國立台灣大學電機資訊學院電子工程學研究所

系統晶片設計實驗
Soc Design Laboratory

Lab6 Report

Baseline WLOS

學生： M11202109 蘇柏丞
M11202103 陳泓宇
M11202207 呂彥霖

老師： 賴 瑾

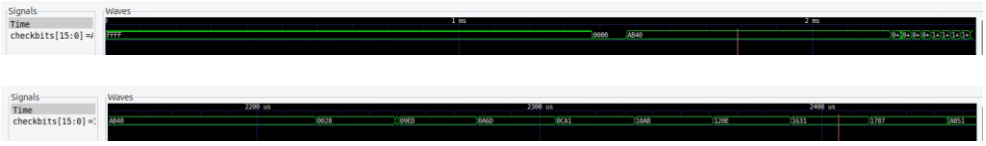
中華民國 112 年 12 月 10 日

A、 Matrix Multiplication

一、 Waveform

B、 Quick Sort

一、 Waveform

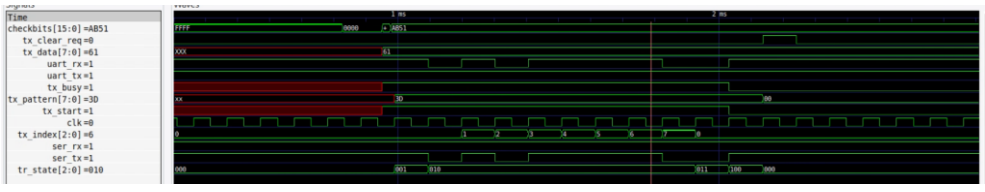


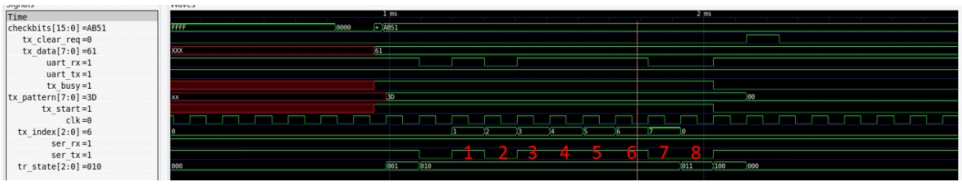
C、 FIR

一、 Waveform

D、 UART

一、 Waveform





二、 How to verify answer from notebook：

在本實驗當中我們首先使用 Vivado 將 UART 功能進行 Synthesis 及 Implementation，檢查完功能與 Timing 後產生 Bitstream。

再來是使用 Online FPGA 實現我們的功能，第一步將 ROM 的大小限制在 8KB，且將 Bitstream 載入我們的 PYNQ 當中，再將所使用到的 IP 匯出。

```
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
```

```
ol = Overlay("design_1.bit")
#ol.ip_dict
```

```
ipOUTPIN = ol.output_pin_0
ipPS = ol.caravel_ps_0
ipReadROMCODE = ol.read_romcode_0
ipUart = ol.axi_uartlite_0
```

```
ol.interrupt_pins
```

```
# See what interrupts are in the system
#ol.interrupt_pins

# Each IP instances has a _interrupts dictionary which lists the na
#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
intUart = ipUart.interrupt
```

第二步，將我們 Fireware 轉成的 HEX 檔打開，同時間分配一塊記憶體位址(DRAM Buffer)給我們用來將 Fireware 進行分割後以每 4byte 為單位轉入 BRAM 當中。

```
# 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("uart.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 =
        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
        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))

```

第三步，將 MMIO 以及 Buffer Size 寫入 PLA，並打入開始訊號進行 Fireware 的搬移。

```
# 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
```

第四步，將 UART 初始化並將 Register 設定好，並確認 UART 狀態。

```
# Initialize AXI UART
uart = UartAXI(ipUart.mmio.base_addr)

# Setup AXI UART register
uart.setupCtrlReg()

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

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

第五步，將 Caravel 的 reset 拉掉，使 Caravel CPU 開始讀取 Fireware 並進行運算，接著打一個 UART 中斷給 Caravel 進行 TX，傳”hello” 進入 Caravel，接著 Caravel 接收到後進行 UART 中斷給 PYNQ 進行 RX，我們確認在 PYNQ 上接收到”hello”。

```
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]))
    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))
            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
#    asyncio.ensure_future(example1()),
#    asyncio.ensure_future(example2()),
#]
# To test this we need to use the asyncio library to schedule our n
# asyncio uses event loops to execute coroutines.
# When python starts it will create a default event loop
# which is what the PYNQ interrupt subsystem uses to handle interru

#loop = asyncio.get_event_loop()
#loop.run_until_complete(asyncio.wait(tasks))

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

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

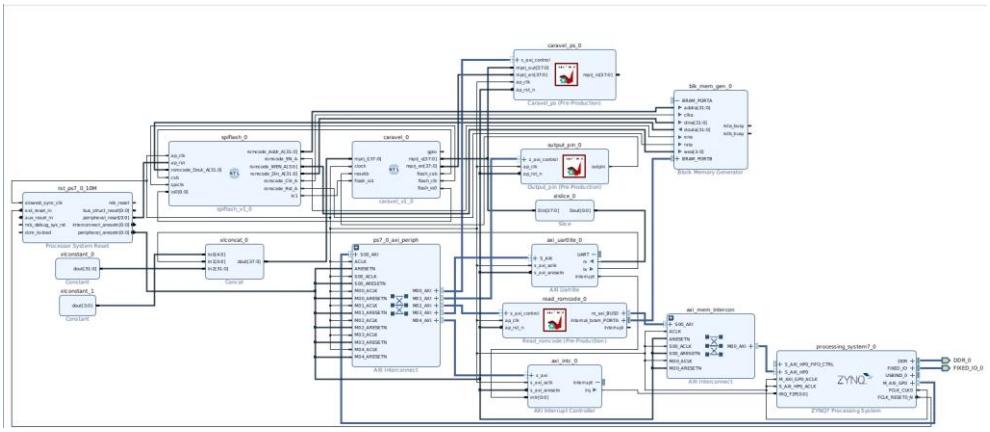
```
Start Caravel Soc
Waitting for interrupt
hello
main(): uart_rx is cancelled now
```

最後，以 mprj[31:16]為 AB51 當作結束訊號。

```
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
```

三、 Block design：



四、 Synthesis report：

1. Timing report
   - Synthesis

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	11.848 ns	Worst Hold Slack (WHS):	-0.762 ns	Worst Pulse Width Slack (WPWS):	11.250 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	-1.523 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	2	Number of Failing Endpoints:	0
Total Number of Endpoints:	13783	Total Number of Endpoints:	13783	Total Number of Endpoints:	5689

Timing constraints are not met.

   - Implementation

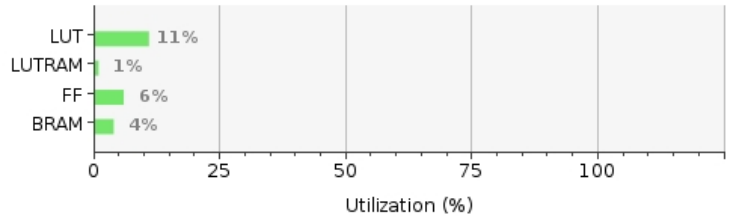
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.557 ns	Worst Hold Slack (WHS):	0.026 ns	Worst Pulse Width Slack (WPWS):	11.250 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	12669	Total Number of Endpoints:	12669	Total Number of Endpoints:	5261

All user specified timing constraints are met.

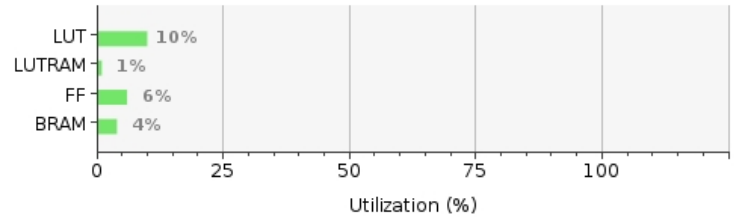
2. resource report
   - Synthesis

Resource	Utilization	Available	Utilization %
LUT	5957	53200	11.20
LUTRAM	241	17400	1.39
FF	6786	106400	6.38
BRAM	6	140	4.29



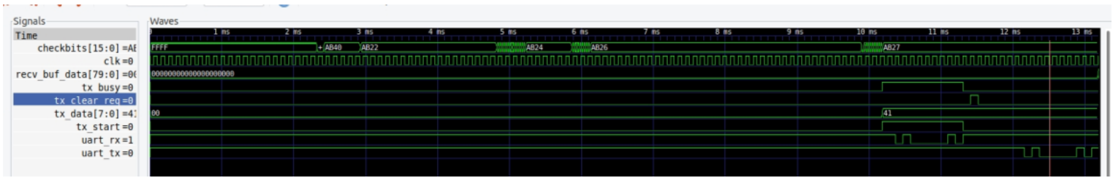
   - Implementation

Resource	Utilization	Available	Utilization %
LUT	5332	53200	10.02
LUTRAM	188	17400	1.08
FF	6159	106400	5.79
BRAM	6	140	4.29



E、 Modify include Matrix Multiplication, Quick Sort, FIR and UART

一、 Wavaform and testbench log




```
Reading uart.hex
uart.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile uart.vcd opened for output.
Matrix Multiplication 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
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
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
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
Matrix Multiplication LA Test 2 passed
Quick Sort LA Test 1 started
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0028
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x037d
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x09ed
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0a6d
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0ca1
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x10ab
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x120e
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x1787
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x2371
Quick Sort LA Test 2 passed
FIR LA Test 1 started
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0000
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0xfff6
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0xffe3
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0xffe7
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0023
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x009e
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0151
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x02dc
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0393
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x044a
UART sending data2
FIR LA Test 2 passed
tx data bit index 0: 1
tx data bit index 1: 0
tx data bit index 2: 0
tx data bit index 3: 0
tx data bit index 4: 0
tx data bit index 5: 0
tx data bit index 6: 1
tx data bit index 7: 0
tx complete 1
UART finished sending
rx data bit index 0: 1
rx data bit index 1: 0
rx data bit index 2: 0
rx data bit index 3: 0
rx data bit index 4: 0
rx data bit index 5: 0
rx data bit index 6: 1
rx data bit index 7: 0
recevied word  65
latency-timer :      409895 (cycles)
```

二、 How to verify answer from notebook：

在本實驗當中我們首先使用 Vivado 將 Matrix Multiplication、 Quick Sort,、FIR 和 UART 功能進行 Synthesis 及 Implementation，檢查完功能與 Timing 後產生 Bitstream。

再來是使用 Online FPGA 實現我們的功能，第一步將 ROM 的大小限制在 8KB，且將 Bitstream 載入我們的 PYNQ 當中，再將所使用到的 IP 匯出。

```
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
```

```
ol = Overlay("design_1.bit")
#ol.ip_dict
```

```
ipOUTPIN = ol.output_pin_0
ipPS = ol.caravel_ps_0
ipReadROMCODE = ol.read_romcode_0
ipUart = ol.axi_uartlite_0
```

```
ol.interrupt_pins
```

```
# See what interrupts are in the system
#ol.interrupt_pins

# Each IP instances has a _interrupts dictionary which lists the na
#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
intUart = ipUart.interrupt
```

第二步，將我們 Fireware 轉成的 HEX 檔打開，同時間分配一塊記憶體位址(DRAM Buffer)給我們用來將 Fireware 進行分割後以每 4byte 為單位轉入 BRAM 當中。

```
# 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("uart.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 =
        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
        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))

```

第三步，將 MMIO 以及 Buffer Size 寫入 PLA，並打入開始訊號進行 Fireware 的搬移。

```
# 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
```

第四步，將 UART 初始化並將 Register 設定好，並確認 UART 狀態。

```
# Initialize AXI UART
uart = UartAXI(ipUart.mmio.base_addr)

# Setup AXI UART register
uart.setupCtrlReg()

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

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

第五步，將 Caravel 的 reset 拉掉，使 Caravel CPU 開始讀取 Fireware 並進行運算與中斷，第一部分是進行 Matrix Multiplication，在 PYNQ 上收到 0xAB22 表示開始運算，運算結束後收取 16 筆結果進行資料驗證。第二部分是進行 Quick Sort，在 PYNQ 上收到 0xAB24 表示開始運算，運算結束後收取 10 筆結果進行資料驗證。第三部分是進行 FIR ，在 PYNQ 上收到 0xAB26 表示開始運算，運算結束後收取 11 筆結果進行資料驗證。最後一部分是進行 UART，接著打一個 UART 中斷給 Caravel 進行 TX，傳”hello”進入 Caravel，接著 Caravel 接收到後進行 UART 中斷給 PYNQ 進行 RX，我們確認在 PYNQ 上接收到”hello”。

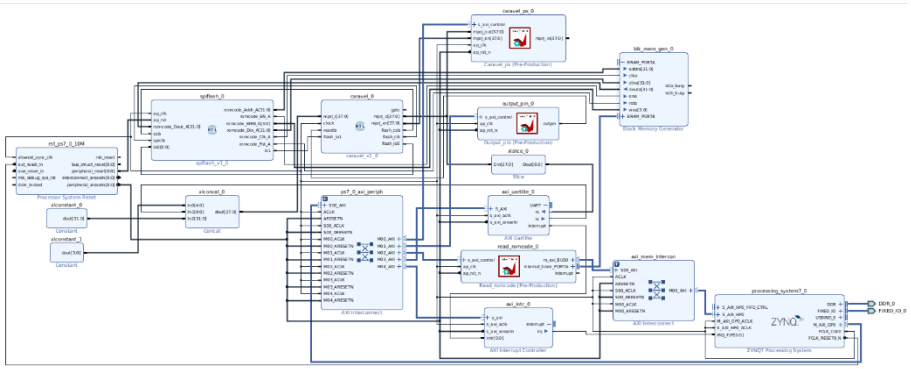
```
Start Caravel Soc
0xab220040
0x3e
0x44
0x4a
0x50
0x3e
0x44
0x4a
0x50
0x3e
0x44
0x4a
0x50
0x3e
0x44
0x4a
```

```
0x4a
0x50
0xab240040
0x28
0x37d
0x9ed
0xa6d
0xca1
0x10ab
0x120e
0x1631
0x1787
0x2371
0xab260040
0x0
0xfff6
0xffe3
0xffe7
0x23
```

```
Start Caravel Soc
0xab220040
0x3e
0x44
0x4a
0x50
0x3e
0x44
0x4a
0x50
0x3e
0x44
0x4a
0x50
0x3e
0x44
0x4a
```

```
buf_MM = []
buf_QS = []
buf_FIR = []

golden_MM = [0x3e,0x44,0x4a,0x50,0x3e,0x44,0x4a,0x50,0x3e,0x44,0x4a,0x50,0x3e,0x44,0x4a,0x50]
golden_QS = [0x28,0x37d,0x9ed,0xa6d,0xca1,0x10ab,0x120e,0x1631,0x1787,0x2371]
golden_FIR = [0x0000,0xfff6,0xffe3,0xffe7,0x0023,0x009e,0x151,0x21b,0x2dc,0x393,0x44a]

ipOUTPIN.write(0x10, 0)
print("Start Caravel Soc")
ipOUTPIN.write(0x10, 1)

# MM start-----------------------------------------------------------------------
temp = ipPS.read(0x1c)
while (temp & 0xffff0000) != 0xab220000:
    temp = ipPS.read(0x1c)
buf0 = temp

for i in range(16):
    temp = (ipPS.read(0x1c)>>16)
    while temp != golden_MM[i]:
        temp = (ipPS.read(0x1c)>>16)
    buf_MM.append(temp)
# MM end  -----------------------------------------------------------------------

# QS start-----------------------------------------------------------------------
temp = ipPS.read(0x1c)
while (temp & 0xffff0000) != 0xab240000:
    temp = ipPS.read(0x1c)
buf1 = temp

for i in range(10):
    temp = (ipPS.read(0x1c)>>16)
    while temp != golden_QS[i]:
        temp = (ipPS.read(0x1c)>>16)
    buf_QS.append(temp)
# QS end  -----------------------------------------------------------------------

# FIR start----------------------------------------------------------------------
temp = ipPS.read(0x1c)
while (temp & 0xffff0000) != 0xab260000:
    temp = ipPS.read(0x1c)
buf2 = temp

for i in range(11):
    temp = (ipPS.read(0x1c)>>16)
    while temp != golden_FIR[i]:
        temp = (ipPS.read(0x1c)>>16)
    buf_FIR.append(temp)
# FIR end  ----------------------------------------------------------------------

print(hex(buf0))
for i in range(len(buf_MM)):
    print(hex(buf_MM[i]))

print(hex(buf1))
for i in range(10):
    print(hex(buf_QS[i]))

print(hex(buf2))
for i in range(11):
    print(hex(buf_FIR[i]))

# 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]))
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))
        if i<len(tx_str):
            ipUart.write(TX_FIFO, ord(tx_str[i]))
            i=i+1
    print(buf, end='')
```

三、 Block design：



四、 Synthesis report：

1. Timing report
   - Synthesis

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	11.848 ns	Worst Hold Slack (WHS):	-0.762 ns	Worst Pulse Width Slack (WPWS):	11.250 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	-1.523 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	2	Number of Failing Endpoints:	0
Total Number of Endpoints:	13783	Total Number of Endpoints:	13783	Total Number of Endpoints:	5689

Timing constraints are not met.

   - Implementation

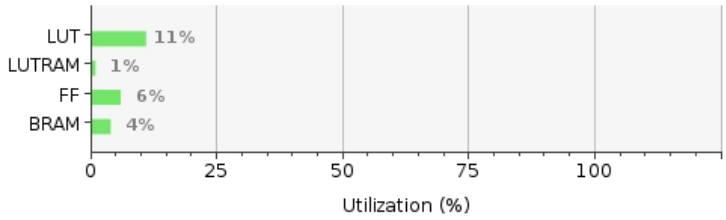
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.557 ns	Worst Hold Slack (WHS):	0.026 ns	Worst Pulse Width Slack (WPWS):	11.250 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	12669	Total Number of Endpoints:	12669	Total Number of Endpoints:	5261

All user specified timing constraints are met.

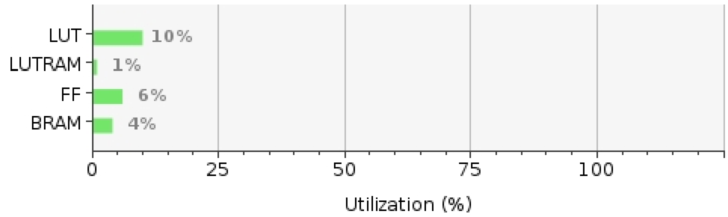
2. resource report
   - Synthesis

Resource	Utilization	Available	Utilization %
LUT	5957	53200	11.20
LUTRAM	241	17400	1.39
FF	6786	106400	6.38
BRAM	6	140	4.29



- Implementation

Resource	Utilization	Available	Utilization %
LUT	5332	53200	10.02
LUTRAM	188	17400	1.08
FF	6159	106400	5.79
BRAM	6	140	4.29



五、 Latency for a character loop back using UART

- Pre-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.
Matrix Multiplication 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
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
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
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
Matrix Multiplication LA Test 2 passed
Quick Sort LA Test 1 started
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0028
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x037d
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x09ed
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0a6d
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0ca1
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x10ab
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x120e
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x1787
Call function Quick Sort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x2371
Quick Sort LA Test 2 passed
FIR LA Test 1 started
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0000
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0xfff6
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0xffe3
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0xffe7
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0023
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x009e
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0151
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x02dc
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0393
Call function FIR() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x044a
UART sending data2
FIR LA Test 2 passed
tx data bit index 0: 1
tx data bit index 1: 0
tx data bit index 2: 0
tx data bit index 3: 0
tx data bit index 4: 0
tx data bit index 5: 0
tx data bit index 6: 1
tx data bit index 7: 0
tx complete 1
UART finished sending
rx data bit index 0: 1
rx data bit index 1: 0
rx data bit index 2: 0
rx data bit index 3: 0
rx data bit index 4: 0
rx data bit index 5: 0
rx data bit index 6: 1
rx data bit index 7: 0
recevied word  65
latency-timer :      119963 (cycles)
```

- PYNQ

```
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]))
    i = 1
    while(i<len(tx_str)):
        await intUart.wait()
        buf = ""
        uart_c_start_time = time.time_ns()
        # Read FIFO until valid bit is clear
        while ((ipUart.read(STAT_REG) & (1<<RX_VALID))):
            buf += chr(ipUart.read(RX_FIFO))
            if i<len(tx_str):
                ipUart.write(TX_FIFO, ord(tx_str[i]))
                i=i+1
        uart_c_end_time = time.time_ns()
        if buf != "":
            print("uart loop back latnecy  for ",buf," = ",uart_c_end_time - uart_c_start_time ,"ns")

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

    temp = ipPS.read(0x1c)
    while (temp & 0xffff0000) != 0xab220000:
        temp = ipPS.read(0x1c)
    start_time = time.time_ns()

    temp = ipPS.read(0x1c)
    while (temp & 0xffff0000) != 0xab270000:
        temp = ipPS.read(0x1c)
    end_time = time.time_ns()

    print("all function latnecy = ",end_time - start_time ,"ns")


# Python 3.7+
async def async_main():
    task2 = asyncio.create_task(caravel_start())
    task1 = asyncio.create_task(uart_rxtx())
    await asyncio.sleep(5)
```

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

```
Start Caravel Soc
all function latnecy =  7212364 ns
Waitting for interrupt
uart loop back latnecy  for  h  =  146846 ns
uart loop back latnecy  for  e  =  162486 ns
uart loop back latnecy  for  l  =  144428 ns
uart loop back latnecy  for  l  =  138751 ns
uart loop back latnecy  for  o  =  139372 ns
```

六、 Suggestion for improving latency for UART loop back

- 我們可以在 UART 的 TX 與 RX 部分加入 buffer，使我們可以將要傳送的資料送入 Tx 的 buffer，避免 cpu 需要等待 UART 回到 idle 才能再次啟動 UART 中間這段時間。而 Rx 的部分的 buffer 就能夠在 Tx 傳送連續的資料時立即的將資料儲存，避免前面接收到的資料被後面的覆蓋掉。

- 透過調整 Baud Rate 來提高 UART 的速度。 Baud Rate 是指每秒鐘傳輸的位數。提高 Baud Rate 會有更快的傳輸速度，但有機會提升錯誤的機率。我們想嘗試增加 Baud Rate，以找到最大的工作頻率。

F、 Observe & Learn

- 我們在 Vivado 將電路 Synthesis 與 Implementation 後發現有 timing violation 的問題，所以我們改變 implementation 的 strategy，我們將 default 改成使用 performance_NetDelay_high 後成功修復我們的 hold time violation。