tasks are done")


async def async_main():
    task0 = asyncio.create_task(caravel_start())
    task1 = asyncio.create_task(check())

    await asyncio.sleep(5)
    await asyncio.sleep(20)
    task1.cancel()

    try:
        await task1
    except asyncio.CancelledError:
        print('main(): uart_rx is cancelled now')
```

```
: asyncio.run(async_main())
```

```
Start Caravel Soc
checkbits = ab61, all tasks are done
```

```
: print ("0x10 = ", hex(ipPS.read(0x10)))
print ("0x14 = ", hex(ipPS.read(0x14)))
print ("0x1c = ", hex(ipPS.read(0x1c)))
print ("0x20 = ", hex(ipPS.read(0x20)))
print ("0x34 = ", hex(ipPS.read(0x34)))
print ("0x38 = ", hex(ipPS.read(0x38)))
```

```
0x10 =  0x0
0x14 =  0x0
0x1c =  0xab610040
0x20 =  0x0
0x34 =  0x20
0x38 =  0x3f
```

- Future Work

1. DMA with SDRAM and controller:
From the performance enhancement, we think the current bottleneck of our work is the data access time. Using a faster SDRAM with DMA can help solve the issues.

2. Assembly code arrangement:
We have discovered several instruction hazards in the assembly code, we may rearrange the instructions to improve those hazards.

3. UART with FIFO.
The baud rate of UART is relatively slow compared to the clock frequency.
It can be a bottleneck for UART to interrupt. Adding a FIFO buffer might help to migrate the problems.

- Github Links
https://github.com/JackeyUi/SoC_Final

分工:

李承澔：
Design the Hardware of Matrix Multiplication

梁凱傑：
Design the Hardware of FIR, Prefetch Controller.
Integrate All Hardware, Report and Presentation.

張豐餘：
Design the Hardware of Sorting.

.