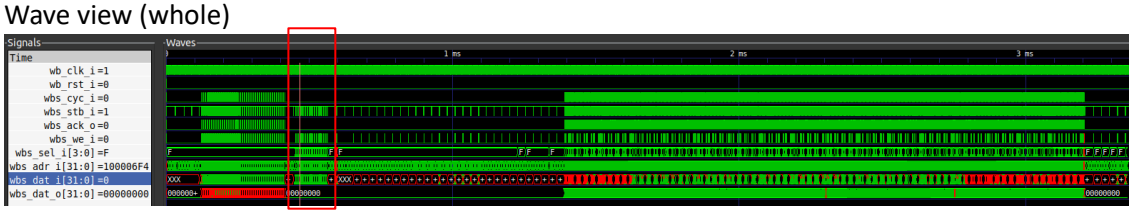
SOC Lab6 Report

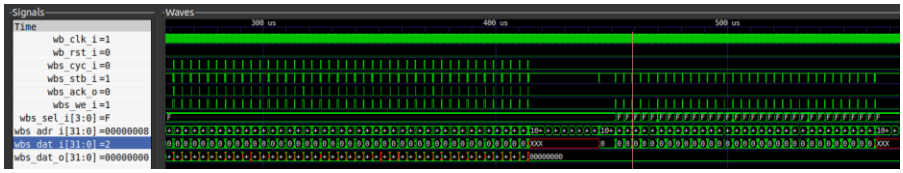
Group 3

110590022 陳冠晰 110000107 陳柏翰 110061217 王彥智

- Firmware simulation
  - Matrix Multiplication
    - Wave view (whole)



    - Wave view (write data in)



    - Wave view (check if the data written is correct)
      When watching the waveform, we have a question.
      Why the initialization of the array is written to the address 0x00000000?
      Then we found out that the array used in the firmware is stored in the memory in “dff,” which address is 0x00000000 (from firmware/section.ids)

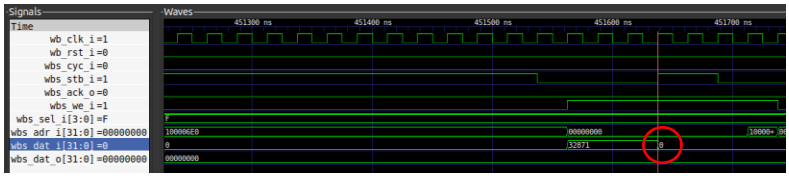
```
MEMORY {
        vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
        dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
        dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
        flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
        mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000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

      That’s the reason why when we are doing something involving the array calculation, we would access this address to get the data or store the data.

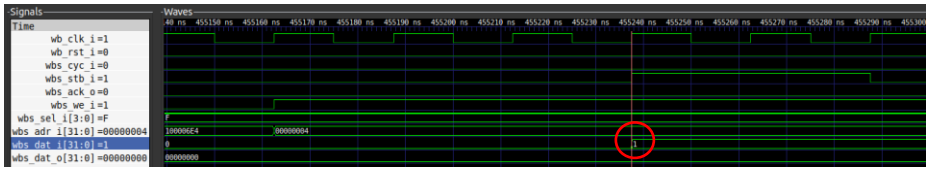
```
for(k = 0;k<SIZE_M;k++)
        sum += A[(i*SIZE_M) + k] * B[(k*SIZE_M) + j];
```

```
/*----- Matrix Multiplication -----*/
int A[SIZE_M*SIZE_M] = {0, 1, 2, 3,
                        0, 1, 2, 3,
                        0, 1, 2, 3,
                        0, 1, 2, 3,
                       };
int B[SIZE_M*SIZE_M] = {1, 2, 3, 4,
                        5, 6, 7, 8,
                        9, 10, 11, 12,
                       13, 14, 15, 16,
                       };
int result[SIZE_M*SIZE_M];
```

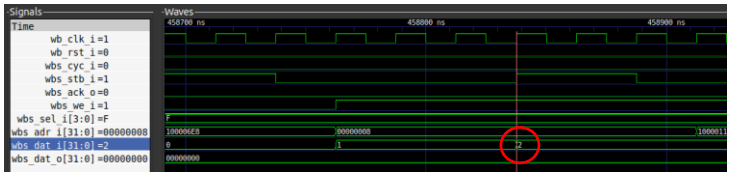
- Data1 = 1



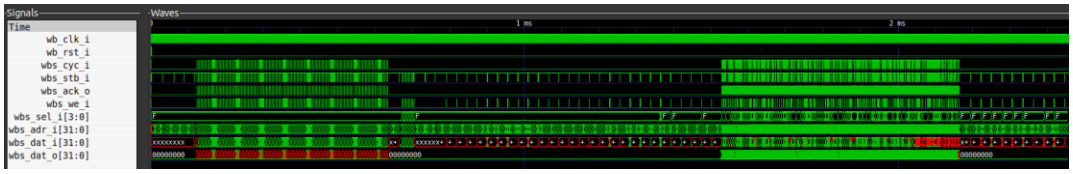
- Data2 = 2



- Data3 = 3



- Quick Sort
  - Wave view (whole)



  - .header file

```
/*----- Quick Sort -----*/
int Q[SIZE_Q] = {893, 40, 3233, 4267, 2669, 2541, 9073, 6023, 5681, 4622};
```

Why the initialization of the array is written to the address 0x00000000?

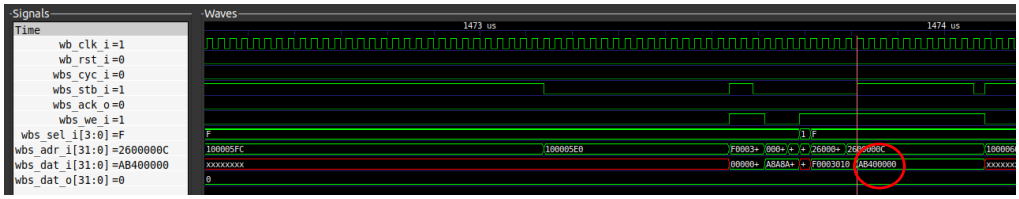
Then we found out that the array used in the firmware is stored in the memory in “dff,” which address is 0x00000000 (from firmware/section.ids)

```
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
}
```

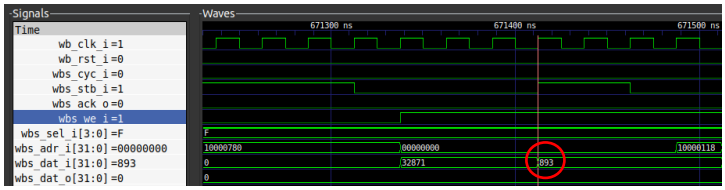
That’s the reason why when we are doing something involving the array calculation, we would access this address to get the data or store the data.

  - To check the data written

- Start mark = AB40

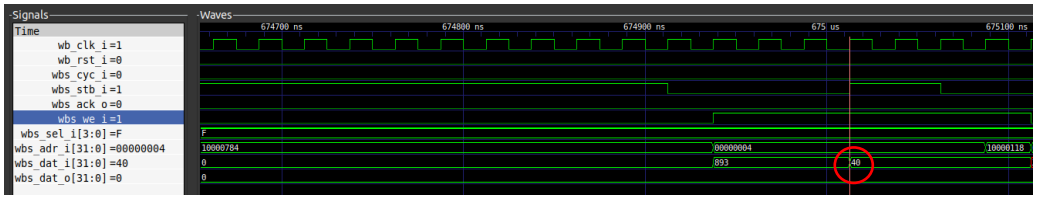


- Data1 = 893

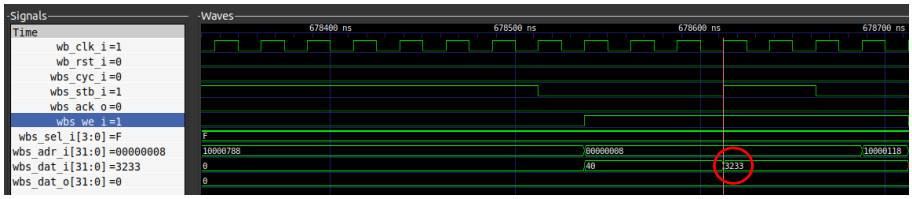


The question we have is WHY the wbs_addr_i = 0x00000000

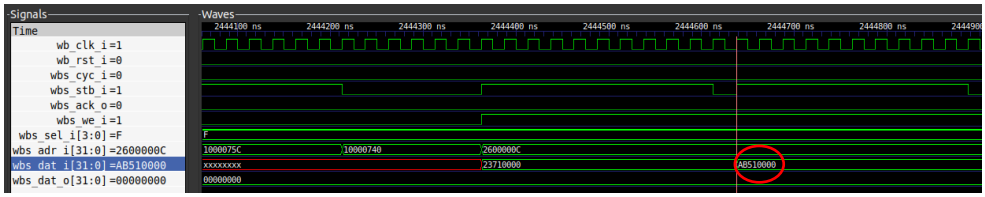
- Data2 = 40



- Data3 = 3233



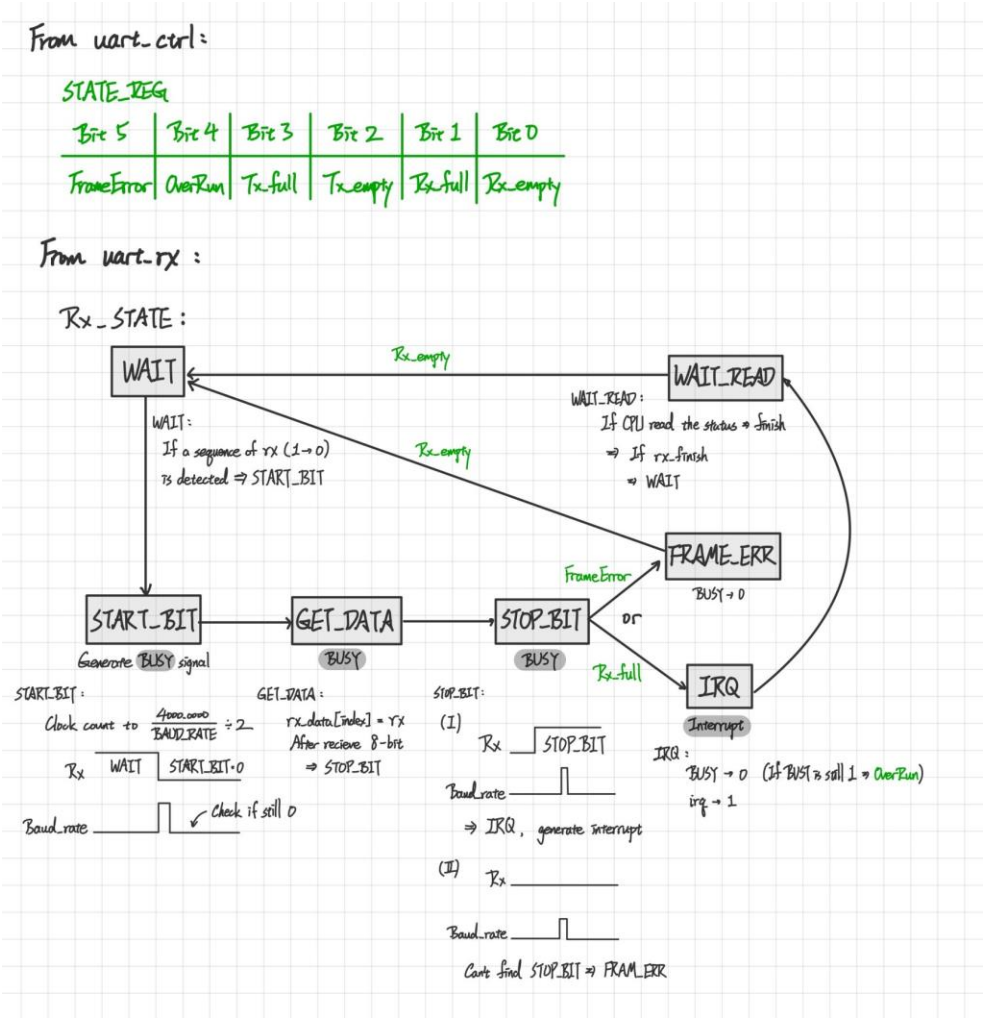
- End mark = AB51



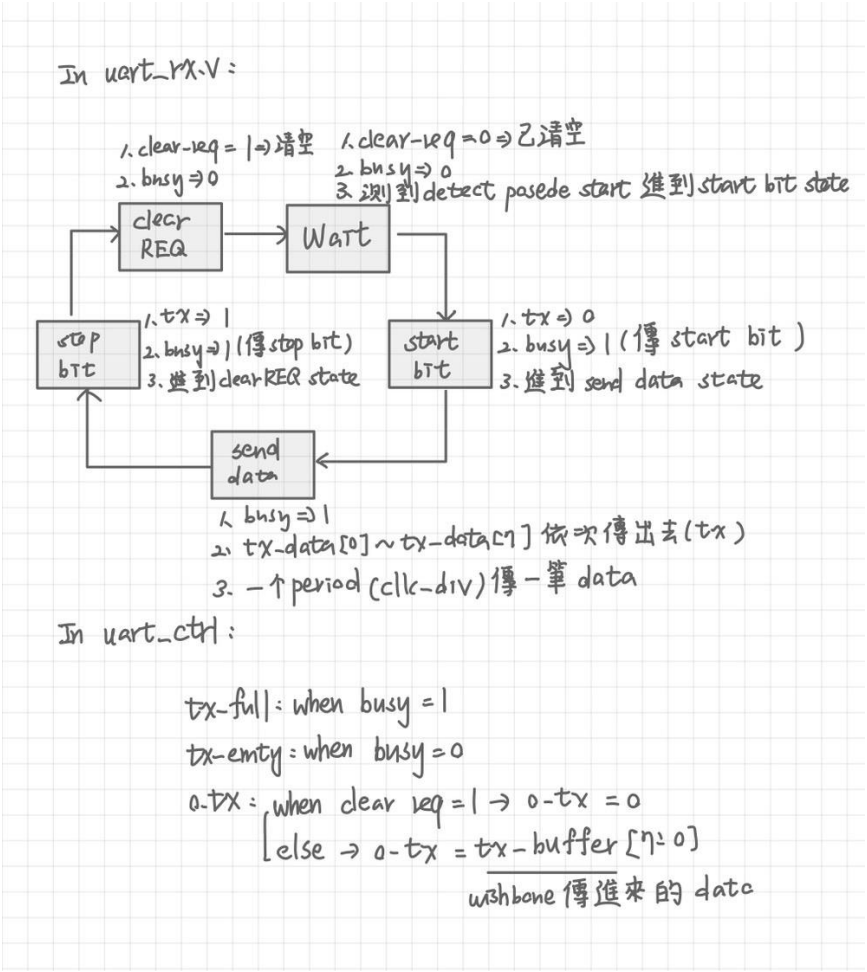
- FIR

  We'll skip this part since this is identical to lab4-1
- ISR – UART/rx & tx
  - UART structure

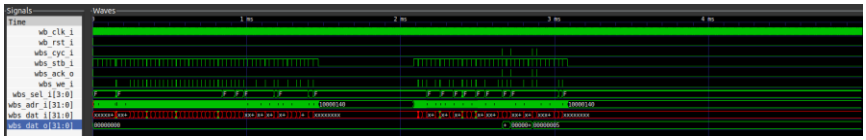
- uart_rx


From uart_ctrl:
STATE_REG
Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0
FrameError | OverRun | Tx_full | Tx_empty | Rx_full | Rx_empty
From uart_rx :
Rx_STATE :
WAIT
WAIT_READ
Rx_empty
WAIT:
If a sequence of rx (1→0)
is detected ⇒ START_BIT
WAIT_READ:
If CPU read the status ⇒ finish
⇒ if rx_finish
⇒ WAIT
FRAME_ERR
BUSY→0
FrameError
START_BIT
GET_DATA
STOP_BIT
or
Generate BUSY signal
BUSY
BUSY
Rx_full
IRQ
Interrupt
START_BIT:
Clock count to 4000_0000 / BAUD_RATE ÷2
Rx WAIT START_BIT=0
Baud_rate
Check if still 0
GET_DATA:
rx_data[index] = rx
After recieve 8-bit
⇒ STOP_BIT
STOP_BIT:
(I)
Rx STOP_BIT
Baud_rate
⇒ IRQ, generate interrupt
(II)
Rx
Baud_rate
Can't find STOP_BIT ⇒ FRAM_ERR
IRQ:
BUSY→0 (if BUSY is still 1 ⇒ OverRun)
irq→1

- uart_tx



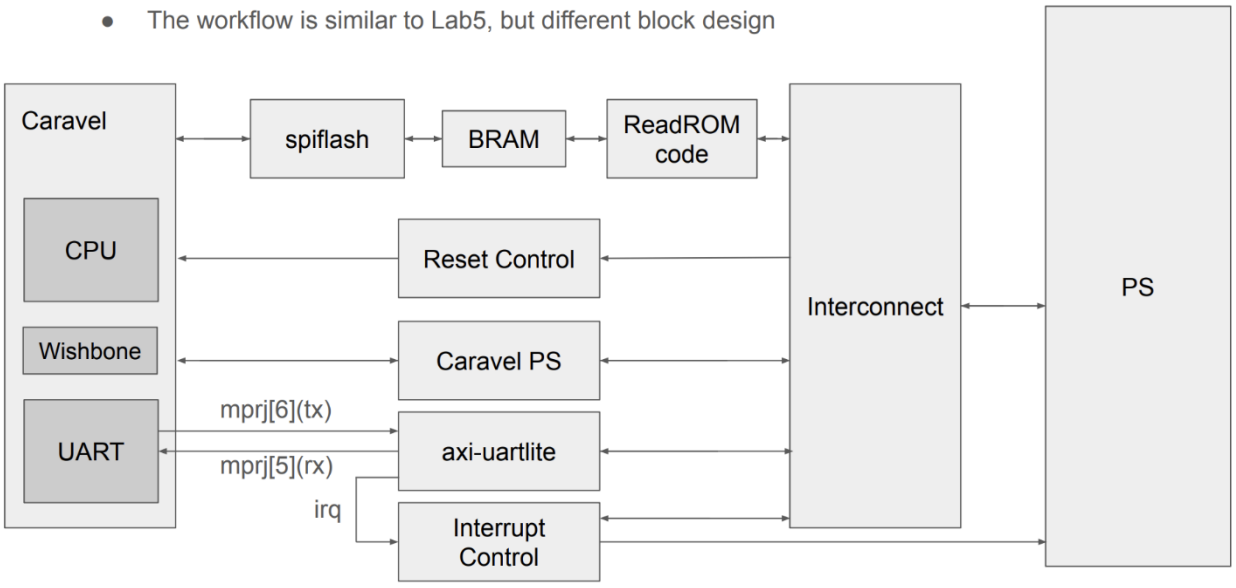
- Wave view (overall)

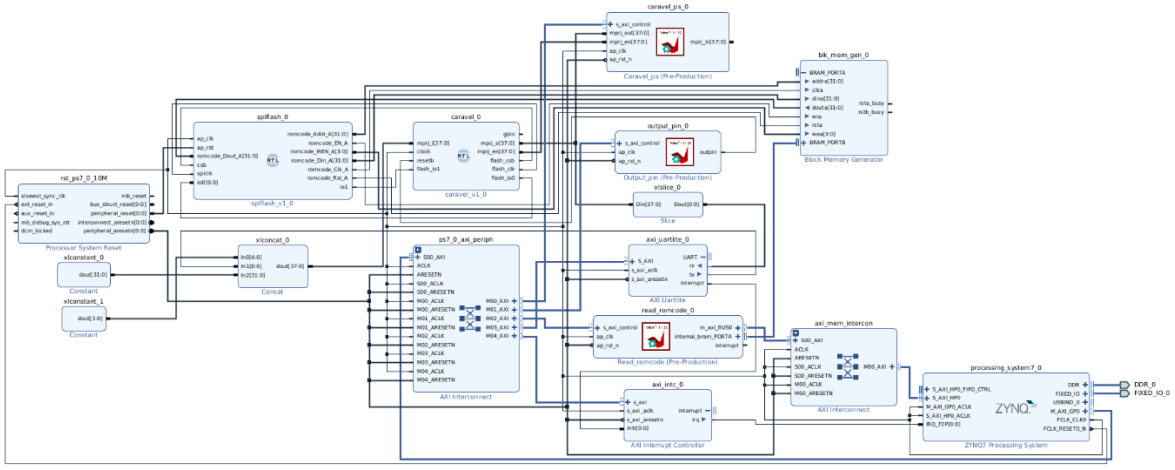


- ./run_sim

```
ubuntu@ubuntu2004:~/caravel-soc_fpga-lab/lab-wlos_baseline/testbench/uart$ ./run_sim
Reading uart.hex
uart.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile uart.vcd opened for output.
LA Test 1 started
tx data bit index 0: 1
tx data bit index 1: 0
tx data bit index 2: 1
tx data bit index 3: 1
tx data bit index 4: 1
tx data bit index 5: 1
tx data bit index 6: 0
tx data bit index 7: 0
tx complete 2
rx data bit index 0: 1
rx data bit index 1: 0
rx data bit index 2: 1
rx data bit index 3: 1
rx data bit index 4: 1
rx data bit index 5: 1
rx data bit index 6: 0
rx data bit index 7: 0
recevied word  61
```

- Block design





- Timing report
  - Timing summery

```
| Design Timing Summary
| ---------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints     WPWS(ns)     TPWS(ns)  TPWS
Failing Endpoints  TPWS Total Endpoints
    -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------     --------     --------  ----------
    -----------  -------------------
     11.308        0.000                      0                12729       -0.584       -1.160                      2                12729       11.250        0.000
0                5267
```

  - Clock summery

```
| Clock Summary
| -------------

Clock        Waveform(ns)       Period(ns)      Frequency(MHz)
-----        ------------       ----------      --------------
clk_fpga_0   {0.000 12.500}     25.000          40.000
```

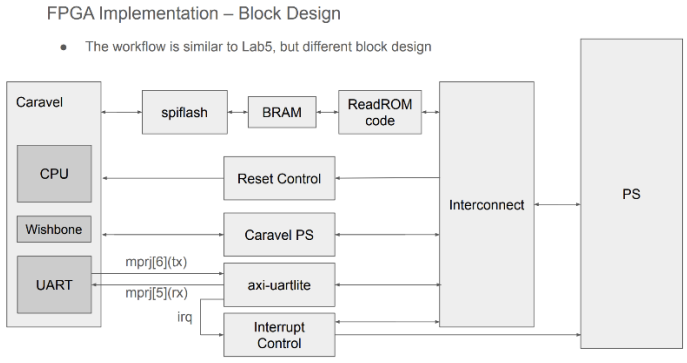
- MAX delay path

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             11.308ns  (required time - arrival time)
  Source:                 design_1_i/processing_system7_0/inst/PS7_i/FCLKCLK[0]
                            (clock source 'clk_fpga_0'  {rise@0.000ns fall@12.500ns period=25.000ns})
  Destination:            design_1_i/caravel_ps_0/inst/control_s_axi_U/int_ps_mprj_out_reg[15]/D
                            (rising edge-triggered cell FDRE clocked by clk_fpga_0  {rise@0.000ns fall@12.500ns period=25.000ns})
  Path Group:             clk_fpga_0
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            12.500ns  (clk_fpga_0 rise@25.000ns - clk_fpga_0 fall@12.500ns)
  Data Path Delay:        3.491ns  (logic 0.225ns (6.445%)  route 3.266ns (93.555%))
  Logic Levels:           2  (BUFG=1 LUT6=1)
  Clock Path Skew:        2.645ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.645ns = ( 27.645 - 25.000 )
    Source Clock Delay      (SCD):    0.000ns = ( 12.500 - 12.500 )
    Clock Pessimism Removal (CPR):    0.000ns
  Clock Uncertainty:      0.377ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.750ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

- Min delay path

```
Min Delay Paths
--------------------------------------------------------------------------------------
Slack (VIOLATED) :        -0.584ns  (arrival time - required time)
  Source:                 design_1_i/processing_system7_0/inst/PS7_i/FCLKCLK[0]
                            (clock source 'clk_fpga_0'  {rise@0.000ns fall@12.500ns period=25.000ns})
  Destination:            design_1_i/caravel_ps_0/inst/control_s_axi_U/int_ps_mprj_out_reg[14]/D
                            (rising edge-triggered cell FDRE clocked by clk_fpga_0  {rise@0.000ns fall@12.500ns period=25.000ns})
  Path Group:             clk_fpga_0
  Path Type:              Hold (Min at Fast Process Corner)
  Requirement:            0.000ns  (clk_fpga_0 rise@0.000ns - clk_fpga_0 rise@0.000ns)
  Data Path Delay:        1.097ns  (logic 0.071ns (6.471%)  route 1.026ns (93.529%))
  Logic Levels:           2  (BUFG=1 LUT6=1)
  Clock Path Skew:        1.213ns (DCD - SCD - CPR)
    Destination Clock Delay (DCD):    1.213ns
    Source Clock Delay      (SCD):    0.000ns
    Clock Pessimism Removal (CPR):    -0.000ns
  Clock Uncertainty:      0.377ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.750ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

- Resource report after synthesis



- Axi_uartlite

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	104	0	0	53200	0.20
LUT as Logic	86	0	0	53200	0.16
LUT as Memory	18	0	0	17400	0.10
LUT as Distributed RAM	0	0			
LUT as Shift Register	18	0			
Slice Registers	113	0	0	106400	0.11
Register as Flip Flop	113	0	0	106400	0.11
Register as Latch	0	0	0	106400	0.00
F7 Muxes	1	0	0	26600	<0.01
F8 Muxes	0	0	0	13300	0.00

- blk_mem_gen

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	10	0	0	53200	0.02
LUT as Logic	8	0	0	53200	0.02
LUT as Memory	2	0	0	17400	0.01
LUT as Distributed RAM	0	0			
LUT as Shift Register	2	0			
Slice Registers	12	0	0	106400	0.01
Register as Flip Flop	12	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

- blk_mem_gen_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

- Caravel_PS

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	119	0	0	53200	0.22
LUT as Logic	119	0	0	53200	0.22
LUT as Memory	0	0	0	17400	0.00
Slice Registers	158	0	0	106400	0.15
Register as Flip Flop	158	0	0	106400	0.15
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

- Caravel_SOC

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	3751	0	0	53200	7.05
LUT as Logic	3697	0	0	53200	6.95
LUT as Memory	54	0	0	17400	0.31
LUT as Distributed RAM	16	0			
LUT as Shift Register	38	0			
Slice Registers	3958	0	0	106400	3.72
Register as Flip Flop	3883	0	0	106400	3.65
Register as Latch	75	0	0	106400	0.07
F7 Muxes	169	0	0	26600	0.64
F8 Muxes	47	0	0	13300	0.35

- Caravel_SOC_MEMORY

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	4	0	0	140	2.86
RAMB36/FIFO*	1	0	0	140	0.71
RAMB36E1 only	1				
RAMB18	6	0	0	280	2.14
RAMB18E1 only	6				

- outputPIN

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	10	0	0	53200	0.02
LUT as Logic	10	0	0	53200	0.02
LUT as Memory	0	0	0	17400	0.00
Slice Registers	12	0	0	106400	0.01
Register as Flip Flop	12	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

- Read_ROMcode

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	739	0	0	53200	1.39
LUT as Logic	664	0	0	53200	1.25
LUT as Memory	75	0	0	17400	0.43
LUT as Distributed RAM	0	0			
LUT as Shift Register	75	0			
Slice Registers	1100	0	0	106400	1.03
Register as Flip Flop	1100	0	0	106400	1.03
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

- Read_ROMcode_MEMORY

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	1	0	0	140	0.71
RAMB36/FIFO*	1	0	0	140	0.71
RAMB36E1 only	1				
RAMB18	0	0	0	280	0.00

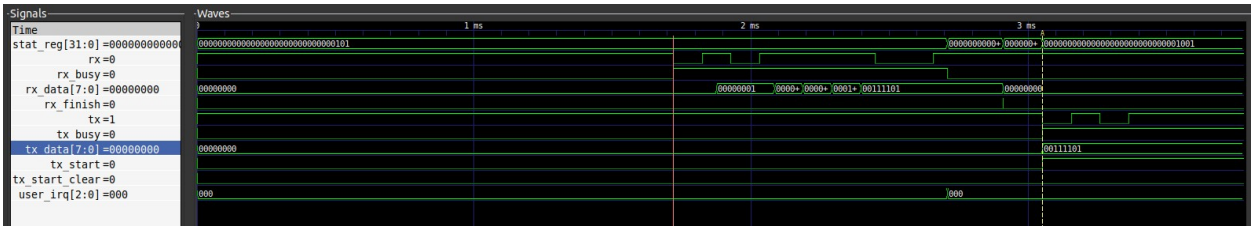
- Reset_ctrl

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	19	0	0	53200	0.04
LUT as Logic	18	0	0	53200	0.03
LUT as Memory	1	0	0	17400	<0.01
LUT as Distributed RAM	0	0			
LUT as Shift Register	1	0			
Slice Registers	40	0	0	106400	0.04
Register as Flip Flop	40	0	0	106400	0.04
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

- Spiflash

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	44	0	0	53200	0.08
LUT as Logic	44	0	0	53200	0.08
LUT as Memory	0	0	0	17400	0.00
Slice Registers	63	0	0	106400	0.06
Register as Flip Flop	63	0	0	106400	0.06
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

- Latency for a character loop back using UART



The latency is time difference between the orange line and the yellow line, which is approximately 1.4ms.

- How do you verify your answer from notebook
  - **run_sim** with all 4 functions combined

```
ubuntu@ubuntu2004:~/caravel-soc_fpga-lab/lab-wlos_baseline/testbench/combined$ ./run_sim
Reading combined.hex
combined.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile combined.vcd opened for output.
FIR Test started
Parallelly run UART
tx data bit index 0: 1
tx data bit index 1: 0
tx data bit index 2: 1
tx data bit index 3: 1
tx data bit index 4: 1
tx data bit index 5: 1
tx data bit index 6: 0
tx data bit index 7: 0
tx complete 2
rx data bit index 0: 1
rx data bit index 1: 0
rx data bit index 2: 1
rx data bit index 3: 1
rx data bit index 4: 1
FIR Test finished
MM Test started
rx data bit index 5: 1
rx data bit index 6: 0
rx data bit index 7: 0
recevied word  61
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x003e
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0044
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x004a
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0050
MM Test passed
QS Test started
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0028
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x037d
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x09ed
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0a6d
QS Test passed
====================================
All FIR/MM/QS and UART are finished!!
====================================
ubuntu@ubuntu2004:~/caravel-soc_fpga-lab/lab-wlos_baseline/testbench/combined$
```

  - FPGA verification on Jupyter notebook

```
In [10]: asyncio.run(async_main())

Start Caravel Soc
Waitting for interrupt
--------- FIR test started ---------
-------> FIR test finished <--------
--------- MM Test started ----------
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x003E 0x3e
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0044 0x44
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x004A 0x4a
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0050 0x50
--------> MM Test finished <--------
-------- QS Test started -----------
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 40 0x28
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 93 0x37
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 2541 0x9e
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 2669 0xa6
--------> QS Test finished <--------
====================================
All FIR/MM/QS and UART are finished!!
====================================
hello
main(): uart_rx is cancelled now

In [11]: 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 =  0xab810040
0x20 =  0x0
0x34 =  0x20
0x38 =  0x3f
```

- Discussion
  After successful run_sim, we faced some difficulties running the implementation on FPGA on Jupyter notebook. Later, we found out that the reason that the python couldn't receive the data send by firmware is that the python code is implementing in a relatively slow speed, which means the firmware has already finished before python sampling the signals for verification.
  The solution we choose is to "let the firmware slower! " Before any signal we want to sample for verification, we add a while loop (30000 counts)before it to wait for the python code.
  Below is one of the signal we want to test:

```
// WAIT = 30000 for PYNQ, 1 for run_sim
#define WAIT 1
```

```
while (count < WAIT) {
        count++;
        reg_mprj_datal = 0xAB510000;
}
count = 0;
```

  Also, the riscv from the firmware code also needs a small modification. Since the original -o compiler results in too large .hex file. We changed the compiler to -o1 to have a smaller .hex file.

- Suggestion for improving latency for UART loop back
  - **Increase Baud Rate:**
    One of the simplest ways to improve UART communication speed is to increase the baud rate. Higher baud rates allow for faster data transmission between devices. Ensure that both the transmitter and receiver can support the selected baud rate.
  - **FIFO Buffers:**
    Use hardware FIFO buffers if available. These buffers can store multiple bytes of data, reducing the time the CPU spends handling individual characters. This can significantly improve throughput and reduce latency.
  - **DMA (Direct Memory Access):**
    If your microcontroller or platform supports it, use DMA to transfer data between the UART and memory without CPU intervention. DMA can reduce the latency introduced by CPU involvement in data transfer.
  - **Interrupts:**
    Utilize UART interrupts to handle data reception and transmission. This allows the CPU to perform other tasks while waiting for UART events, reducing overall latency.