SOC Design

Lab #6 – lab-wlos_baseline

Report

Group no: 5

Members:

M11207415 陳謝鎧

M11207002 陳泊佑

M11107426 廖千慧

M11207328 吳奕帆

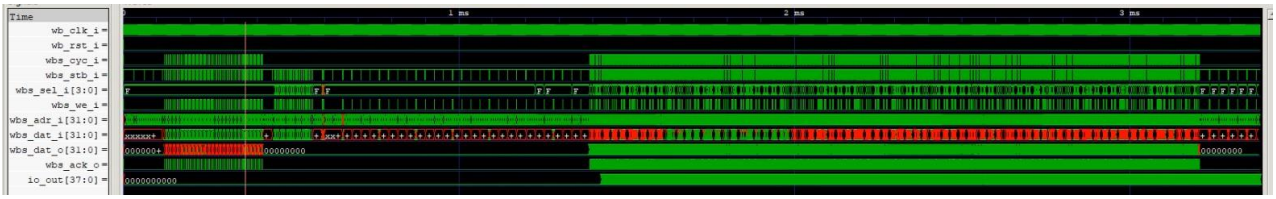
1. How do you verify your answer from notebook

1.1. Simulation on Matrix Multiplication, Quick Sort, FIR and UART separately

- Matrix Multiplication

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

```
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

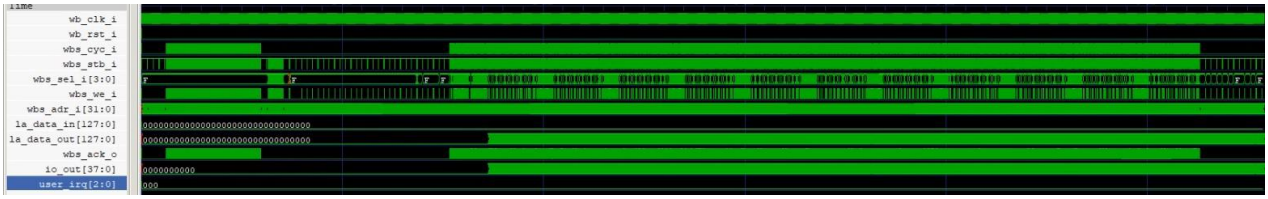
[40, 893, 2541, 2669, 3233, 4267, 4622, 5681, 6023, 9073]

```
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
ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/counter_la_qs$
```

- 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 fir() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x021b
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
LA Test 2 passed
```

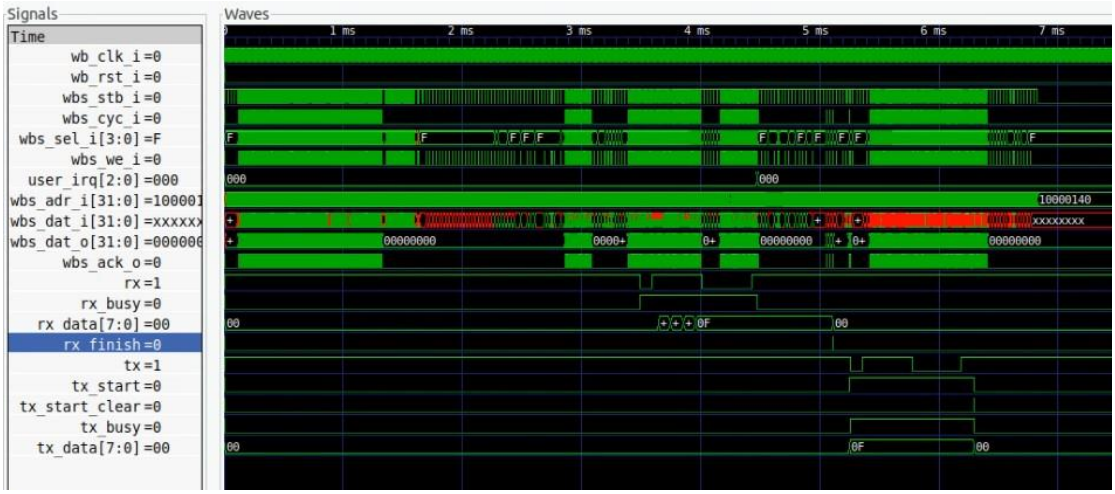


- UART

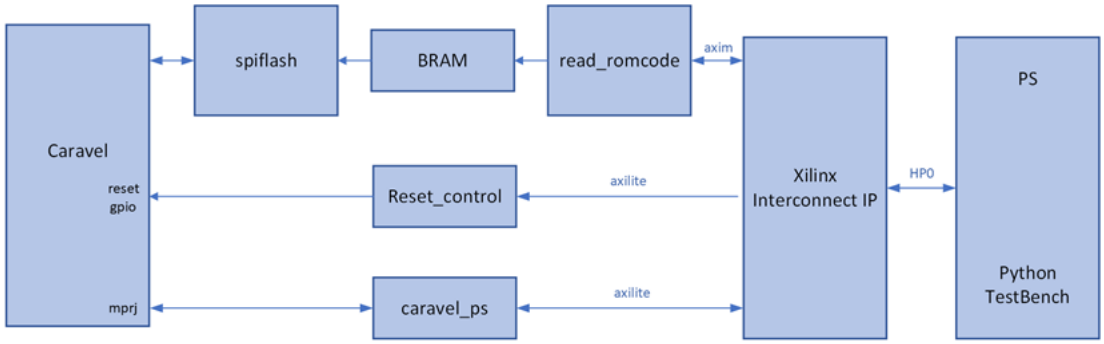
```
ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/uart$ source run_clean
ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/uart$ source run_sim
Reading uart.hex
uart.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile uart.vcd opened for output.
LA Test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

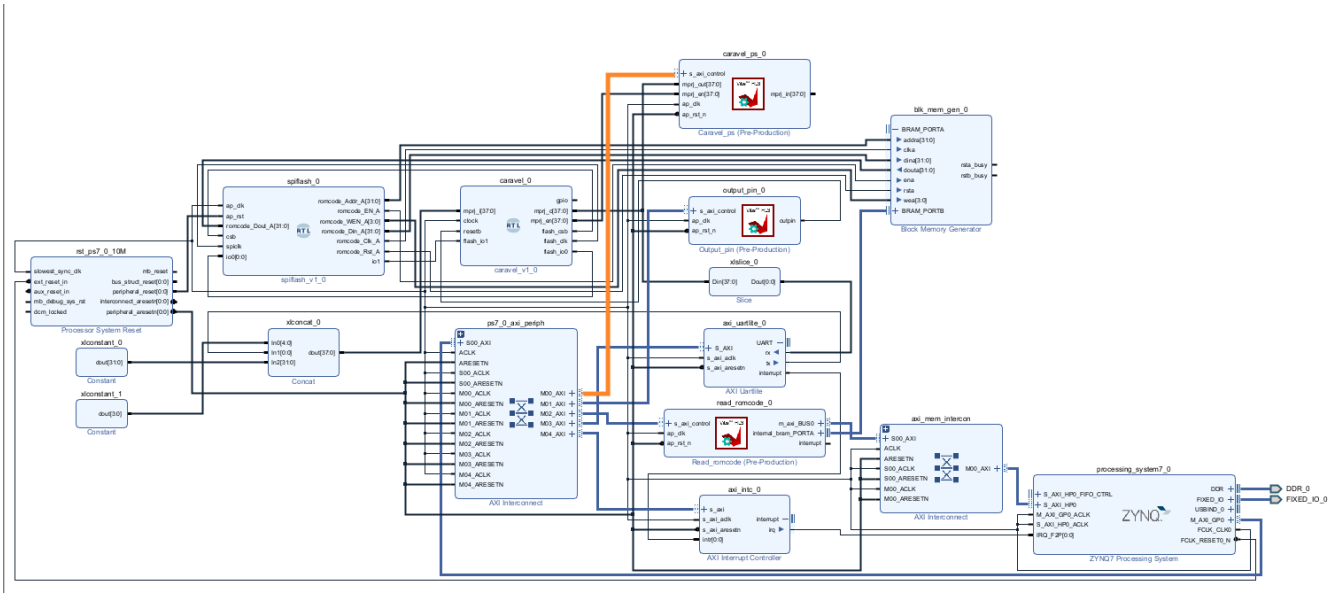
2. Simulation on integrating Matrix Multiplication, Quick Sort, FIR and UART

```
ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/all$ source run_clean
ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/all$ source run_sim
Reading all.hex
all.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile all.vcd opened for output.
uart
LA Test 1 started
qsort
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
tx data bit index 0: 1
tx data bit index 1: 1
tx data bit index 2: 1
tx data bit index 3: 1
tx data bit index 4: 0
mm
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
tx data bit index 5: 0
tx data bit index 6: 0
tx data bit index 7: 0
tx complete 1
rx data bit index 0: 1
rx data bit index 1: 1
rx data bit index 2: 1
rx data bit index 3: 1
rx data bit index 4: 0
rx data bit index 5: 0
rx data bit index 6: 0
rx data bit index 7: 0
recevied word  15
fir
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x021b
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

Monitor: Timeout, Test LA (RTL) Failed

ubuntu@ubuntu2004:~/Desktop/lab-wlos_baseline/testbench/all$
```



3. Block design





這次會多 axi-uartlite 跟 Interrupt Control 這兩個 ip，

4.Timing report/ resource report after synthesis

- **Timing report**

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             11.295ns  (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:        3.680ns  (logic 0.225ns (6.115%)  route 3.455ns (93.885%))
  Logic Levels:           2  (BUFG=1 LUT6=1)
  Clock Path Skew:        2.820ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.820ns = ( 27.820 - 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
```

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             17.406ns  (required time - arrival time)
  Source:                 design_1_i/output_pin_0/inst/control_s_axi_U/int_outpin_ctrl_reg[0]/C
                            (rising edge-triggered cell FDRE clocked by clk_fpga_0  {rise@0.000ns fall@12.500ns period=25.000ns})
  Destination:            design_1_i/caravel_0/inst/housekeeping/serial_data_staging_2_reg[12]/CLR
                            (recovery check against rising-edge clock clk_fpga_0  {rise@0.000ns fall@12.500ns period=25.000ns})
  Path Group:             **async_default**
  Path Type:              Recovery (Max at Slow Process Corner)
  Requirement:            25.000ns  (clk_fpga_0 rise@25.000ns - clk_fpga_0 rise@0.000ns)
  Data Path Delay:        6.715ns  (logic 0.580ns (8.637%)  route 6.135ns (91.363%))
  Logic Levels:           1  (LUT1=1)
  Clock Path Skew:        -0.097ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.719ns = ( 27.719 - 25.000 )
    Source Clock Delay      (SCD):    2.945ns
    Clock Pessimism Removal (CPR):    0.129ns
  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**

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	5381	0	0	53200	10.11
LUT as Logic	5193	0	0	53200	9.76
LUT as Memory	188	0	0	17400	1.08
LUT as Distributed RAM	18	0			
LUT as Shift Register	170	0			
Slice Registers	6162	0	0	106400	5.79
Register as Flip Flop	6162	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. Latency for a character loop back

5-1. Use UART

In [22]:
```
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)
```

In [23]:
```
asyncio.run(async_main())
```

```
Start Caravel Soc
Waitting for interrupt
latency time: 0.007434129714965820
hlatency time: 0.013600349426269531
elatency time: 0.019428491592407227
llatency time: 0.024479389190673828
llatency time: 0.02961897850036621
olatency time: 0.03439927101135254

main(): uart_rx is cancelled now
```

- **Latency**

1. start tx to h : 7.4 - 0 = 7.4ms
2. receive h to receive e : 13.6 –7.4 = 6.2ms
3. receive e to receive l : 19.4 – 13.6 = 5.8s
4. receive l to receive l : 24.4 – 19.4 = 5ms

5. receive l to receive o : 29.6 – 24.4 = 5.2ms

5-2. Use integrate

```
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:",(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)
```

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

         Start Caravel Soc
         Waitting for interrupt
         latency: 0.004868507385253906
         hlatency: 0.011801004409790039
         elatency: 0.019613027572631836
         llatency: 0.026221036911010742
         llatency: 0.03118133544921875
         olatency: 0.035891056060791016

         main(): uart_rx is cancelled now
```

- **Latency**

1. start tx to h : 4.9 - 0 = 4.9ms
2. receive h to receive e : 11.8 –4.9 = 6.9ms
3. receive e to receive l : 19.6 – 11.8 = 7.8ms
4. receive l to receive l : 26.2 – 19.6 = 6.6ms
5. receive l to receive o : 31.1 – 26.2 = 4.9ms

6. Screenshot of Execution result on workload

6.1 Execute “uart.hex”

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
```

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("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 = 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
        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}
```

In [19]:
```
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 new coroutine.
# 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 interrupts

#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')
```

In [20]:
```
asyncio.run(async_main())
```

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

In [21]:
```
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
```

由執行結果得知，在 0X1C 的位置上值為 0XAB51。

6.2 Execute “integrate.hex”

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
```

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("all.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
        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}
```

In [9]:
```
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 new coroutine.
# 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 interrupts

#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')
```

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

```
Start Caravel Soc
Waitting for interrupt
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 =  0xab510040
0x20 =  0x0
0x34 =  0x20
0x38 =  0x3f
```

由執行結果得知，在 0X1C 的位置上值為 0XAB51。

7. Suggestion for improving latency for UART loop back

1. 使用 FIFO（First In, First Out）。

在 UART 通信中，FIFO 可以用來緩存接收到的數據，而不是立即將每個字節的數據進行處理。

這樣的做法可以減少中斷的頻率，因為不需要每收到一個字節就觸發一次中斷。相反，當 FIFO 中的數據達到一定數量，或者某些特定的事件發生時，再一次性地處理 FIFO 中的數據。這樣可以提高效率，減少中斷處理的開銷。

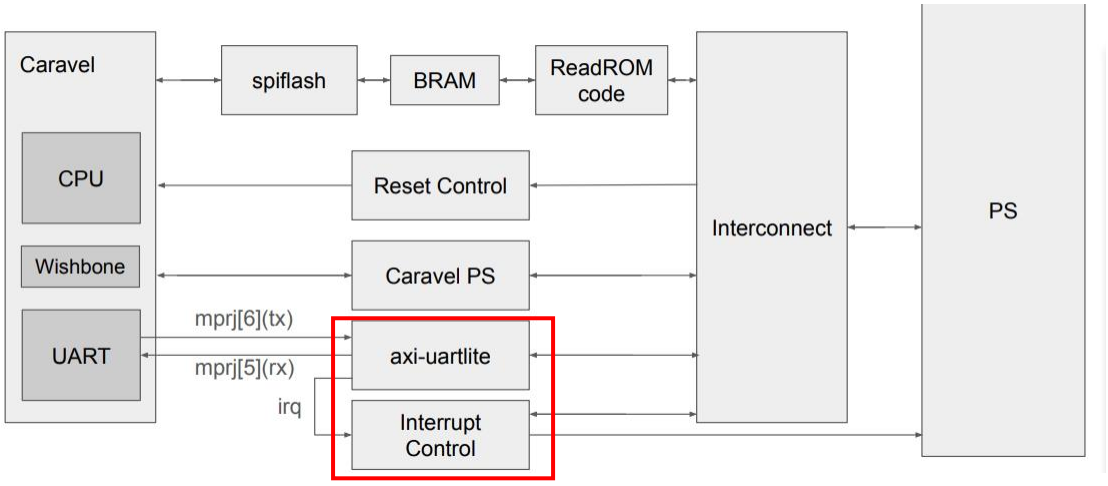
實現：若要增加 FIFO 到 UART，需要定義 confiquration registers 用於 uart 和 fifo 的 handshake 以及處理跟 firmware 之間的 hankshake。

2. 可以提高 baudrate 來增加數據傳輸。

3. 使用 DMA 進行接收。

8. What else do you observe

這次 lab 主要分為 2 個部分，主要是驗證 firmware 端的 4 個功能正確性，以及把它們整合在一起，最後在放到 FPGA 來上來進行驗證，相較於 LAB5 此次多了兩個 IP，分別是 axi-uartlite 和 interrupt-Control，當收到資料時，需要產生中斷訊號的 ip 以及收資料的 ip，運作原理如下



當 ps 端寫一筆 data 給 caravel，它會透過 axi-interconnect 送到 axi-uartlite，然後 caravel 會去接這筆資料，data 會經過 axi-uartlite-rx 傳給 caravel_uart-tx，而從 caravel 端要傳 data 給 ps 端，data 會經由 caravel_uart-rx 傳給 axi-uartlite-tx，然後當 axi-uartlite 接收到資料時，它會去 trigger interrupt，而資料會經由 interrupt-ctrl 傳給 ps 端。(ps 端的 irq 會讓 ps 端知道已經有資料從 caravel 送過來了)

而其他的 ip 功能跟 lab5 相同，執行 pythin code 來進行驗

證，透過 ip 來完成資料的傳遞。

透過 read_romcode ip 來將.hex 傳遞。將資料從 DDR 內傳遞至硬體 BRAM 內。

透過 RsetControl ip 來溝通 PS 與 Caravel 內的 MPRJ_IO。用此 ip 設定 reset 信號控制，1 或 0 分別控制 Caravel 的 reset pin 是 assert 還是 de-assert。

透過 MPRJ_IO 經由 caravel_ps 這個 ip 在執行完後溝通。提供 AXI Lite 接口供 PS CPU 讀取 MPRJ_IO/OUT/EN bit，用 HLS 實現並導出 IP 以供 Vivado 使用。

在此次實驗過程中，因為忘了在 include.rtl.list 放入修改的檔案路徑，導致一直跑出 error，以及一開始在 run firmware 時會一直出現 time_out，所幸最後都有在 github 上找到解答。