國立清華大學

系統晶片設計

SOC Design

國立清華大學

NATIONAL TSING HUA UNIVERSITY

Lab 6

組別: 第 12 組

學號:111063548、111061624、112501538

姓名:蕭方凱、尤弘瑋、葉承泓

指導老師:賴瑾教授

目錄

1. How do you verify your answer from notebook?

在透過 Vivado 軟體的.tcl 檔執行完 synthesis 及 implementation 後，產生出 caravel_fpga.bit 及 caravel_fpga.hwh 這兩個檔案，我們將這兩個檔案和 caravel_fpga_uart.ipynb、uartlite.py、uart.hex，共五個檔案（也就是在作業區繳交的 Required_files_for_FPGA/lab6 implementation 資料夾中的五個檔案）上傳到 onlineFPGA 的網頁上。接著嘗試修改 caravel_fpga_uart.ipynb 檔案的內容，以達到檢驗 firmware 及 hardware 是否正常運作的效果，我從題目所連結的 Github 上原有的 template code（執行 only UART 的 part）為基底進行修改，並增加一些功能，按照程式碼的執行流程說明如下：

(1) 首先，先進行 FPGA 以及其與 Caravel SoC 兩者之間 interface 的初始狀態設置，包含 import 相關 library

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
```

(2) 讀取.bit 檔資料以得知整個 Caravel SoC 的 block 資訊

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

(3) 定義不同 interface 的名稱，例如 ipOUTPIN 對應到 output_pin、ipPS 對應到 caravel_ps 等，其中在這個 lab 多使用了 UARTLite 的 interface，因此也指定 ipUart 對應到 axi_uartlite

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
```

(4) 將 firmware code 的.hex 檔案讀取並放入 ROM 中

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))
```

(5) 將 ROM code 的 size 等相關資訊寫入 ipROMCODE 的 0x10、0x14、0x1C 等位址中。最後透過將 ipROMCODE 的 0x00 的位址 program 為 1 來表示 IP start，而開始將這些 code 由 rom_buffer 搬移到 spiflash 所接著的 BRAM 中，直到搬移結束

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
```

(6) 接著 initialize UART，並存取 UART 的 status register 以確認 Rx 仍未 full 的狀態（代表才可以傳入 data）

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}
```

以上的部分幾乎與原始程式相同，底下是主要有修改到的部分：

(7) uart_rxtx()函式主要是用來傳輸"hello\n"字串給 UART 的 Rx 端，並且接收 UART 的 Tx 端所傳回的資訊，最後將回傳的結果 print 在螢幕上

```
async def uart_rxtx():
    # Reset FIFOs, enable interrupts
    ipUart.write(CTRL_REG, 1<<RST_TX | 1<<RST_RX | 1<<INTR_EN)
    print("Waiting for interrupt")
    tx_str = "hello\n"
    ############################## Added by us ##############################
    latency_timer1_start=time()
    #########################################################################
    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
        ############################## Added by us ##############################
        latency_timer1_end=time()
        #########################################################################
        print(buf, end='')
        ############################## Added by us ##############################
        print('\nNotebook received "'+buf+'", the same string as what we sent, and the latency for this string using UART = ', latency_timer1_end-latency_timer1_start," (s).")
        #########################################################################
```

我們在其中加入了 latency timer 以計算傳出到接收回來之間共需要花多久的時間，也就是 UART 運作的 latency。（這部分的結果會在第 4.題中做說明。）

(8) caravel_start()函式則是用來 check 這次 lab 中 3 個 workload（在我們的實作中，依序執行 matrix multiplication matmul() → quick sort qsort() → FIR fir()）是否有正確執行完畢：（由於程式碼較長，因此分為 2 部分截圖）

```
async def caravel_start():
    ipOUTPIN.write(0x10, 0)
    print("Start Caravel SoC")
    ipOUTPIN.write(0x10, 1)
    ############################## Added by us ##############################

    while ((ipPS.read(0x1c)>>16) != 0xab40):
        continue
    # Because print() function takes a lot of time to execute (resulting in missing mprj_io code), we cannot print the information here, and we will print all the information together at last.

    ###print("Info: Start matrix multiplication test...")
    while ((ipPS.read(0x1c)>>16) != 0x0044):
        continue
    ###print("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0044, which is 68 in decimal")

    while ((ipPS.read(0x1c)>>16) != 0x0050):
        continue
    ###print("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0050, which is 80 in decimal")


    while ((ipPS.read(0x1c)>>16) != 0xab51):
        continue
    ###print("Success: Matrix multiplication test passed ♪")


    ###print("Info: Start Q sort test...")
    while ((ipPS.read(0x1c)>>16) != 2541):
        continue
    ###print("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x09ed, which is 2541 in decimal")


    while ((ipPS.read(0x1c)>>16) != 6023):
        continue
    ###print("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x1787, which is 6023 in decimal")

    while ((ipPS.read(0x1c)>>16) != 0xab52):
        continue
    ###print("Success: Q sort test passed ♪")


    ###print("Info: Start FIR test...")
    while ((ipPS.read(0x1c) & 0xffff0000) != 0xffe70000):
        continue
    ###print("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value passed, 0xffe7, which is -25 in decimal")

    while ((ipPS.read(0x1c)>>16) != 732):
        continue
    ###print("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x02dc, which is 732 in decimal")

    while ((ipPS.read(0x1c)>>16) != 0xab60):
        continue
    ###print("Success: FIR test passed ♪")
```

```
    print("Successfully start matrix multiplication test (checkbits = 0xAB40)")
    ####print("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x003e, which is 62 in decimal")
    print("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0044, which is 68 in decimal")
    #####print("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x004a, which is 74 in decimal")
    print("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0050, which is 80 in decimal")
    print("Success: Matrix multiplication test passed ♪ (checkbits = 0xAB51)")
    print("Successfully start Q sort test (checkbits = 0xAB51)")
    #####print("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0028, which is 40 in decimal")
    #####print("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x037d, which is 893 in decimal")
    print("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x09ed, which is 2541 in decimal")
    #####print("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0a6d, which is 2669 in decimal")
    print("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x1787, which is 6023 in decimal")
    print("Success: Q sort test passed ♪ (checkbits = 0xAB52)")
    print("Successfully start FIR test (checkbits = 0xAB52)")
    #####print("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x044a, which is 1098 in decimal")
    print("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value passed, 0xffe7, which is -25 in decimal")
    print("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x02dc, which is 732 in decimal")
    print("Success: FIR test passed ♪ (checkbits = 0xAB60)")
    ########################################################################
```

上圖程式碼的執行流程大致如下：首先先將 ipOUTPIN 的 0x10 位置 program 為 0，以開始 Caravel SoC 的運作，接著由於我們在 counter_la_uart.c 中依序將 reg_mprj_datal 的值令為：

A. reg_mprj_datal=0xAB40 表示開始執行 matrix multiplication
B. matrix multiplication 執行完畢後將結果依序指定在 reg_mprj_datal 上

以方便 testbench 觀察

C. reg_mprj_datal=0xAB51 表示 matrix multiplication 執行完畢，並開始執行 quick sort
D. quick sort 執行完畢後將結果依序指定在 reg_mprj_datal 上以方便 testbench 觀察
E. reg_mprj_datal=0xAB52 表示 quick sort 執行完畢，並開始執行 FIR
F. FIR 執行完畢後將結果依序指定在 reg_mprj_datal 上以方便 testbench 觀察
G. reg_mprj_datal=0xAB60 表示 FIR 執行完畢

因此我們只需要特地去 check 0xAB40、0xAB51、0xAB52、0xAB60 這四個特定值有無出現在 reg_mprj_datal 所對應的 pin 腳（也就是透過 ipPS.read(0x1c)即可讀取到此值），即可知道 Caravel 中的 CPU 是否有成功完成這些 workload，並且因為在每個 workload 執行結束後，都會將部分結果也放在 reg_mprj_datal 上，因此也可以去看看此 pin 腳上的訊號有沒有在正確的時間出現 golden value。但由於 PS 端（Jupyter Notebook 端）的 check 執行較 Caravel SoC 中慢，也就是有可能會發生 miss（golden 值還沒被偵測到就已經被改掉了）的情形，且特別是在一個 workload 運算完畢後，會快速將一筆一筆資料接續放至 reg_mprj_datal 上，導致此時的訊號變化快速，因此我們很難連續透過 Jupyter Notebook 端偵測到這些值的快速（相對於 PS 端而言）變化，因此我們只挑選其中幾個值來進行判斷，只要這些零散分布的值都計算正確，且最後的 endmark 也都有偵測到，就當作是 CPU 有正確執行 workload。另外，由於 print()函數會需要耗費許多時間執行，若 PS 端在偵測到正確的數值後就立刻 print information 的話，有可能會太慢而導致下一筆資料偵測不到，故我們實作上會直到全部都偵測完畢（表示 CPU 有正確依照我們所期待的流程執行，且過程中結果也正確無誤），才一起 print 出。最後的結果如第(9)點所示。

(9) 透過 asyncio 的 library 可以將 workload 與 UART 分成 2 個 task，當其中一方不需要用到 CPU 時（例如執行 asyncio.sleep()），就會自動將 CPU 轉調給另一個 task 使用，以促進 CPU 的使用率(utilization)。執行結果如下：

```
# Python 3.7+
async def async_main():
    task2 = asyncio.create_task(caravel_start())
    ############################## Added by us ##############################
    #await asyncio.sleep(1)
    ###task3 = asyncio.create_task(check_matrix_multiplication())
    #####task1 = asyncio.create_task(uart_rxtx_for_only_one_char())
    #########################################################################
    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
Successfully start matrix multiplication test (checkbits = 0xAB40)
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0044, which is 68 in decimal
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0050, which is 80 in decimal
Success: Matrix multiplication test passed ♪ (checkbits = 0xAB51)
Successfully start Q sort test (checkbits = 0xAB51)
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x09ed, which is 2541 in decimal
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x1787, which is 6023 in decimal
Success: Q sort test passed ♪ (checkbits = 0xAB52)
Successfully start FIR test (checkbits = 0xAB52)
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value passed, 0xffe7, which is -25 in decimal
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x02dc, which is 732 in decimal
Success: FIR test passed ♪ (checkbits = 0xAB60)
Waitting for interrupt

Notebook received "", the same string as what we sent, and the latency for this string using UART =  0.00162506103515625  (s).
h
Notebook received "h", the same string as what we sent, and the latency for this string using UART =  0.007600069046020508  (s).
e
Notebook received "e", the same string as what we sent, and the latency for this string using UART =  0.013179302215576172  (s).
l
Notebook received "l", the same string as what we sent, and the latency for this string using UART =  0.018865346908569336  (s).

Notebook received "", the same string as what we sent, and the latency for this string using UART =  0.020744800567626953  (s).
l
Notebook received "l", the same string as what we sent, and the latency for this string using UART =  0.02974081039428711  (s).

Notebook received "", the same string as what we sent, and the latency for this string using UART =  0.031629323959350586  (s).
o
Notebook received "o", the same string as what we sent, and the latency for this string using UART =  0.03348278999328613  (s).


Notebook received "
", the same string as what we sent, and the latency for this string using UART =  0.043314218521118164  (s).
main(): uart_rx is cancelled now
```

(10) 最後再 check 目前的 reg_mprj_datal 值，確認是否符合期待：

In [11]:

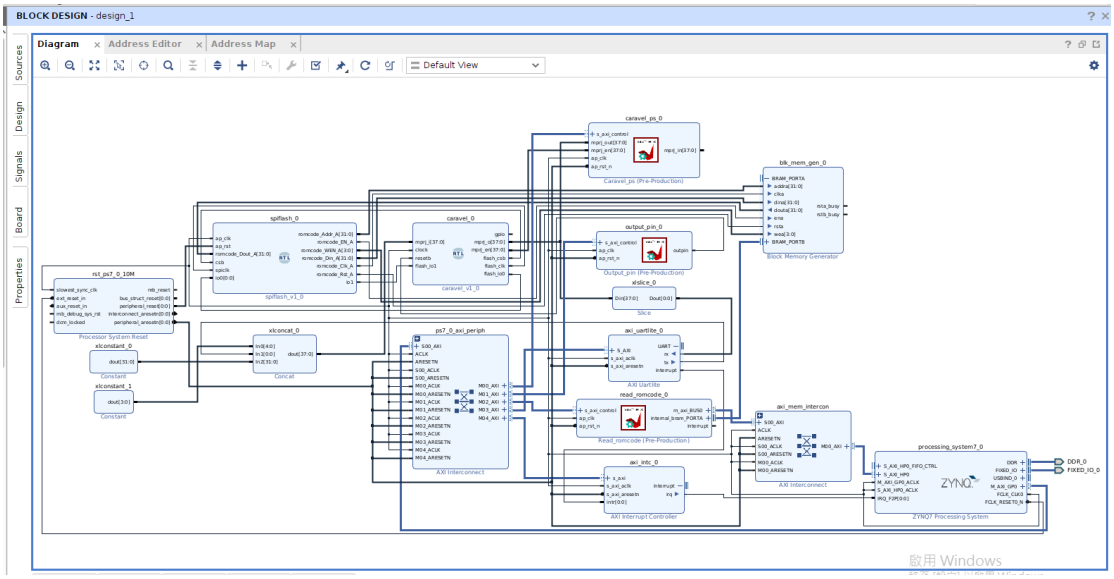
```
print ("0x10 = ", hex(ipPS.read(0x10)))
print ("0x14 = ", hex(ipPS.read(0x14)))
print ("0x1c = ", hex(ipPS.read(0x1c)))
print ("0x20 = ", hex(ipPS.read(0x20)))
print ("0x34 = ", hex(ipPS.read(0x34)))
print ("0x38 = ", hex(ipPS.read(0x38)))
```

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

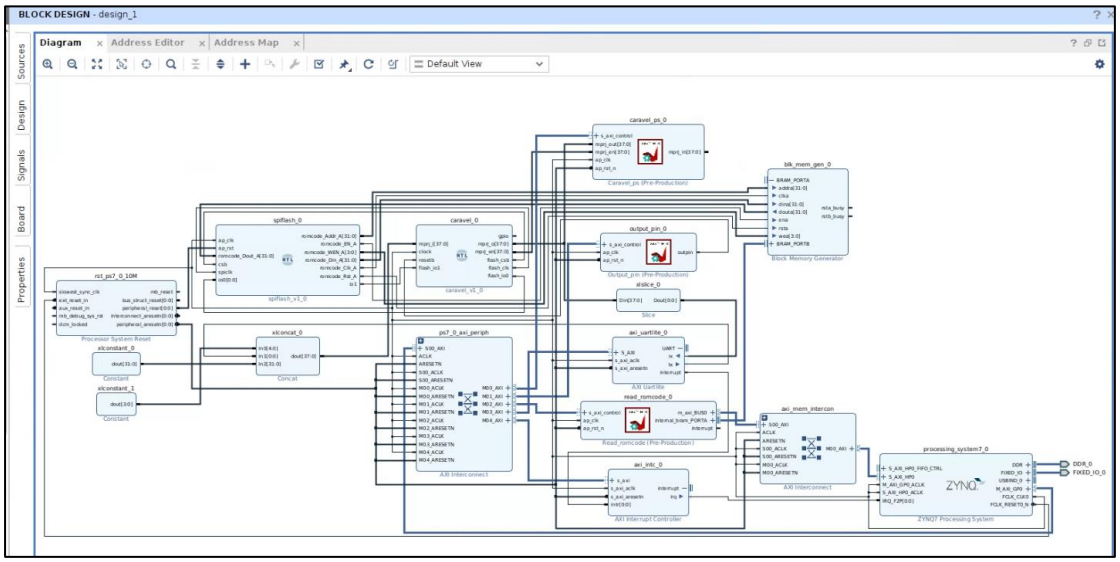
由於在此配置下 UART 比其他 workload 還要晚完成，且我們在 /firmware/isr.c 中設定當 UART 偵測到'\n'字元表示為傳輸的字串中最後一個字元，此時會將 reg_mprj_datal 值設置為 0xAB61。故結果與預期相符合！

2. Block design

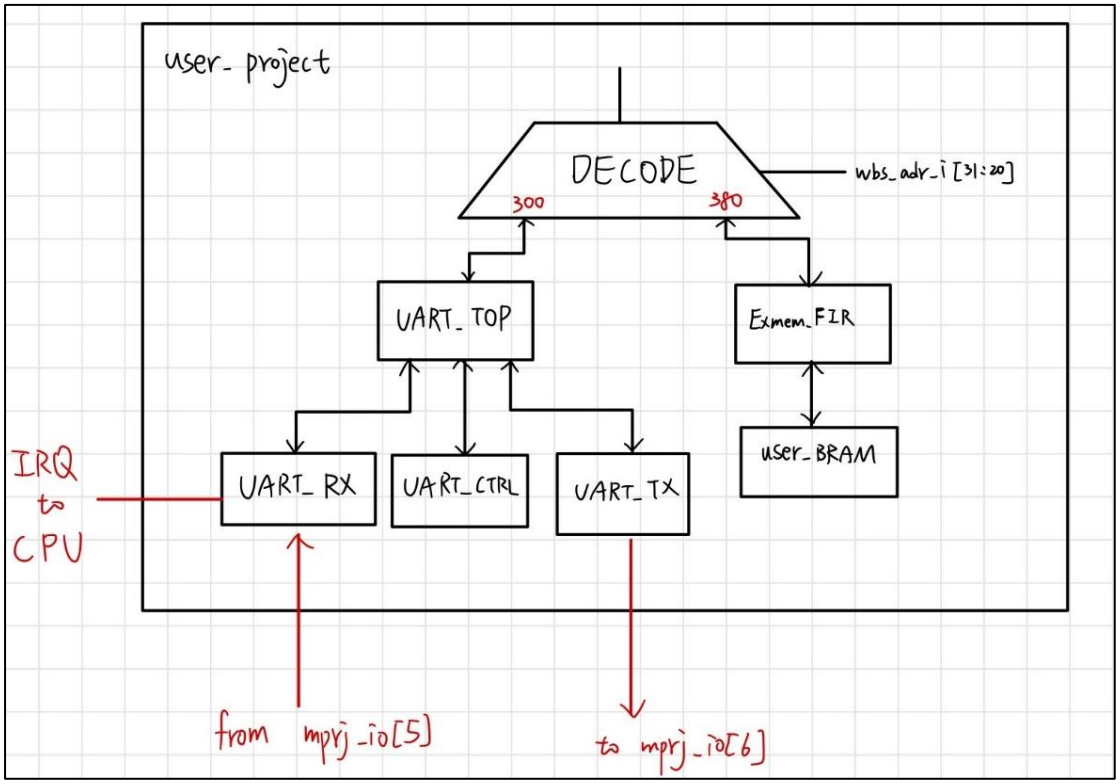
(1). Whole block design (when user project area has only UART module)



(2). Whole block design (when user project has only UART+user_BRAM)



Block diagram :



Decode (“rtl/user/user_proj_example.counter.v”)

```
// IRQ
assign irq = 3'b000;    // Unused

// LA
assign la_data_out = {{(127-BITS){1'b0}}, count};
// Assuming LA probes [63:32] are for controlling the count register
assign la_write = ~la_oenb[63:32] & ~{BITS{valid}};
// Assuming LA probes [65:64] are for controlling the count clk & reset
assign clk = (~la_oenb[64]) ? la_data_in[64]: wb_clk_i;
assign rst = (~la_oenb[65]) ? la_data_in[65]: wb_rst_i;

assign decoded = wbs_adr_i[31:20] == 12'h380 ? 1'b1 : 1'b0;
always @(posedge clk) begin
    if (rst) begin
        ready <= 1'b0;
        delayed_count <= 16'b0;
    end else begin
        ready <= 1'b0;
        if ( valid && !ready ) begin
            if ( delayed_count == DELAYS ) begin
                delayed_count <= 16'b0;
                ready <= 1'b1;
            end else begin
                delayed_count <= delayed_count + 1;
            end
        end
    end
end
```

Integrate all task(“testbench/lab6_integration/counter_la_uart.c”)

```
////////////////////////////// matrix multiplication (counter_la_mm.c) //////////////////////////////
int* tmp = matmul();
reg_mprj_datal = *tmp << 16;
reg_mprj_datal = *(tmp+1) << 16;
reg_mprj_datal = *(tmp+2) << 16;
reg_mprj_datal = *(tmp+3) << 16;

reg_mprj_datal = *(tmp+9) << 16;

reg_mprj_datal = 0xAB510000;

////////////////////////////// Q sort (counter_la_qs.c) //////////////////////////////
tmp = qsort();
reg_mprj_datal = *tmp << 16;
reg_mprj_datal = *(tmp+1) << 16;
reg_mprj_datal = *(tmp+2) << 16;
reg_mprj_datal = *(tmp+3) << 16;
reg_mprj_datal = *(tmp+4) << 16;
reg_mprj_datal = *(tmp+5) << 16;
reg_mprj_datal = *(tmp+6) << 16;
reg_mprj_datal = *(tmp+7) << 16;
reg_mprj_datal = *(tmp+8) << 16;
reg_mprj_datal = *(tmp+9) << 16;

reg_mprj_datal = 0xAB520000;
```

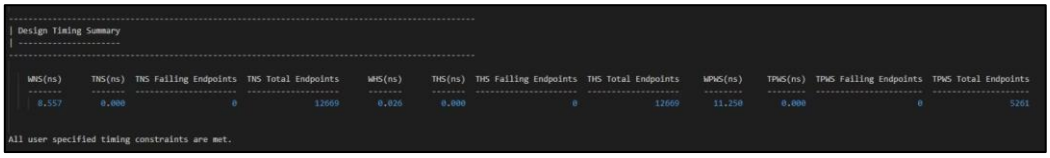
```
////////////////////////////// FIR (counter_la_fir.c) //////////////////////////////
tmp = fir();
reg_mprj_datal = *tmp << 16;
reg_mprj_datal = *(tmp+1) << 16;
reg_mprj_datal = *(tmp+2) << 16;
reg_mprj_datal = *(tmp+3) << 16;
reg_mprj_datal = *(tmp+4) << 16;
reg_mprj_datal = *(tmp+5) << 16;
reg_mprj_datal = *(tmp+6) << 16;
reg_mprj_datal = *(tmp+7) << 16;
reg_mprj_datal = *(tmp+8) << 16;
reg_mprj_datal = *(tmp+9) << 16;
reg_mprj_datal = *(tmp+10) << 16;

reg_mprj_datal = 0xAB600000;
```

3. Timing report/ resource report after synthesis

(1). Only UART module

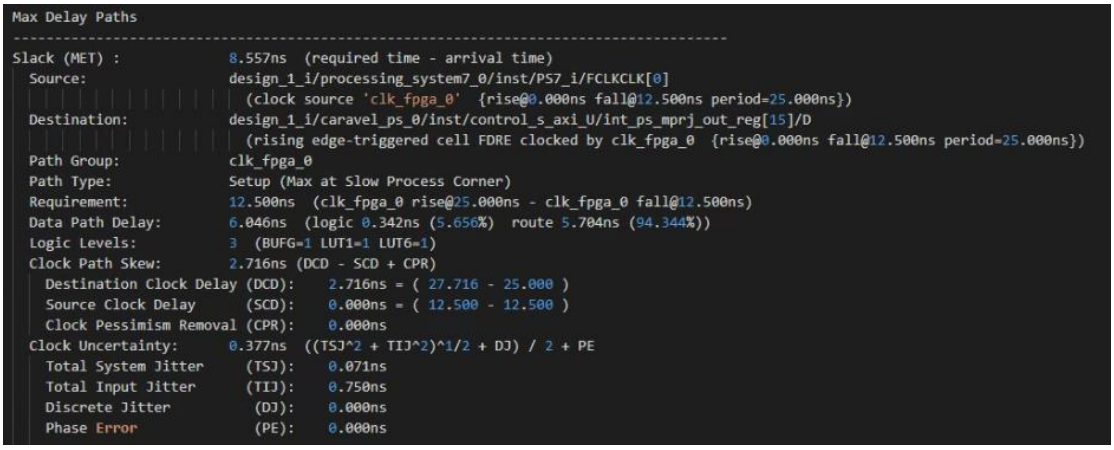
A. Timing report



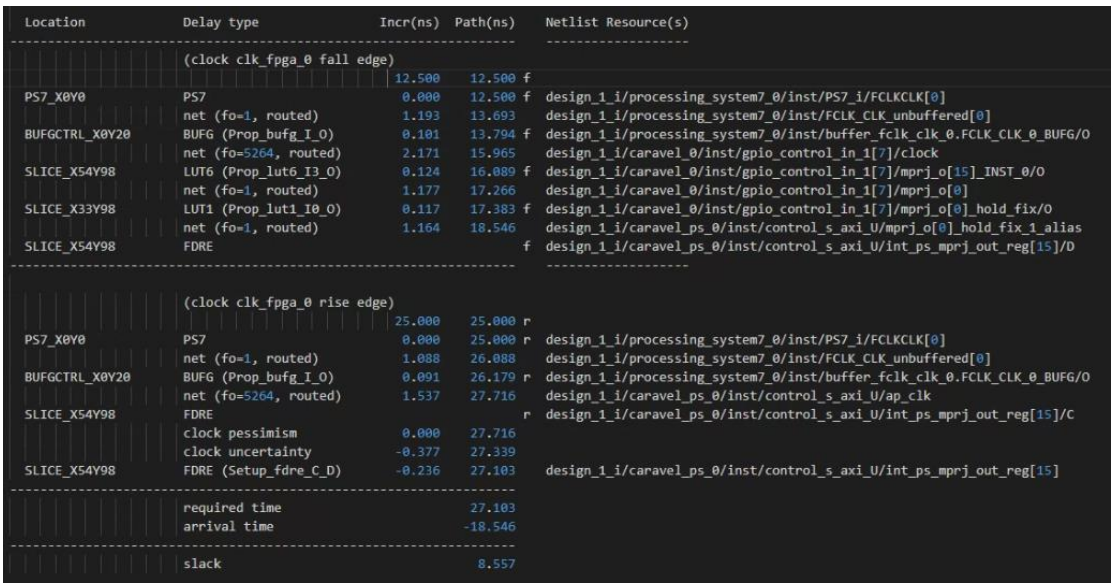
WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints	WPWS(ns)	TPWS(ns)	TPWS Failing Endpoints	TPWS Total Endpoints
8.557	0.000	0	12669	0.026	0.000	0	12669	11.250	0.000	0	5261

All user specified timing constraints are met.

Longest path



```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             8.557ns  (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:        6.046ns  (logic 0.342ns (5.656%)  route 5.704ns (94.344%))
  Logic Levels:           3  (BUFG=1 LUT1=1 LUT6=1)
  Clock Path Skew:        2.716ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.716ns = ( 27.716 - 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
```



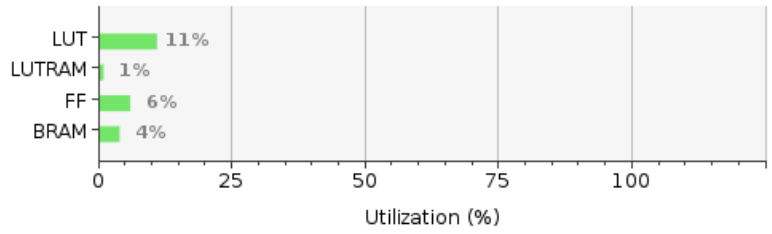
```
  Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
-------------------------------------------------------------------    -------------------
                       (clock clk_fpga_0 fall edge)
                                                 12.500    12.500 f
  PS7_X0Y0             PS7                          0.000    12.500 f  design_1_i/processing_system7_0/inst/PS7_i/FCLKCLK[0]
                       net (fo=1, routed)           1.193    13.693    design_1_i/processing_system7_0/inst/FCLK_CLK_unbuffered[0]
  BUFGCTRL_X0Y20       BUFG (Prop_bufg_I_O)         0.101    13.794 f  design_1_i/processing_system7_0/inst/buffer_fclk_clk_0.FCLK_CLK_0_BUFG/O
                       net (fo=5264, routed)        2.171    15.965    design_1_i/caravel_0/inst/gpio_control_in_1[7]/clock
  SLICE_X54Y98         LUT6 (Prop_lut6_I3_O)        0.124    16.089 f  design_1_i/caravel_0/inst/gpio_control_in_1[7]/mprj_o[15]_INST_0/O
                       net (fo=1, routed)           1.177    17.266    design_1_i/caravel_0/inst/gpio_control_in_1[7]/mprj_o[0]
  SLICE_X33Y98         LUT1 (Prop_lut1_I0_O)        0.117    17.383 f  design_1_i/caravel_0/inst/gpio_control_in_1[7]/mprj_o[0]_hold_fix/O
                       net (fo=1, routed)           1.164    18.546    design_1_i/caravel_ps_0/inst/control_s_axi_U/mprj_o[0]_hold_fix_1_alias
  SLICE_X54Y98         FDRE                                       f  design_1_i/caravel_ps_0/inst/control_s_axi_U/int_ps_mprj_out_reg[15]/D
-------------------------------------------------------------------    -------------------

                       (clock clk_fpga_0 rise edge)
                                                 25.000    25.000 r
  PS7_X0Y0             PS7                          0.000    25.000 r  design_1_i/processing_system7_0/inst/PS7_i/FCLKCLK[0]
                       net (fo=1, routed)           1.088    26.088    design_1_i/processing_system7_0/inst/FCLK_CLK_unbuffered[0]
  BUFGCTRL_X0Y20       BUFG (Prop_bufg_I_O)         0.091    26.179 r  design_1_i/processing_system7_0/inst/buffer_fclk_clk_0.FCLK_CLK_0_BUFG/O
                       net (fo=5264, routed)        1.537    27.716    design_1_i/caravel_ps_0/inst/control_s_axi_U/ap_clk
  SLICE_X54Y98         FDRE                                       r  design_1_i/caravel_ps_0/inst/control_s_axi_U/int_ps_mprj_out_reg[15]/C
                       clock pessimism              0.000    27.716
                       clock uncertainty           -0.377    27.339
  SLICE_X54Y98         FDRE (Setup_fdre_C_D)       -0.236    27.103    design_1_i/caravel_ps_0/inst/control_s_axi_U/int_ps_mprj_out_reg[15]
-------------------------------------------------------------------
                       required time                         27.103
                       arrival time                         -18.546
-------------------------------------------------------------------
                       slack                                 8.557
```

B. Resource/Utilization report

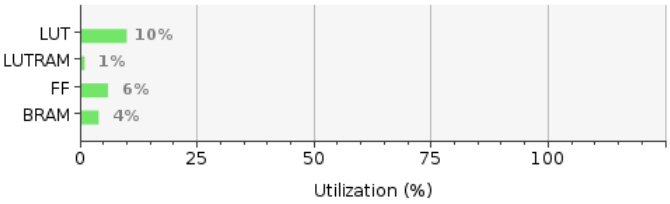
(After synthesis)

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



(After 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



(2). UART + user_BRAM

A. Timing report

Longest path

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             8.726ns  (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:        5.757ns  (logic 0.341ns (5.924%)  route 5.416ns (94.076%))
  Logic Levels:           3  (BUFG=1 LUT1=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

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------    -------------------------------------------    -------------------
                         (clock clk_fpga_0 fall edge)
                                                   12.500    12.500 f
    PS7_X0Y0             PS7                          0.000    12.500 f  design_1_i/processing_system7_0/inst/PS7_i/FCLKCLK[0]
                         net (fo=1, routed)           1.193    13.693    design_1_i/processing_system7_0/inst/FCLK_CLK_unbuffered[0]
    BUFGCTRL_X0Y20       BUFG (Prop_bufg_I_O)         0.101    13.794 f  design_1_i/processing_system7_0/inst/buffer_fclk_clk_0.FCLK_CLK_0_BUFG/O
                         net (fo=5286, routed)        2.067    15.861    design_1_i/caravel_0/inst/gpio_control_in_1[7]/clock
    SLICE_X51Y96         LUT6 (Prop_lut6_I3_O)        0.124    15.985 f  design_1_i/caravel_0/inst/gpio_control_in_1[7]/mprj_o[15]_INST_0/O
                         net (fo=1, routed)           1.065    17.049    design_1_i/caravel_0/inst/gpio_control_in_1[7]/mprj_o[0]
    SLICE_X30Y96         LUT1 (Prop_lut1_I0_O)        0.116    17.165 f  design_1_i/caravel_0/inst/gpio_control_in_1[7]/mprj_o[0]_hold_fix/O
                         net (fo=1, routed)           1.091    18.257    design_1_i/caravel_ps_0/inst/control_s_axi_U/mprj_o[0]_hold_fix_1_alias_1
    SLICE_X51Y96         FDRE                                       f  design_1_i/caravel_ps_0/inst/control_s_axi_U/int_ps_mprj_out_reg[15]/D
  -------------------    -------------------------------------------    -------------------
```

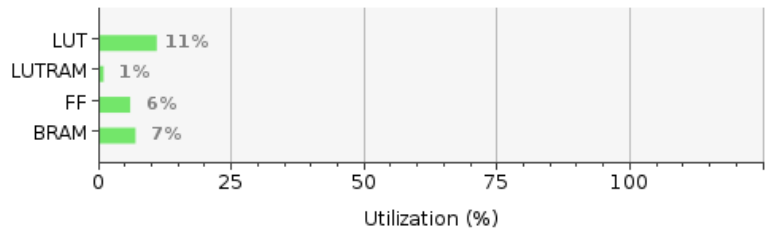
```
  Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
-------------------    -------------------------------------------    -------------------
                       (clock clk_fpga_0 fall edge)
                                                 12.500    12.500 f
  PS7_X0Y0             PS7                          0.000    12.500 f  design_1_i/processing_system7_0/inst/PS7_i/FCLKCLK[0]
                       net (fo=1, routed)           1.193    13.693    design_1_i/processing_system7_0/inst/FCLK_CLK_unbuffered[0]
  BUFGCTRL_X0Y20       BUFG (Prop_bufg_I_O)         0.101    13.794 f  design_1_i/processing_system7_0/inst/buffer_fclk_clk_0.FCLK_CLK_0_BUFG/O
                       net (fo=5286, routed)        2.067    15.861    design_1_i/caravel_0/inst/gpio_control_in_1[7]/clock
  SLICE_X51Y96         LUT6 (Prop_lut6_I3_O)        0.124    15.985 f  design_1_i/caravel_0/inst/gpio_control_in_1[7]/mprj_o[15]_INST_0/O
                       net (fo=1, routed)           1.065    17.049    design_1_i/caravel_0/inst/gpio_control_in_1[7]/mprj_o[0]
  SLICE_X30Y96         LUT1 (Prop_lut1_I0_O)        0.116    17.165 f  design_1_i/caravel_0/inst/gpio_control_in_1[7]/mprj_o[0]_hold_fix/O
                       net (fo=1, routed)           1.091    18.257    design_1_i/caravel_ps_0/inst/control_s_axi_U/mprj_o[0]_hold_fix_1_alias_1
  SLICE_X51Y96         FDRE                                       f  design_1_i/caravel_ps_0/inst/control_s_axi_U/int_ps_mprj_out_reg[15]/D
-------------------    -------------------------------------------    -------------------

                       (clock clk_fpga_0 rise edge)
                                                 25.000    25.000 r
  PS7_X0Y0             PS7                          0.000    25.000 r  design_1_i/processing_system7_0/inst/PS7_i/FCLKCLK[0]
                       net (fo=1, routed)           1.088    26.088    design_1_i/processing_system7_0/inst/FCLK_CLK_unbuffered[0]
  BUFGCTRL_X0Y20       BUFG (Prop_bufg_I_O)         0.091    26.179 r  design_1_i/processing_system7_0/inst/buffer_fclk_clk_0.FCLK_CLK_0_BUFG/O
                       net (fo=5286, routed)        1.466    27.645    design_1_i/caravel_ps_0/inst/control_s_axi_U/ap_clk
  SLICE_X51Y96         FDRE                                       r  design_1_i/caravel_ps_0/inst/control_s_axi_U/int_ps_mprj_out_reg[15]/C
                       clock pessimism              0.000    27.645
                       clock uncertainty           -0.377    27.268
  SLICE_X51Y96         FDRE (Setup_fdre_C_D)       -0.285    26.983    design_1_i/caravel_ps_0/inst/control_s_axi_U/int_ps_mprj_out_reg[15]
-------------------------------------------------------------------
                       required time                         26.983
                       arrival time                         -18.257
-------------------------------------------------------------------
                       slack                                 8.726
```

B. Resource/Utilization report

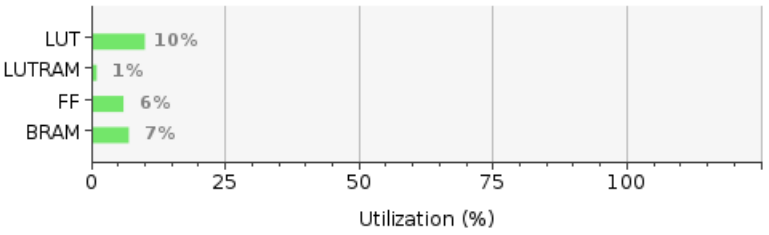
(After synthesis)

Resource	Utilization	Available	Utilization %
LUT	5938	53200	11.16
LUTRAM	241	17400	1.39
FF	6802	106400	6.39
BRAM	10	140	7.14



(After implementation)

Resource	Utilization	Available	Utilization %
LUT	5344	53200	10.05
LUTRAM	188	17400	1.08
FF	6173	106400	5.80
BRAM	10	140	7.14



4. Latency for a character loop back using UART

在 caravel_fpga_uart.ipynb 檔案中，我們除了有實作第 1.點所述之內容外，還在 uart_rxtx()函式中加入了 latency timer 的功能，它會在 ipUart 正要傳輸給 Caravel SoC 前記錄當下的時間，並在接收到回傳的字元之後立刻記錄下當時的時間，將兩個時間相減取出差值，即為此字元透過 UART 傳輸的 latency：

```
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"
    ############################## Added by us ##############################
    latency_timer1_start=time()
    ########################################################################
    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
        ############################## Added by us ##############################
        latency_timer1_end=time()
        ########################################################################
        print(buf, end='')
        ############################## Added by us ##############################
        print('\nNotebook received "'+buf+'", the same string as what we sent, and the latency for this string using UART =
        ########################################################################
```

以上述方式執行的結果如下：

```
Waitting for interrupt

Notebook received "", the same string as what we sent, and the latency for this string using UART =  0.00162506103515625  (s).
h
Notebook received "h", the same string as what we sent, and the latency for this string using UART =  0.007600069046020508  (s).
e
Notebook received "e", the same string as what we sent, and the latency for this string using UART =  0.013179302215576172  (s).
l
Notebook received "l", the same string as what we sent, and the latency for this string using UART =  0.018865346908569336  (s).

Notebook received "", the same string as what we sent, and the latency for this string using UART =  0.020744800567626953  (s).
l
Notebook received "l", the same string as what we sent, and the latency for this string using UART =  0.02974081039428711  (s).

Notebook received "", the same string as what we sent, and the latency for this string using UART =  0.031629323959350586  (s).
o
Notebook received "o", the same string as what we sent, and the latency for this string using UART =  0.03348278999328613  (s).


Notebook received "
", the same string as what we sent, and the latency for this string using UART =  0.043314218521118164  (s).
main(): uart_rx is cancelled now
```

透過此種方式，即可分別將'h'、'e'、'l'、'l'、'o'、'\n'這五個字元從傳出到接收之間的時間差 print 在螢幕上。由於實際上 FPGA implementation 環境的 variation 以及 CPU 在當下的 interrupt 接收時間有變化，都可能導致每筆的接收時間不一致，此算法下 latency for a character loop back using UART 約落在 0.001 s~0.035 s 的區間。

然而，使用上述方式，在過程中可能因為 PS 端計算了與 UART 無關的運算式而導致 latency 測量得不準確，例如其中會包含運算 while 迴圈以及 i=i+1 的運算

等，導致測量出的時間並非只有 UART 的 contribution，而是可能還含有其他運算，因此為了更測量 UART 傳送單一字元的 latency，我們還特別以 uart_rxtx()函式為基底，而新增了 uart_rxtx_for_only_one_char()函式：

```
############################ Added by us ############################
async def uart_rxtx_for_only_one_char():
    # Reset FIFOs, enable interrupts
    ipUart.write(CTRL_REG, 1<<RST_TX | 1<<RST_RX | 1<<INTR_EN)
    print("Waitting for interrupt")
    tx_str = "s"
    buf = ""

    latency_timer_start=time()

    ipUart.write(TX_FIFO, ord(tx_str[0]))
    await intUart.wait()
    buf = chr(ipUart.read(RX_FIFO))

    latency_timer_end=time()


    print("\nNotebook received '"+buf+"', the same character as what we sent, and the latency for one character loop-back using UART = ", latency_timer_end-latency_timer_start," (s).")
#####################################################################
```

這個函式就避免使用到前面所提到的與 UART 運作流程中無關的指令，而是盡量在資料傳輸當下記錄下時間，並且也只傳輸一個字元's'以方便觀察，此時將 task 1 執行的函式改為此：

```
# Python 3.7+
async def async_main():
    task2 = asyncio.create_task(caravel_start())
    ################################ Added by us ################################
    #await asyncio.sleep(1)
    ###task3 = asyncio.create_task(check_matrix_multiplication())
    task1 = asyncio.create_task(uart_rxtx_for_only_one_char())
    #############################################################################
    #####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')
```

執行結果如下：

```
Waitting for interrupt

Notebook received 's', the same character as what we sent, and the latency for one character loop-back using UART =  0.009213447570800781  (s).
```

可發現測量出的 latency for UART 大約為 0.009s，相較於前一個方法的平均值還低很多，這也比較接近 UART 執行 one-character loop-back 實際所需的時間！

5. Suggestion for improving latency for UART loop back

UART 的整個 trigger 以及資料傳遞的流程為：當 UART 的 Rx 接收到足夠的資訊後，就會透過 user_project 與 CPU 之間的 irq 的訊號線告訴 CPU 有 interrupt，然後 CPU 就會被 trap 到 interrupt service routine (ISR)，也就是 isr.c 執行其中的程式碼（在此 lab 中即為將 Rx 接收到的訊號直接丟給 Tx 去輸出，以方便觀察 UART 行為的正確與否），執行完 ISR 後才會回到原本執行的 workload 中繼續執行，而 UART 在此時則將 Tx_buffer 中的值透過 UART 的 pin 腳（mprj_io[6]）來輸出給 testbench。由上述過程中，我們嘗試尋找看看哪些步驟能夠加速，以達到更小的 latency，我們有想到其中幾個方法：

(1). Improve baud rate

依據 workbook 所述，在本次 lab 中並沒有實作出 baud rate 變動的方法，而是使用 default 的 9600，我們認為可以透過 design 提高 baud rate（使傳輸的 clock 速度變快），加速資料傳送速率，使得每秒能連續傳遞的 bit 數增加。

(2). Add FIFO

透過加入 FIFO，可以將資料先暫存起來，當達到一定量時可一次 interrupt CPU，使 CPU 進入 ISR 執行 UART 的操作，這樣一來即可不需要每次在傳送一筆 data 時就要 interrupt 一次，而是整個 FIFO 的資料只需要 interrupt 一次就可全部完成。雖然對於單一 data 的 latency 並無改善，但整體平均下來由於 interrupt 只需呼叫一次，故還是可以減少從 interrupt raise 到 CPU 真正開始執行 ISR 之間的時間，而使平均 latency 下降。

(3). Using user-defined protocol

由於 UART 的 protocol 中規範每個 word data 的傳輸過程必須要有 start bit 以及 stop bit，以確保資料傳送的正確性，但若當我們需要傳遞一個字串時，每個字元都需要一組 start bit 與 stop bit，平均下來會增加 latency，因此我們認為如果可以透過在傳送字串時，先傳 data length 讓對方得知，如此一來在真正傳 data 時就可以省去字串中間 character 的 start bit 及 stop bit，平均下來可以大幅改善 latency 的問題（尤其是對於長字串）。

6. What else do you observe?

在此處我們稍微介紹一下我們的實作結果與發現，以 workbook 第 18 頁的幾點來做說明：

- **Replicate baseline experiment**

(1) Simulation on Matrix Multiplication

執行 run_sim 後結果如下：

```
ubuntu@ubuntu2004:~/lab_6/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

Monitor: Timeout, Test LA (RTL) Failed

ubuntu@ubuntu2004:~/lab_6/lab-wlos_baseline/testbench/counter_la_mm$
```

可發現矩陣乘法結果正確，但最終卻沒有$finish，觀察 counter_la_mm_tb.v：

counter_la_mm.c ● counter_la_mm_tb.v ×

C: > Users > Hong > AppData > Roaming > MobaXterm > slash > RemoteFiles > 722394_2_28 > counter_la_mm_tb.v

```
149             $display ("Monitor: Timeout, Test LA (GL) Failed");
150         `else
151             $display ("Monitor: Timeout, Test LA (RTL) Failed");
152         `endif
153         $display("%c[0m",27);
154         $finish;
155     end
156
157     initial begin
158         wait(checkbits == 16'hAB40);
159         $display("LA Test 1 started");
160         //wait(checkbits == 16'hAB41);
161
162         wait(checkbits == 16'h003E);
163         $display("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
164         wait(checkbits == 16'h0044);
165         $display("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
166         wait(checkbits == 16'h004A);
167         $display("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
168         wait(checkbits == 16'h0050);
169         $display("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
170
171         wait(checkbits == 16'hAB51);
172         $display("LA Test 2 passed");
173         #10000;
174         $finish;
175     end
```

發現$finish 的條件為 testbench 偵測到 checkbits（即 reg_mprj_data1）的值為 0xAB51，因此我們到 counter_la_mm.c：

counter_la_mm.c × counter_la_mm_tb.v

C: > Users > Hong > AppData > Roaming > MobaXterm > slash > RemoteFiles > 722394_2_27 > C counter_la_mm.c

```
109     reg_la1_oenb = reg_la1_iena = 0xFFFFFFFF;    // [63:32]
110     reg_la2_oenb = reg_la2_iena = 0x00000000;    // [95:64]
111     reg_la3_oenb = reg_la3_iena = 0x00000000;    // [127:96]
112
113     // Flag start of the test
114     reg_mprj_datal = 0xAB400000;
115
116     // Set Counter value to zero through LA probes [63:32]
117     reg_la1_data = 0x00000000;
118
119     // Configure LA probes from [63:32] as inputs to disable counter write
120     reg_la1_oenb = reg_la1_iena = 0x00000000;
121
122 /*
123     while (1) {
124         if (reg_la0_data_in > 0x1F4) {
125             reg_mprj_datal = 0xAB410000;
126             break;
127         }
128     }
129 */
130     int *tmp = matmul();
131     reg_mprj_datal = *tmp << 16;
132     reg_mprj_datal = *(tmp+1) << 16;
133     reg_mprj_datal = *(tmp+2) << 16;
134     reg_mprj_datal = *(tmp+3) << 16;
135
136     //print("\n");
137     //print("Monitor: Test 1 Passed\n\n");  // Makes simulation very long!
138     reg_mprj_datal = *(tmp+9) << 16;
139
140     // Added by us
141     reg_mprj_datal = 0xAB510000;
142 }
```

在其中加入

reg_mprj_data1 = 0xAB510000;

重跑模擬後即可得到

```
ubuntu@ubuntu2004:~/lab_6/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
ubuntu@ubuntu2004:~/lab_6/lab-wlos_baseline/testbench/counter_la_mm$
```

(2) Simulation on Quick Sort

執行 run_sim 後結果如下：

```
ubuntu@ubuntu2004:~/lab_6/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:~/lab_6/lab-wlos_baseline/testbench/counter_la_qs$
```

(3) Simulation on FIR

執行 run_sim 後結果如下：

```
ubuntu@ubuntu2004:~/lab_6/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
LA Test 2 passed
ubuntu@ubuntu2004:~/lab_6/lab-wlos_baseline/testbench/counter_la_fir$
```

(4) Simulation on UART

執行 run_sim 後結果如下：

```
ubuntu@ubuntu2004:~/lab_6/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

ubuntu@ubuntu2004:~/lab_6/lab-wlos_baseline/testbench/uart$
```

會 time out 主要原因是因為需要等到 testbench 的 Rx 接收到回傳的值，但我們並不曉得何時會送來，因此無法事先預測$finish 的時間點。途中可知 Rx 確實有收到與 Tx 相同的回傳值，因此 UART 運作正常。

(5) UART FPGA

首先先合成出.bit 檔以及.hwh 檔，步驟包含 synthesis 及 implementation。Synthesis 的結果會發現有 hold time violation，我們透過討論區[1]中同學分享的解法，將-job 的位置由 3 改為 2（因為我們一開始在設置時只有 allocate 最多 2 個 jobs 給它），並將

set_property -name "strategy" -value "Vivado Implementation Defaults" -objects $obj

修改為

set_property -name "strategy" -value "Performance_NetDelay_high" -objects $obj

[1] https://github.com/bol-edu/HLS-SOC-Discussions/discussions/158

如此一來雖然 synthesis 仍有 hold time violation（因為上述為 implementation 的 strategy），但 implementation 後就沒有 hold time violation 了（猜測可能是因為加入考量走線，loading 變大→delay 變大而導致）。

接著放到 onlineFPGA 中結果如下：

```
In [13]: asyncio.run(async_main())

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

In [14]: 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
```

- **Firmware code integrates Matrix Multiplication, Quick Sort, FIR and UART**

如 counter_la_uart.c 中所示（即下圖）

```
#ifdef USER_PROJ_IRQ0_EN
    // unmask USER_IRQ_0_INTERRUPT
    mask = irq_getmask();
    mask |= 1 << USER_IRQ_0_INTERRUPT; // USER_IRQ_0_INTERRUPT = 2
    irq_setmask(mask);
    // enable user_irq_0_ev_enable
    user_irq_0_ev_enable_write(1);
#endif
```

```
////////////////////////////// matrix multiplication (counter_la_mm.c) //////////////////////////////
int* tmp = matmul();
reg_mprj_datal = *tmp << 16;
reg_mprj_datal = *(tmp+1) << 16;
reg_mprj_datal = *(tmp+2) << 16;
reg_mprj_datal = *(tmp+3) << 16;

reg_mprj_datal = *(tmp+9) << 16;

reg_mprj_datal = 0xAB510000;

////////////////////////////// Q sort (counter_la_qs.c) //////////////////////////////
tmp = qsort();
reg_mprj_datal = *tmp << 16;
reg_mprj_datal = *(tmp+1) << 16;
reg_mprj_datal = *(tmp+2) << 16;
reg_mprj_datal = *(tmp+3) << 16;
reg_mprj_datal = *(tmp+4) << 16;
reg_mprj_datal = *(tmp+5) << 16;
reg_mprj_datal = *(tmp+6) << 16;
reg_mprj_datal = *(tmp+7) << 16;
reg_mprj_datal = *(tmp+8) << 16;
reg_mprj_datal = *(tmp+9) << 16;

reg_mprj_datal = 0xAB520000;

////////////////////////////// FIR (counter_la_fir.c) //////////////////////////////
tmp = fir();
reg_mprj_datal = *tmp << 16;
reg_mprj_datal = *(tmp+1) << 16;
reg_mprj_datal = *(tmp+2) << 16;
reg_mprj_datal = *(tmp+3) << 16;
reg_mprj_datal = *(tmp+4) << 16;
reg_mprj_datal = *(tmp+5) << 16;
reg_mprj_datal = *(tmp+6) << 16;
reg_mprj_datal = *(tmp+7) << 16;
reg_mprj_datal = *(tmp+8) << 16;
reg_mprj_datal = *(tmp+9) << 16;
reg_mprj_datal = *(tmp+10) << 16;

reg_mprj_datal = 0xAB600000;
```

將 Rx 提早一點開放，可以提早得到回傳值。並且將 3 個 workload 都 integrate 在一起，之間以一些 reg_mprj_data1 的 mark 作為區分，以方便 testbench 得知完成哪些部份了。

- **Hardware integrates exmem-fir with UART design in user project area, like Lab4-2.**

  如/lab_6/lab-wlos_baseline/rtl/user/uart.v 所示，是由 lab4-2 的 WB_decoder 依照類似作法而得。

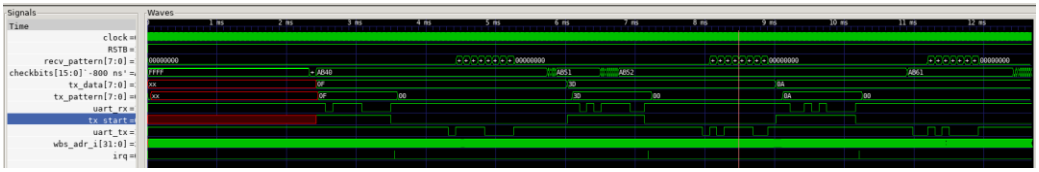
- **Simulation**

  Testbench（counter_la_uart_tb.v）會去偵測 reg_mprj_data1 的 mark 並 check 計算結果的正確性，另外，收到 UART 訊號也會將其 print 在螢幕上，執行結果如下：

```
ubuntu@ubuntu2004:~/lab_6/lab-wlos_baseline/testbench/lab6_integration$ source run_s
im
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
Info: Start UART test 1...
Info: Start matrix multiplication test...
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
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
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed,
0x003e, which is    62 in decimal
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed,
0x0044, which is    68 in decimal
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed,
0x004a, which is    74 in decimal
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed,
0x0050, which is    80 in decimal
Success: Matrix multiplication test passed ♪
Info: Start Q sort test...
Info: Start UART test 2...
tx data bit index 0: 1
tx data bit index 1: 0
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0
x0028, which is    40 in decimal
tx data bit index 2: 1
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0
x037d, which is   893 in decimal
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0
x09ed, which is  2541 in decimal
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0
x0a6d, which is  2669 in decimal
```

```
tx data bit index 3: 1
tx data bit index 4: 1
Success: Q sort test passed ♪
Info: Start FIR test ...
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
Info: Start the last UART test ('\n') ...
tx data bit index 0: 0
tx data bit index 1: 1
tx data bit index 2: 0
tx data bit index 3: 1
tx data bit index 4: 0
tx data bit index 5: 0
tx data bit index 6: 0
tx data bit index 7: 0
tx (last) complete
Success: UART test passed ♪
rx data bit index 0: 0
rx data bit index 1: 1
rx data bit index 2: 0
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  10
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0
44a, which is  1098 in decimal
Success: FIR test passed ♪
LA Test 1 passed
-----------------------------------------------------
----------Congratulations! Pass all tests------------
```

有在 workload 執行當中的不同時間點下測試 UART 功能皆正常！
相關波形如下：



- **Run on FPGA**

  在第 1.題中有詳細的介紹~