SOC LAB#6 Test (lab-wlos_baseline)Report

Group no: 4

Members:

M11107410 羅善寬

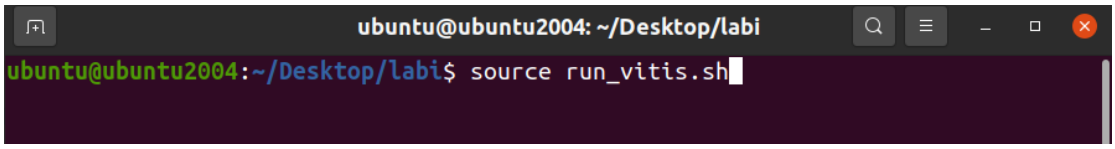
M11107004 曹榮恩

M11107409 陳昱碩

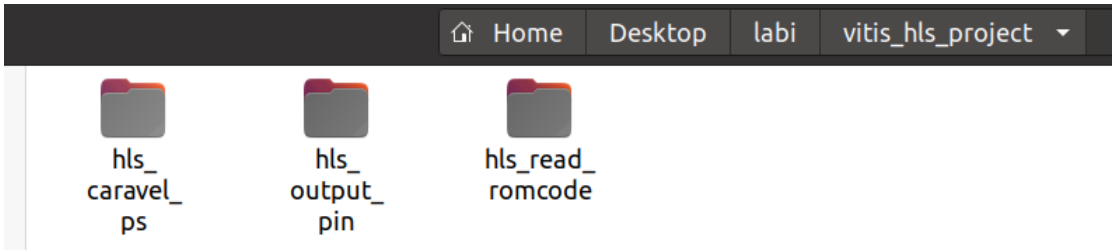
1. Run_vitis

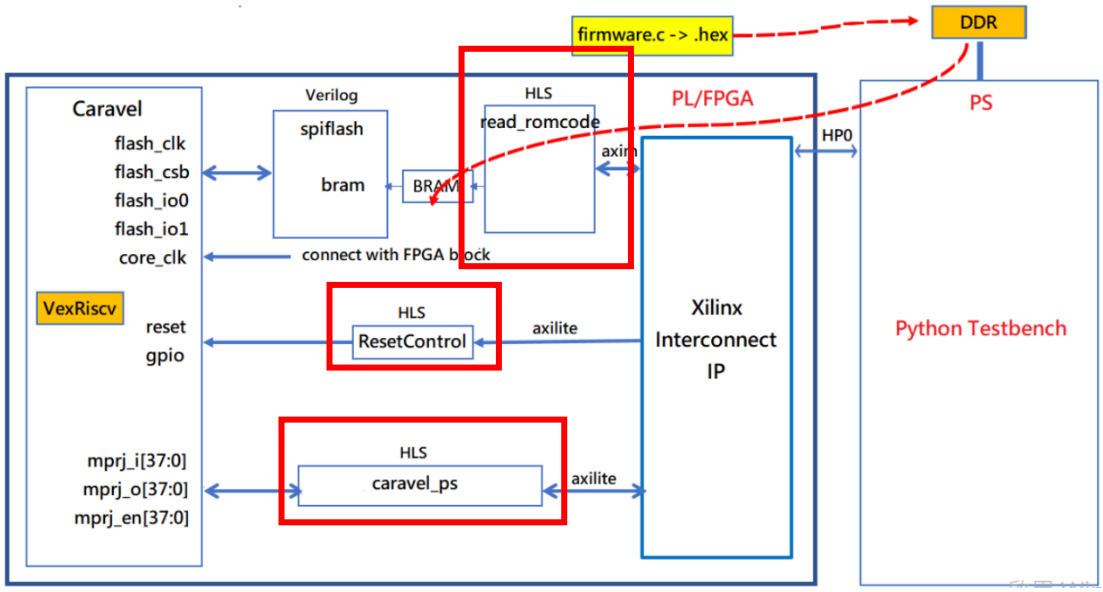
1.1 Vitis HLS(生成 IP)

這步驟已由 lab5 完成。



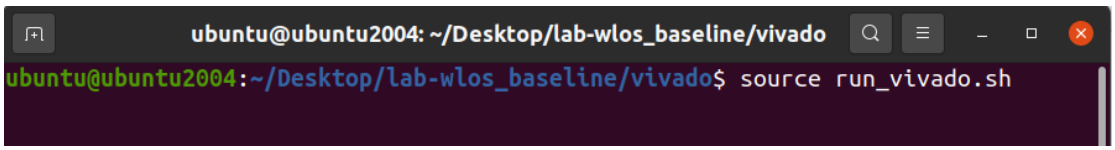
以下三個硬體由 vitis 產生 ip。





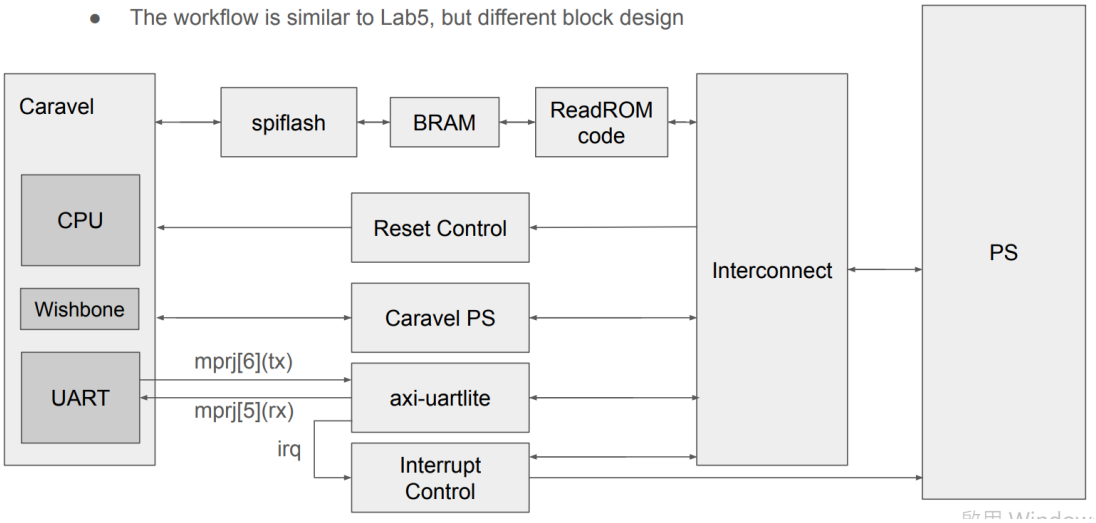
2. Run_vivado

2.1 Vivado (IP、SOC 整合)



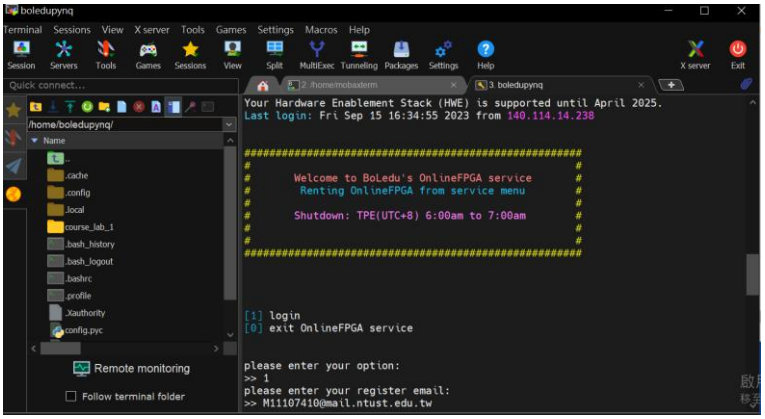
將整個系統整合起來。

ax_uartlite 和 interrupt_control 為直接抓 xilinx 的 ip。

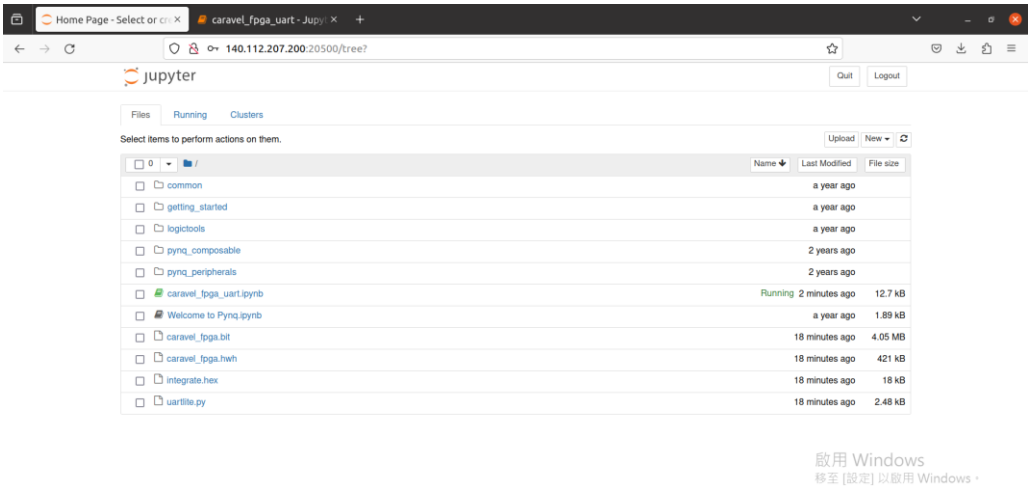


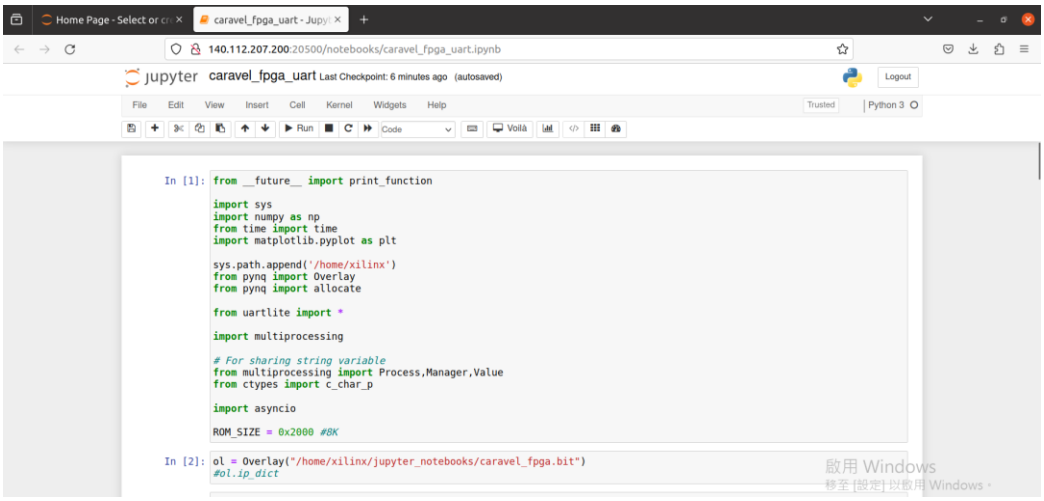
3. Online FPGA

租借一個線上的 FPGA 板子。

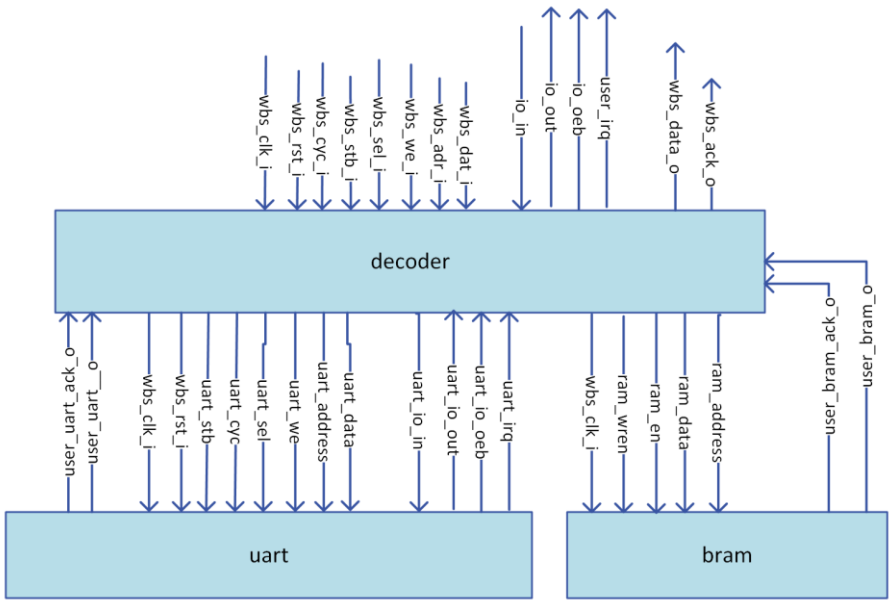


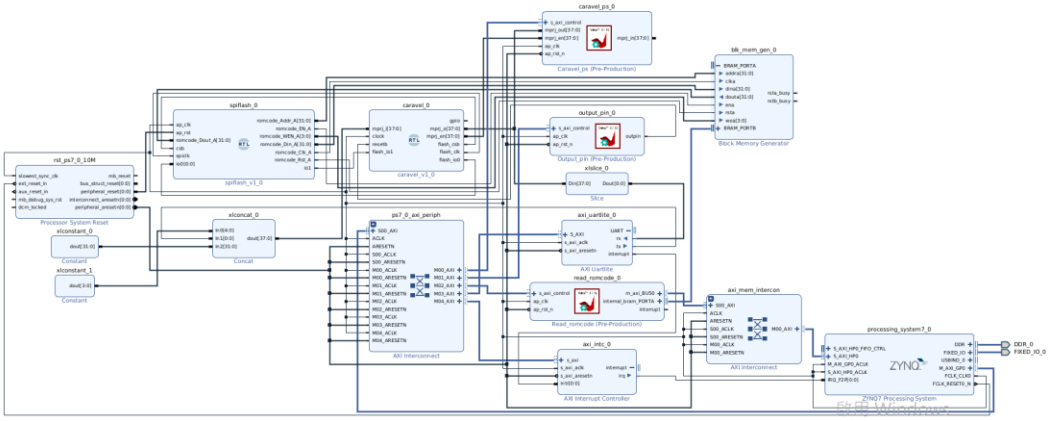
租借板子後在 jupyter 上用 python 驗證。





4. Block design





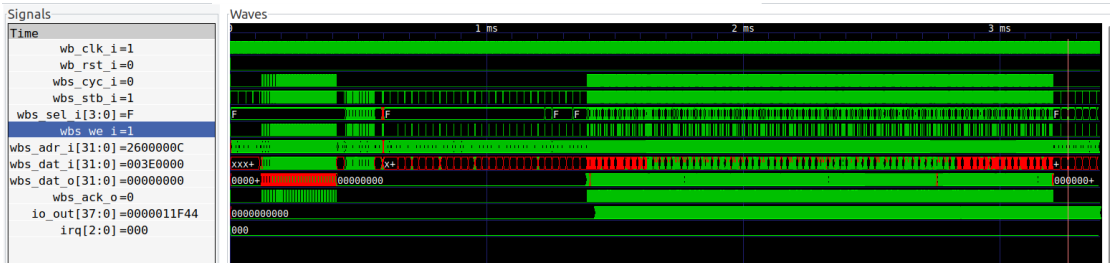
5. How do you verify your answer from notebook

5.1 Simulation on Matrix Multiplication, Quick Sort, FIR and UART separately

- Matrix Multiplication

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} = \begin{pmatrix} 62 & 68 & 74 & 80 \\ 62 & 68 & 74 & 80 \\ 62 & 68 & 74 & 80 \\ 62 & 68 & 74 & 80 \end{pmatrix}$$

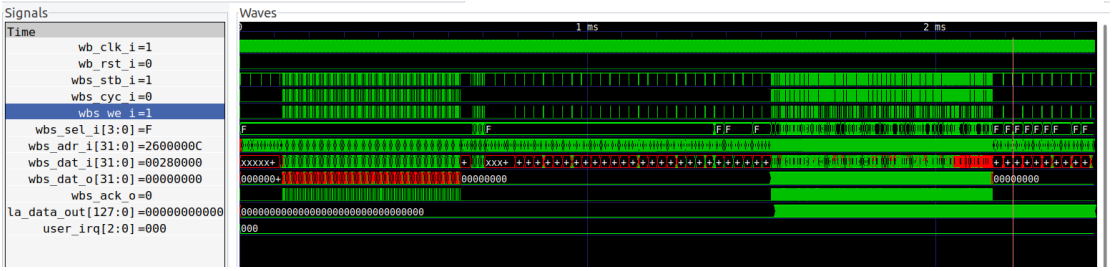
```
ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/counter_la_mm$ source run_clean
ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/counter_la_mm$ source run_sim
Reading counter_la_mm.hex
counter_la_mm.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_mm.vcd opened for output.
LA Test 1 started
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
LA Test 2 passed
```



➢ Quick Sort

{40, 893, 2541, 2669, 3233, 4267, 4622, 5681, 6023, 9073}

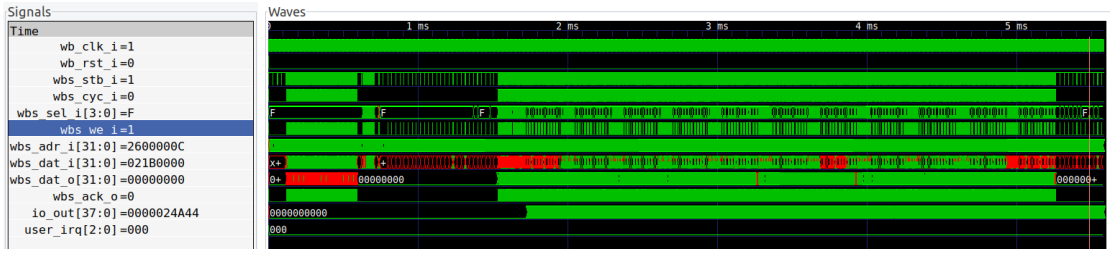
```
ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/counter_la_qs$ source run_clean
ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/counter_la_qs$ source run_sim
Reading counter_la_qs.hex
counter_la_qs.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_qs.vcd opened for output.
LA Test 1 started
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0028
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x037d
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x09ed
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0a6d
LA Test 2 passed
```



➢ FIR

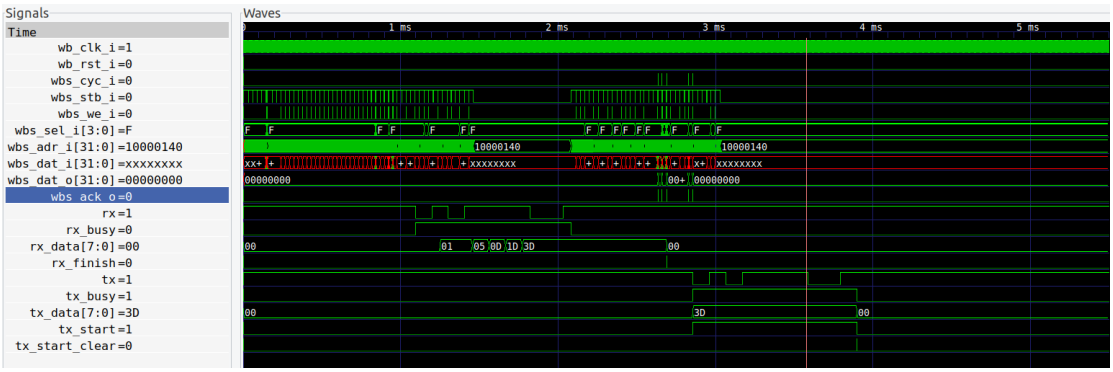
{0, -10, -29, -25, 35, 158, 337, 539, 732, 915, 1098}

```
ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/counter_la_fir$ source run_clean
ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/counter_la_fir$ source run_sim

Reading counter_la_fir.hex
counter_la_fir.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_fir.vcd opened for output.
LA Test 1 started
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x021b
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x02dc
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0393
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x044a
LA Test 2 passed
```



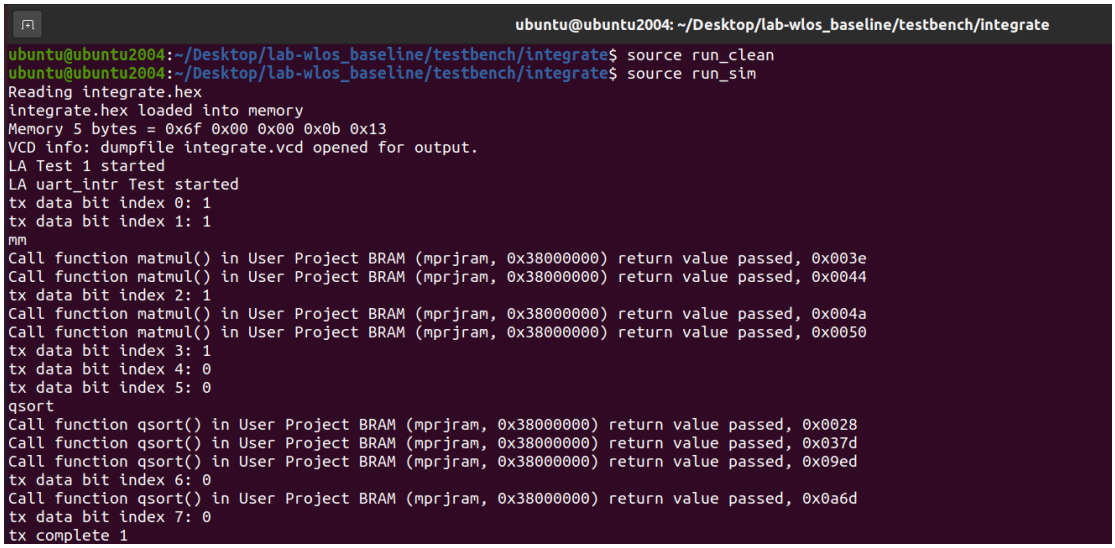
➢ UART

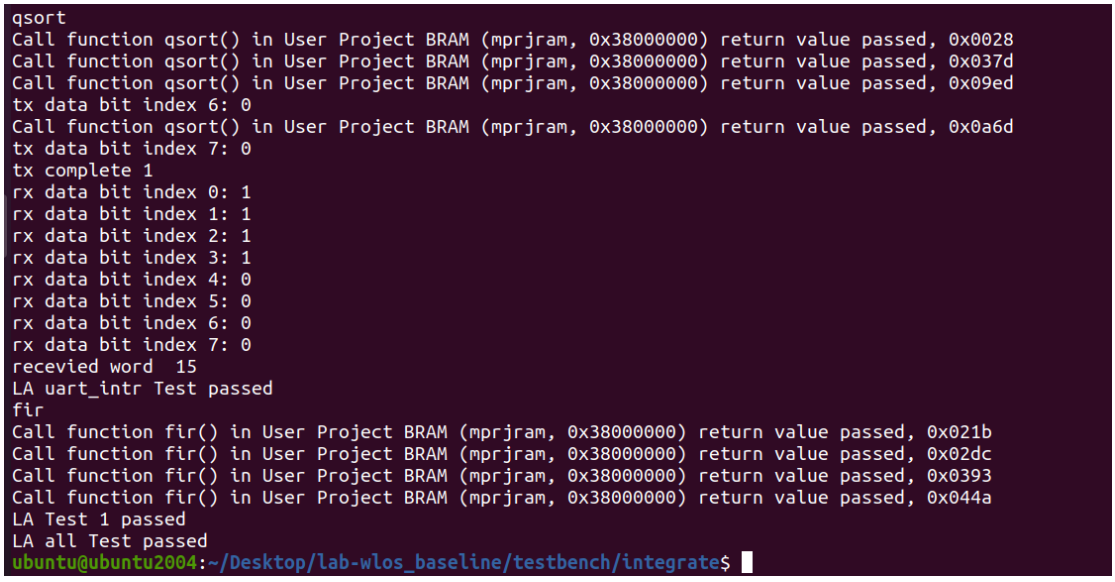
```
ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/uart$ source run_clean
ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/uart$ source run_sim
Reading uart.hex
uart.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile uart.vcd opened for output.
LA Test 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

Monitor: Timeout, Test LA (RTL) Failed
```



5.2 Simulation on Matrix Multiplication, Quick Sort, FIR and UART integratedly

- intergrate





```
async def check():
    #x = hex(ipPS.read(0x1c)[:16])
    while(((ipPS.read(0x1c)) & 0xffff0000) == 0xab410000):
        continue
    print("mm passed")
    while(((ipPS.read(0x1c)) & 0xffff0000) == 0xab420000):
        continue
    print("qs passed")
    while(((ipPS.read(0x1c)) & 0xffff0000) == 0xab430000):
        continue
    print("fir passed")
```

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

```
Start Caravel Soc
Waitting for interrupt
mm passed
qs passed
fir passed
hello
main(): uart_rx is cancelled now
```

6. Timing report/ resource report after synthesis

5.1 resource report

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	5385	0	0	53200	10.12
LUT as Logic	5197	0	0	53200	9.77
LUT as Memory	188	0	0	17400	1.08
LUT as Distributed RAM	18	0			
LUT as Shift Register	170	0			
Slice Registers	6165	0	0	106400	5.79
Register as Flip Flop	6165	0	0	106400	5.79
Register as Latch	0	0	0	106400	0.00
F7 Muxes	170	0	0	26600	0.64
F8 Muxes	47	0	0	13300	0.35

5.2 timing report

```
219 Max Delay Paths
220 --------------------------------------------------------------------------------------
221 Slack (MET) :             11.294ns  (required time - arrival time)
222   Source:                 design_1_i/processing_system7_0/inst/PS7_i/FCLKCLK[0]
223                             (clock source 'clk_fpga_0'  {rise@0.000ns fall@12.500ns period=25.000ns})
224   Destination:            design_1_i/caravel_ps_0/inst/control_s_axi_U/int_ps_mprj_out_reg[14]/D
225                             (rising edge-triggered cell FDRE clocked by clk_fpga_0  {rise@0.000ns fall@12.500ns period=25.000ns})
226   Path Group:             clk_fpga_0
227   Path Type:              Setup (Max at Slow Process Corner)
228   Requirement:            12.500ns  (clk_fpga_0 rise@25.000ns - clk_fpga_0 fall@12.500ns)
229   Data Path Delay:        3.692ns  (logic 0.225ns (6.095%)  route 3.467ns (93.905%))
230   Logic Levels:           2  (BUFG=1 LUT6=1)
231   Clock Path Skew:        2.833ns (DCD - SCD + CPR)
232     Destination Clock Delay (DCD):    2.833ns = ( 27.833 - 25.000 )
233     Source Clock Delay      (SCD):    0.000ns = ( 12.500 - 12.500 )
234     Clock Pessimism Removal (CPR):    0.000ns
235   Clock Uncertainty:      0.377ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
236     Total System Jitter     (TSJ):    0.071ns
237     Total Input Jitter      (TIJ):    0.750ns
238     Discrete Jitter          (DJ):    0.000ns
239     Phase Error              (PE):    0.000ns
240
```

```
1338 Max Delay Paths
1339 --------------------------------------------------------------------------------------
1340 Slack (MET) :             17.534ns  (required time - arrival time)
1341   Source:                 design_1_i/output_pin_0/inst/control_s_axi_U/int_outpin_ctrl_reg[0]/C
1342                             (rising edge-triggered cell FDRE clocked by clk_fpga_0  {rise@0.000ns fall@12.500ns period=25.000ns})
1343   Destination:            design_1_i/caravel_0/inst/housekeeping/pad_count_1_reg[0]/CLR
1344                             (recovery check against rising-edge clock clk_fpga_0  {rise@0.000ns fall@12.500ns period=25.000ns})
1345   Path Group:             **async_default**
1346   Path Type:              Recovery (Max at Slow Process Corner)
1347   Requirement:            25.000ns  (clk_fpga_0 rise@25.000ns - clk_fpga_0 rise@0.000ns)
1348   Data Path Delay:        6.678ns  (logic 0.580ns (8.685%)  route 6.098ns (91.315%))
1349   Logic Levels:           1  (LUT1=1)
1350   Clock Path Skew:        -0.007ns (DCD - SCD + CPR)
1351     Destination Clock Delay (DCD):    2.802ns = ( 27.802 - 25.000 )
1352     Source Clock Delay      (SCD):    2.938ns
1353     Clock Pessimism Removal (CPR):    0.129ns
1354   Clock Uncertainty:      0.377ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
1355     Total System Jitter     (TSJ):    0.071ns
1356     Total Input Jitter      (TIJ):    0.750ns
1357     Discrete Jitter          (DJ):    0.000ns
1358     Phase Error              (PE):    0.000ns
1359
```

7. Latency for a character loop back using UART

In [12]:
```
import time
async def uart_rxtx():
    # Reset FIFOs, enable interrupts
    ipUart.write(CTRL_REG, 1<<RST_TX | 1<<RST_RX | 1<<INTR_EN)
    print("Waitting for interrupt")
    tx_str = "hello\n"
    ipUart.write(TX_FIFO, ord(tx_str[0]))
    start = time.time()
    i = 1
    while(True):
        await intUart.wait()
        buf = ""
        # Read FIFO until valid bit is clear
        while ((ipUart.read(STAT_REG) & (1<<RX_VALID))):
            buf += chr(ipUart.read(RX_FIFO))
            end = time.time()
            print("latency time:",(end - start))
            if i<len(tx_str):
                ipUart.write(TX_FIFO, ord(tx_str[i]))
                i=i+1
        #print(buf, end='')

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

# Python 3.5+
#tasks = [ # Create a task list
```

In [17]:
```
asyncio.run(async_main())
```

```
Start Caravel Soc
Waitting for interrupt
latency time: 0.003509998321533203
latency time: 0.0084381103515625
latency time: 0.01298069953918457
latency time: 0.017049074172973633
latency time: 0.02138662338256836
latency time: 0.025643587112426758
main(): uart_rx is cancelled now
```

	latency
start tx 到接收到 h	3.5 - 0 = 3.5 ms
接收 h 之後到接收到 e	8.4 – 3.5 = 4.9 ms
接收 e 之後到接收到 l	13.0 – 8.4 = 4.6 ms
接收 l 之後到接收到 l	17.0 – 13.0 = 4.0 ms
接收 l 之後到接收到 o	21.4 – 17.0 = 4.4 ms

8. Screenshot of Execution result on workload

In [1]:
```
from __future__ import print_function

import sys
import numpy as np
from time import time
import matplotlib.pyplot as plt

sys.path.append('/home/xilinx')
from pynq import Overlay
from pynq import allocate

from uartlite import *

import multiprocessing

# For sharing string variable
from multiprocessing import Process,Manager,Value
from ctypes import c_char_p

import asyncio

ROM_SIZE = 0x2000 #8K
```

In [2]:
```
ol = Overlay("/home/xilinx/jupyter_notebooks/caravel_fpga.bit")
#ol.ip_dict
```

In [3]:
```
ipOUTPIN = ol.output_pin_0
ipPS = ol.caravel_ps_0
ipReadROMCODE = ol.read_romcode_0
ipUart = ol.axi_uartlite_0
```

In [4]:
```
ol.interrupt_pins
```

Out[4]:
```
{'axi_intc_0/intr': {'controller': 'axi_intc_0',
  'index': 0,
  'fullpath': 'axi_intc_0/intr'},
 'axi_uartlite_0/interrupt': {'controller': 'axi_intc_0',
  'index': 0,
  'fullpath': 'axi_uartlite_0/interrupt'}}
```

In [5]:
```
# See what interrupts are in the system
#ol.interrupt_pins

# Each IP instances has a _interrupts dictionary which lists the names of the interrupts
#ipUart._interrupts

# The interrupts object can then be accessed by its name
# The Interrupt class provides a single function wait
# which is an asyncio coroutine that returns when the interrupt is signalled.
intUart = ipUart.interrupt
```

open “.hex”。

將“.hex”寫進去 bram。

“.hex”裡以”@”開頭。

In [6]:
```
# Create np with 8K/4 (4 bytes per index) size and be initiled to 0
rom_size_final = 0

npROM = np.zeros(ROM_SIZE >> 2, dtype=np.uint32)
npROM_index = 0
npROM_offset = 0
fiROM = open("integrate.hex", "r+")
#fiROM = open("counter_wb.hex", "r+")

for line in fiROM:
    # offset header
    if line.startswith('@'):
        # Ignore first char @
        npROM_offset = int(line[1:].strip(b'\x00'.decode()), base = 16)
        npROM_offset = npROM_offset >> 2 # 4byte per offset
        #print (npROM_offset)
        npROM_index = 0
        continue
    #print (line)

    # We suppose the data must be 32bit alignment
    buffer = 0
    bytecount = 0
    for line_byte in line.strip(b'\x00'.decode()).split():
        buffer += int(line_byte, base = 16) << (8 * bytecount)
        bytecount += 1
        # Collect 4 bytes, write to npROM
        # Collect 4 bytes, write to npROM
        if(bytecount == 4):
            npROM[npROM_offset + npROM_index] = buffer
            # Clear buffer and bytecount
            buffer = 0
            bytecount = 0
            npROM_index += 1
            #print (npROM_index)
            continue
    # Fill rest data if not alignment 4 bytes
    if (bytecount != 0):
        npROM[npROM_offset + npROM_index] = buffer
        npROM_index += 1

fiROM.close()

rom_size_final = npROM_offset + npROM_index
#print (rom_size_final)

#for data in npROM:
#    print (hex(data))
```

In [7]:
```
# Allocate dram buffer will assign physical address to ip ipReadROMCODE

#rom_buffer = allocate(shape=(ROM_SIZE >> 2,), dtype=np.uint32)
rom_buffer = allocate(shape=(rom_size_final,), dtype=np.uint32)

# Initial it by npROM
#for index in range (ROM_SIZE >> 2):
for index in range (rom_size_final):
    rom_buffer[index] = npROM[index]

#for index in range (ROM_SIZE >> 2):
#    print ("0x{0:08x}".format(rom_buffer[index]))

# Program physical address for the romcode base address


# 0x00 : Control signals
#        bit 0  - ap_start (Read/Write/COH)
#        bit 1  - ap_done (Read/COR)
#        bit 2  - ap_idle (Read)
#        bit 3  - ap_ready (Read)
#        bit 7  - auto_restart (Read/Write)
#        others - reserved
# 0x10 : Data signal of romcode
#        bit 31~0 - romcode[31:0] (Read/Write)
```

```
# 0x14 : Data signal of romcode
#        bit 31~0 - romcode[63:32] (Read/Write)
# 0x1c : Data signal of length_r
#        bit 31~0 - length_r[31:0] (Read/Write)

ipReadROMCODE.write(0x10, rom_buffer.device_address)
ipReadROMCODE.write(0x1C, rom_size_final)

ipReadROMCODE.write(0x14, 0)

# ipReadROMCODE start to move the data from rom_buffer to bram
ipReadROMCODE.write(0x00, 1) # IP Start
while (ipReadROMCODE.read(0x00) & 0x04) == 0x00: # wait for done
    continue

print("Write to bram done")
```

```
Write to bram done
```

In [8]:
```
# Initialize AXI UART
uart = UartAXI(ipUart.mmio.base_addr)

# Setup AXI UART register
uart.setupCtrlReg()

# Get current UART status
uart.currentStatus()
```

Out[8]:
```
{'RX_VALID': 0,
 'RX_FULL': 0,
 'TX_EMPTY': 1,
 'TX_FULL': 0,
 'IS_INTR': 0,
 'OVERRUN_ERR': 0,
 'FRAME_ERR': 0,
 'PARITY_ERR': 0}
```

給 reset 為 1 的信號，caravel 會重新執行 firmware (caravel_start)。

In [9]:
```
import time
async def uart_rxtx():
    # Reset FIFOs, enable interrupts
    ipUart.write(CTRL_REG, 1<<RST_TX | 1<<RST_RX | 1<<INTR_EN)
    print("Waitting for interrupt")
    tx_str = "hello\n"
    ipUart.write(TX_FIFO, ord(tx_str[0]))
    start = time.time()
    i = 1
    while(True):
        await intUart.wait()
        buf = ""
        # Read FIFO until valid bit is clear
        while ((ipUart.read(STAT_REG) & (1<<RX_VALID))):
            buf += chr(ipUart.read(RX_FIFO))
            end = time.time()
            #print("latency time:",(end - start))
            if i<len(tx_str):
                ipUart.write(TX_FIFO, ord(tx_str[i]))
                i=i+1
        print(buf, end='')

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

```
async def check():
    #x = hex(ipPS.read(0x1c)[:16])
    while(((ipPS.read(0x1c)) & 0xffff0000) == 0xab410000):
        continue
    print("mm passed")
    while(((ipPS.read(0x1c)) & 0xffff0000) == 0xab420000):
        continue
    print("qs passed")
    while(((ipPS.read(0x1c)) & 0xffff0000) == 0xab430000):
        continue
    print("fir passed")
```

```
# Python 3.7+
async def async_main():
    task2 = asyncio.create_task(caravel_start())
    task1 = asyncio.create_task(uart_rxtx())
    task0 = asyncio.create_task(check())
    # Wait for 5 second
    await asyncio.sleep(10)
    task1.cancel()
    try:
        await task1
    except asyncio.CancelledError:
        print('main(): uart_rx is cancelled now')
```

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

Start Caravel Soc
Waitting for interrupt
mm passed
qs passed
fir passed
hello
main(): uart_rx is cancelled now
```

執行後“.hex”，查看 mprj_io 的輸出，0x1c 腳位值為 0xab51。

```
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 =  0xab510040
0x20 =  0x0
0x34 =  0x20
0x38 =  0x3f
```

9. Suggestion for improving latency for UART loop back

要改善 UART loopback 延遲，可以優化通信流程，以縮短資料從傳送設備到接收設備再返回的時間。

- 優化 firmware code
- UART 使用硬體來支援
- 中斷

使用中斷驅動的 UART 通信。使微控制器在等待數據時執行其他任務，減少閒置時間，提高整體系統效能。

- Baud Rate 優化

確保 Baud Rate 設置為傳送和接收設備都可靠支援的最高值。較高的 Baud Rate 能夠實現更快的資料傳輸。

- FIFO 緩衝區

FIFO 緩衝區可在微控制器需要處理數據之前存儲多個字節，從而減少處理時間。

- DMA（直接內存訪問）

使用 DMA 來進行 UART 數據傳輸。DMA 允許數據在不涉及 CPU 的情況下直接在內存和外設之間傳輸，降低 CPU 的負擔，提高延遲性能。

- buffer 和 data flow

重疊數據處理和傳輸。

10. What else do you observe

這次 lab6 所實現的功能為中斷並完成 uart 傳輸。

在軟韌方面，當 caravel 接收到外部中斷(if USER_PROJ_IRQ0_EN)，會將 CSR 中的 mask 更新為 2(10_2)，並且 irq enable。之後會執行 isr.c，也就是執行 uart.c，來 read 使他中斷的 data，之後再寫 data 回去外部。裡面包含 user_uart.h 來定義 read(0x30000000)和 write(0x30000004)的腳位。

在硬體中，當 caravel 接收到外部中斷(if USER_PROJ_IRQ0_EN)時，外部 data 從 mprj_io[5]進來到 uart_rx。當 carvel 執行 isr.c 的 read data 時，經由 uart_ctrl 傳 data 給 carvel。當 carvel 執行 isr.c 的 write data 時，經由 uart_ctrl 傳 data 給 uart_tx，並且再由 mprj_io[6]回傳給外部。

而驗證方面 test bench simulation 是由 tbuart 來對 mprj_io[5]和 mprj_io[6]來做 data 讀寫。中間彼此的 interaction 是由 simulator 來跑。

而使用 python code 來跑驗證結果，中間彼此的 interaction 則由實際的 ip 來完成 data 傳遞溝通。

一開始 .hex 檔是載入在 ps side 的 DDR 裡面，需要將.hex 載入到硬體 bram 裡面，所以要透過 read_romcode 這 ip 來傳遞 data。

其中還有 ps side 要與 caravel 的 gpio、reset 溝通所需的 ResetControl ip，以及 ps side 與 caravel 的 mprj_io 溝通所需的 caravel_ps ip。

執行開始前所需的 reset，我們從 python code 可對 ResetControl ip 設定 reset 信號，caravel 就開始跑，當 firmware 做完之後，可以將 reset release 掉。

當 caravel 跑完之後，透過 mprj_io 經由 caravel_ps ip 送一些信息跟 python code 溝通，來驗證功能是否正確。

這次因為有 uart 和中斷，因此還加上 axi_uartlite 和 interrupt_ctrl。

當 ps side 傳 data 給 caravel，data 經由 interconnect，再經由 axi_uartlite 傳給 caravel。也就是 axi_uartlite_tx 傳給 caravel_uart_rx。反之當 caravel 傳 data 給 ps side，data 經由 axi_uartlite，再經由 interconnect，以及再經由 interrupt_ctrl 傳給 ps side。也就是 caravel_uart_tx 傳給 axi_uartlite_rx。而 interrupt_ctrl 是讓 ps side 知道 caravel 有送 data 過來，可以去讀取。