

[SoC Lab] Lab6

tags: SoC Lab, SOC Design

Team 13

Student ID	Name
311551095	林聖博
312551174	張祐誠

附上此篇Hackmd Link: <https://hackmd.io/@Sheng08/SJyEtQy8p>

- [SoC Lab] Lab6
 - Lab 6 Spec
 - Simulation
 - Three Functions Execute in User Project
 - UART FPGA
 - Integration
 - Block design
 - Timing report/ resource report after synthesis
 - Latency for a character loop back using UART
 - Only use UART
 - Use integrate
 - How do you verify your answer from notebook
 - Integrate
 - Screenshot of Execution result on workload
 - Suggestion for improving latency for UART loop back
 - What else do you observe
 - Github link

Lab 6 Spec

- https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/lab-wlos_baseline

Simulation

Three Functions Execute in User Project

- **FIR**

$[0, -10, -9, 23, 56, 63, 56, 23, -9, -10, 0]$



$[0, -10, -29, -25, 35, 158, 337, 539, 732, 915, 1098]$

- Simulation

```
ubuntu@ubuntu2004:~/lab/lab6/SoC-Lab-lab6_new/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 (y[0]) passed, 0x0000
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[1]) passed, 0xffff6
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[2]) passed, 0xffe3
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[3]) passed, 0xffe7
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[4]) passed, 0x0023
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[5]) passed, 0x009e
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[6]) passed, 0x0151
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[7]) passed, 0x021b
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[8]) passed, 0x02dc
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[9]) passed, 0x0393
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[10]) passed, 0x044a
LA Test 2 passed
```

- Verify

- Firewall

```
int* tmp = fir();
reg_mprj_data1 = *tmp << 16;
reg_mprj_data1 = *(tmp+1) << 16;
reg_mprj_data1 = *(tmp+2) << 16;
reg_mprj_data1 = *(tmp+3) << 16;
reg_mprj_data1 = *(tmp+4) << 16;
reg_mprj_data1 = *(tmp+5) << 16;
reg_mprj_data1 = *(tmp+6) << 16;
reg_mprj_data1 = *(tmp+7) << 16;
reg_mprj_data1 = *(tmp+8) << 16;
reg_mprj_data1 = *(tmp+9) << 16;
reg_mprj_data1 = *(tmp+10) << 16;
```

counter_la_fir.c

- Testbench

- 利用 testbench 中的 **checkbits** 來驗證 FIR 計算結果是否正確
 - 輸出(與預期結果相符):
 - Hex: {0x0000, 0xFFFF6, 0xFFE3, 0xFFE7, 0x0023, 0x009E, 0x0151, 0x021B, 0x02DC, 0x0393, 0x044A}
 - Dec: {0, -10, -29, -25, 35, 158, 337, 539, 732, 915, 1098}

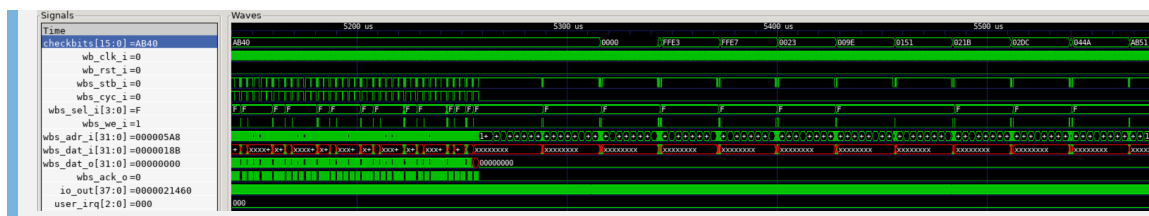
counter_la_fir_tb.v

```

wait(checkbits == 16'd0);
$display("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[0]) passed, 0x%x", checkbits);
wait(checkbits == 16'hFFF6); // -10
$display("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[1]) passed, 0x%x", checkbits);
wait(checkbits == 16'hFFE3); // -29
$display("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[2]) passed, 0x%x", checkbits);
wait(checkbits == 16'hFFE7); // -25
$display("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[3]) passed, 0x%x", checkbits);
wait(checkbits == 16'd35);
$display("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[4]) passed, 0x%x", checkbits);
wait(checkbits == 16'd158);
$display("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[5]) passed, 0x%x", checkbits);
wait(checkbits == 16'd337);
$display("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[6]) passed, 0x%x", checkbits);
wait(checkbits == 16'd539);
$display("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[7]) passed, 0x%x", checkbits);
wait(checkbits == 16'd732);
$display("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[8]) passed, 0x%x", checkbits);
wait(checkbits == 16'd915);
$display("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[9]) passed, 0x%x", checkbits);
wait(checkbits == 16'd1098);
$display("Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[10]) passed, 0x%x", checkbits);

```

◦ Waveform



• Matrix Multiplication

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

◦ Simulation

```

ubuntu@ubuntu2004:~/lab/lab6/Soc-Lab-lab6_new/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

```

◦ Verify

■ Firewall

```

int *tmp = matmul();
reg_mprj_data1 = *tmp << 16;
reg_mprj_data1 = *(tmp+1) << 16;
reg_mprj_data1 = *(tmp+2) << 16;
reg_mprj_data1 = *(tmp+3) << 16;

```

counter_la_mm.c

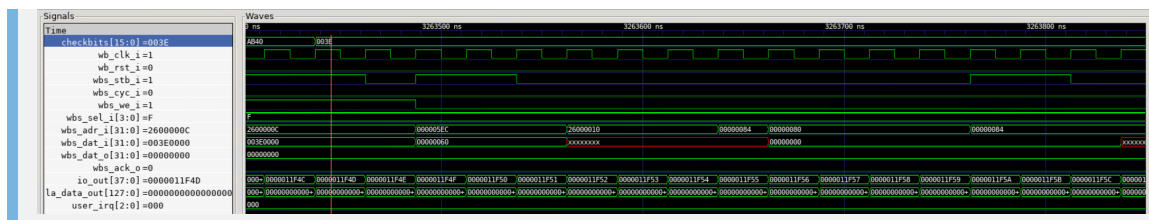
- Testbench

- 此部分驗證第一列(row)數值正確性，並由 Simulation 結果可以確認符合預期結果
 - Hex: {0x003E, 0x0044, 0x004A, 0x0050}
 - Dec: {62, 68, 74, 80}

counter_la_mm_tb.v

```
wait(checkbits == 16'h003E);
$display("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
wait(checkbits == 16'h0044);
$display("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
wait(checkbits == 16'h004A);
$display("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
wait(checkbits == 16'h0050);
$display("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
```

- Waveform



- Quick Sort

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



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

- Simulation

```
ubuntu@ubuntu2004:~/lab/lab6/SoC-Lab-lab6_new/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 qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0028
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x037d
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x09ed
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0a6d
LA Test 2 passed
```

- Verify

- Fireware

```
int* tmp = qsort();
reg_mprj_data1 = *tmp << 16;
reg_mprj_data1 = *(tmp+1) << 16;
reg_mprj_data1 = *(tmp+2) << 16;
reg_mprj_data1 = *(tmp+3) << 16;
```

counter_la_qs.c

- Testbench

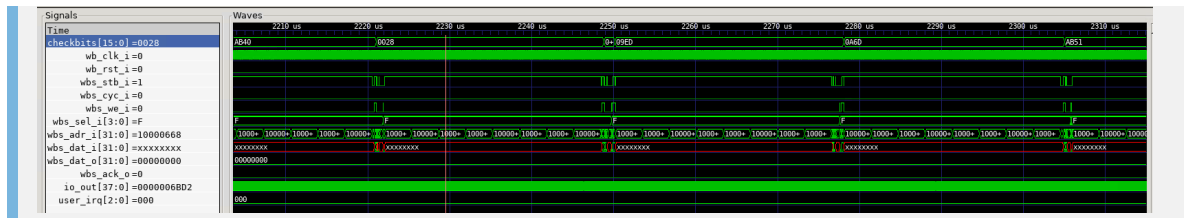
- 此部分驗證前四個數值正確性，並由 Simulation 結果可以確認符合預期結果
 - Hex: {0x0028, 0x037D, 0x09ED, 0x0A6D}

- Dec: {40, 893, 2541, 2669}

counter_la_qs_tb.v

```
wait(checkbits == 16'd40);
$display("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
wait(checkbits == 16'd893);
$display("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
wait(checkbits == 16'd2541);
$display("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
wait(checkbits == 16'd2669);
$display("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
```

- Waveform



UART FPGA

- Firmware
 - In order to avoid the LA Timeout, add the waiting statement.
 - First wait the tx is not empty and then wait the rx has data
 - Finally, set a delay while loop to wait for rx finish

```
#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
while(reg_uart_stat >> 2 & 0x1 == 0)
;;
while(reg_uart_stat & 0x1 == 1)
;;
int j=0;
while(j < 1000) ++j;
reg_mprj_dat1 = 0xAB510000;
```

- Simulation

```
ubuntu@ubuntu2004:~/caravel-soc_fpga-lab/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
LA Test 1 passed
```

- Verify

- Fireware

counter_la_uart.c

```
156 #ifdef USER_PROJ_IRQ0_EN
157     // unmask USER_IRQ_0_INTERRUPT
158     mask = irq_getmask();
159     mask |= 1 << USER_IRQ_0_INTERRUPT; // USER_IRQ_0_INTERRUPT
160     irq_setmask(mask);
161     // enable user_irq_0_ev_enable
162     user_irq_0_ev_enable_write(1);
163 #endif
```

- Testbench

- 由 `tbuart.v` 確認 rx 與 tx 相關狀態與資料

tbuart.v

```
100     always@(posedge clk)begin
101         case(recv_state)
102             R_WAIT: recv_pattern <= 0;
103             R_GET_DATA: begin
104                 recv_pattern <= {ser_rx, recv_pattern[7:1]};
105                 $display("rx data bit index %d: %b", rx_index, ser_rx);
106             end
107             default: recv_pattern <= 0;
108         endcase
109     end
```

```

167     always@(posedge clk)begin
168         case(tr_state)
169             T_WAIT: ser_tx <= 1;
170             T_START_BIT: ser_tx <= 0;
171             T_SEND_DATA:begin
172                 ser_tx <= tx_pattern[tx_index];
173                 $display("tx data bit index %d: %b", tx_index, tx_pattern[tx_index]);
174             end
175             T_STOP_BIT: ser_tx <= 1;
176             T_CLEAR: ser_tx <= 1;
177             default: ser_tx <= 1;
178         endcase
179     end

```

◦ Waveform



• Synthesis & Implementation

◦ Modification

- May occur the timing violation durning implementation
- Modify the `phy_opt_design` and re-implement it
- **Reference:** <https://github.com/bol-edu/HLS-SOC-Discussions/discussions/158#discussioncomment-7827859>

◦ Timing report

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 9.198 ns	Worst Hold Slack (WHS): 0.010 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.

◦ Utilization

Reports	Design Runs	DRC	Methodology	Power	Timing	Utilization						
Q	≡	≡	%	Hierarchy								
Name	^	1	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	Block RAM Tile (140)	Bonded IOPADs (130)	BUFGCTRL (32)
design_1_wrapper	✓	N	5332	6159	170	47	2354	5144	188	6	130	5
design_1_i (design_1)	>	I	5332	6159	170	47	2354	5144	188	6	0	5

• Online FPGA

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

```

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

```

```

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

```

Integration

- Firmware
 - Enable the interrupt earlier and add some delay after the enable.

```

int delay = 0;
// ===== matmul =====

int *tmp_mat = matmul();
while(delay < 500){ delay++; }
delay = 0;
reg_mprj_datal = *tmp_mat << 16;
reg_mprj_datal = *(tmp_mat+1) << 16;
reg_mprj_datal = *(tmp_mat+2) << 16;

#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

while(delay < 1000) ++delay;
delay = 0;
reg_mprj_datal = *(tmp_mat+3) << 16;
reg_mprj_datal = 0xAB600000;

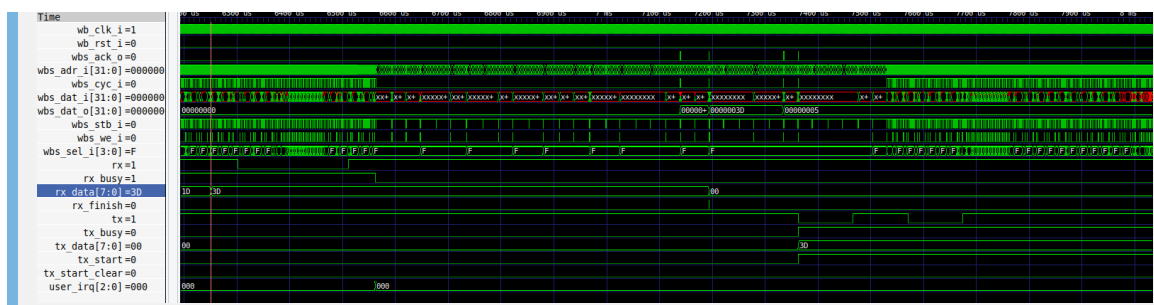
// ===== fir =====
int* tmp_fir = fir();
reg_mprj_datal = *(tmp_fir+7) << 16;
reg_mprj_datal = *(tmp_fir+8) << 16;
reg_mprj_datal = *(tmp_fir+9) << 16;
reg_mprj_datal = *(tmp_fir+10) << 16;
reg_mprj_datal = 0xAB610000;

```


- Simulation

```
ubuntu@ubuntu2004:~/caravel-soc_fpga-lab/lab-wlos_baseline/testbench/all$ source run_sim
Reading counter_la_all.hex
counter_la_all.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_all.vcd opened for output.
uart_task delay      5379748 clocks
User function starts at      2447338
LA Test 1 started
===== Test Matmul =====
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
uart_task starts at      5379748
tx data bit index 0: 1
tx data bit index 1: 0
tx data bit index 2: 1
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0050
===== Matmul Fisish =====
===== Test FIR =====
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
uart_task ends at      6510375
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
received word 61
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[7]) passed, 0x021b
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[8]) passed, 0x02dc
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[9]) passed, 0x0393
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[10]) passed, 0x044a
===== FIR Fisish =====
===== Test Qsort =====
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0028
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0037d
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x009ed
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0a6d
===== Qsort Fisish =====
LA Test 1 passed
```

- Waveform










- 本次 Lab 目標使用 axi-uartlite 跟 Interrupt Control IP



Design Timing Summary

All user specified timing constraints are met.

- Utilization (Resource report)

ports	Design Runs	DRC	Methodology	Power	Timing	Utilization	x					
				Hierarchy								
Name			Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	Block RAM Tile (140)	Bonded IOPADs (130)	BUFGCTRL (32)
▼	 design_1_wrapper		5347	6175	170	47	2255	5159	188	7	130	5
>	 design_1_i (design_1)		5347	6175	170	47	2255	5159	188	7	0	5

Latency for a character loop back using UART

Only use UART

- Python Jupyter (PYNQ)

```

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"
    ipUart.write(TX_FIFO, ord(tx_str[0]))
    i = 1
    while(True):
        await intUart.wait()
        buf = ""
        # Read FIFO until valid bit is clear
        start_time = datetime.now()
        while ((ipUart.read(STAT_REG) & (1<<RX_VALID))):
            buf += chr(ipUart.read(RX_FIFO))
        print(f"get char {buf} from rx, interval time =", datetime.now().microsecond-start_time.microsecond, "ms")
        if i<len(tx_str):
            ipUart.write(TX_FIFO, ord(tx_str[i]))
            i=i+1
        print(buf, end='')
    #

```

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

```

Start Caravel Soc
Waiting for interrupt
get char h from rx, interval time = 107 ms
get char e from rx, interval time = 123 ms
get char l from rx, interval time = 115 ms
get char l from rx, interval time = 115 ms
get char o from rx, interval time = 107 ms
get char
from rx, interval time = 101 ms
main(): uart_rx is cancelled now

```

- Latency 計算

使用 UART 傳送 hello\n 字串

1. start to h: 107 ms
2. h to e: 123 ms
3. e to l: 115 ms
4. l to l: 115 ms
5. l to o: 107 ms

Use integrate

- Python Jupyter (PYNQ)

```

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"
    ipUart.write(TX_FIFO, ord(tx_str[0]))
    i = 1
    while(True):
        await intUart.wait()
        buf = ""
        # Read FIFO until valid bit is clear
        start = datetime.now()
        while ((ipUart.read(STAT_REG) & (1<<RX_VALID))):
            buf += chr(ipUart.read(RX_FIFO))
            print(f"get char {buf} from rx, interval time =", datetime.now().microsecond - start.microsecond, "ms")
            if i<len(tx_str):
                ipUart.write(TX_FIFO, ord(tx_str[i]))
                i=i+1
        print(buf, end='')

```

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

```

Start Caravel Soc
Waiting for interrupt
get char h from rx, interval time = 147 ms
hget char e from rx, interval time = 985 ms
eget char l from rx, interval time = 111 ms
lget char l from rx, interval time = 106 ms
lget char o from rx, interval time = 99 ms
oget char
from rx, interval time = 99 ms

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

```

- Latency 計算

使用 UART 傳送 `hello\n` 字串

1. start to `h`: 147 ms
2. `h` to `e`: 985 ms
3. `e` to `l`: 111 ms
4. `l` to `l`: 106 ms
5. `l` to `l`: 99 ms

How do you verify your answer from notebook

Integrate

- Python Jupyter (PYNQ)

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

Start Caravel Soc
Waiting for interrupt
get char h from rx, interval time = 147 ms
hget char e from rx, interval time = 985 ms
eget char l from rx, interval time = 111 ms
lget char l from rx, interval time = 106 ms
lget char o from rx, interval time = 99 ms
oget char
from rx, interval time = 99 ms

main(): uart_rx is cancelled now
```

Screenshot of Execution result on workload

當 **Reset** 的訊號為 1 時，Caravel 開始(重新)執行 Firmware Code (**caravel_start**)。

```
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}
```

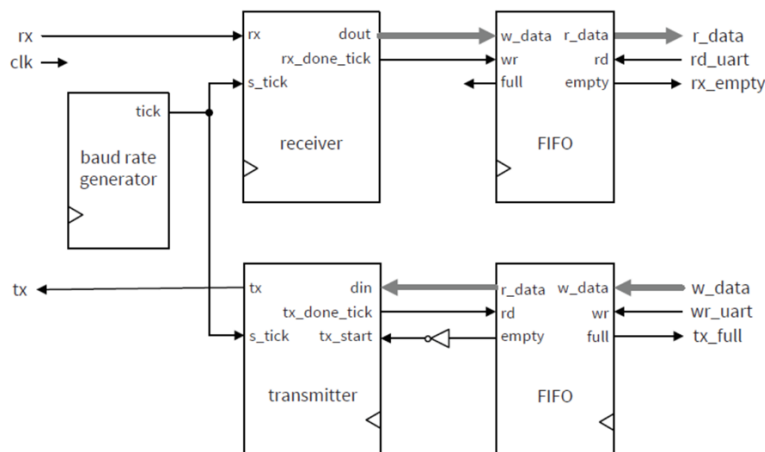
執行 **.hex** 後，查看 **mprj_io([31:16])** 的 Output，**0x1c** 腳位值為 **0xAB51**。

```
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
```

Suggestion for improving latency for UART loop back

1. 結合 FIFO (First In, First Out) Buffer



- 當利用 FIFO 方法，系統能夠在累積一定數量的資料後才觸發單次中斷，從而一次性處理這些資料。此方式增強接收效能，避免了因頻繁的中斷而導致的效率低下，尤其特別適合於大量資料的傳輸情境
 - 在 UART 傳輸，透過 FIFO 可以 buffer 接收到的資料，而不是立即處理每個 Byte
- 成效:
 - 可降低中斷的頻率，因為不需要每收到一個 Byte 就觸發一次中斷。當 FIFO 中的資料達到一定量時，才處理這些資料

➤ 減少中斷次數：不使用 FIFO 的情況下，每接收到一個字元(Byte)可能產生一次中斷。使用 64-Byte FIFO，可以設定中斷觸發條件

- 例如: 當 FIFO 達到一定 fill level (如 32 Bytes (half full)) 時才產生中斷

➤ 預估中斷頻率(Interrupt Frequency)：假設每秒接收 1000 Bytes 下，使用 FIFO 的中斷頻率大約降至每秒約 31 次，相比原來的 1000 次中斷，減少約 97%

- 如果每 32 bytes (Half full) 觸發一次中斷，則每秒中斷的次數為:

$$\text{Interrupts per second} = \frac{\text{Number of bytes per interrupt}}{\text{bytes per second}} = \frac{32_{\text{bytes/interrupt}}}{1000_{\text{bytes/sec}}} \approx 31.25 \text{ times/sec}$$

- 實作構想:
 - 定義 Configuration Registers 以進行 UART 和 FIFO 之間的 handshake，已用於處理與 Firmware 的交互

2. 使用 DMA 進行接收

- 透過使用 DMA (Direct Memory Access，直接記憶體存取) 來執行 UART 的資料傳輸，可以實現在不占用 CPU 資源的情況下，直接在記憶體和外圍設備間進行資料轉移。這種方法有效減少了 CPU 的負擔，從而提升了數據處理的速度和降低了延遲時間

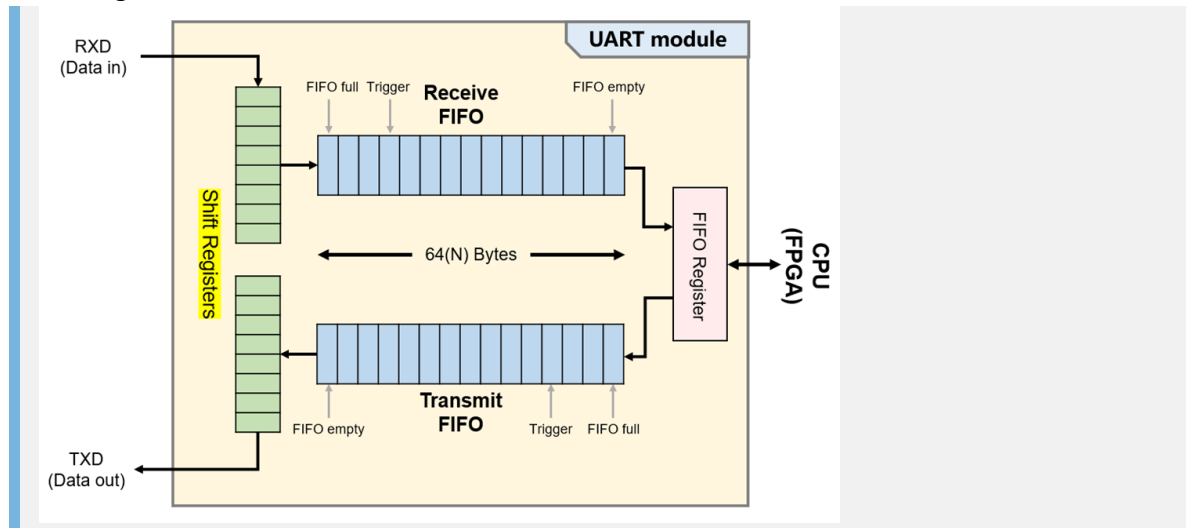
3. 可以動態提高 Baud Rate 來增加資料傳輸

- 根據實際的通訊品質和資料流動情況，動態地調節 Baud Rate
- 能夠確保資料的完整無誤情況下，同時也能加快資料傳輸的速率

經過本次 Lab6 實驗過程，目標於期末 Project 針對 UART loop back 進行優化與設計，並期望能透過 FIFO 方式提升 UART 傳輸效率，減少 UART 時常中斷導致 CPU overhead 過高的問題。

- 概述:

◦ Block Diagram



◦ Handshake

1. **Buffer Full** : 當 Buffer 填滿時，UART FIFO 會通知 CPU 處理所有 Buffer (緩衝區)中的資料，然後清空 Buffer，以便接收後續的資料
2. **Special Character Received** : 若在資料中接收到特定的結束字元，如: `\n`，UART FIFO 會通知 CPU 處理到該特殊字元為止的所有資料，然後調整 Buffer 的剩餘部分
3. **Timeout** : 若在超過設定的時間沒有接收到新資料，UART FIFO 會通知 CPU 處理所有 Buffer 的資料，並清空 Buffer

- Proposal 可參考以下連結:

https://drive.google.com/drive/folders/1MXkaTi6Wh6umBkulmf_QCtx0RjloPC1k

What else do you observe

這次 Lab 6 主要著重於結合先前完成的 Modules 並且實現 UART 傳輸功能與驗證 Fireware 功能的正確性以及整合這些功能到 FPGA 上。與 Lab 5 相比，Lab 6 新增兩個重要的 IP：`axi-uartlite` 用於資料傳輸與 `Interrupt-Control` 用於產生中斷訊號。

實驗流程：當 PS (Processing System) 端向 Caravel 發送資料時，資料會通過 `AXI-interconnect` 傳送到 `axi-uartlite`，再由 Caravel 接收。此過程涉及到從 `axi-uartlite-RX` 到 `caravel_uart-TX` 的資料傳輸。當 Caravel 需要向 PS 端傳送資料時，則是通過 `caravel_uart-RX` 到 `axi-uartlite-TX` 的路徑進行。一旦 `axi-uartlite` 收到資料，則會觸發中斷，並通過 `Interrupt-control` 把資料傳送給 PS 端。

在 Software 與 Fireware 方面，當 Caravel 接收到中斷信號時，會更新 CSR 中的 `mask`，並啟用中斷。隨後會執行 `isr.c`，即進行 UART 資料的讀寫。硬體方面的工作原理類似，其中涉及到從 `mprj_io[5]` 到 `uart_RX` 的資料流，以及從 `uart_ctrl` 到 `mprj_io[6]` 的反向資料流。

Lab 6 還包括使用 Testbench Simulation 和 Python 程式碼來進行功能驗證。通過 `read_romcode` IP，將 `.hex` 文件從 PS 端的 DDR 傳輸到硬體 BRAM。`ResetControl` IP 負責管理 PS 端和 Caravel 之間的 GPIO 和 reset 訊號，而 `caravel_ps` IP 則用於 PS 端和 Caravel 的 `mprj_io` 通信。

最後總結這次 Lab 6 通過新增的 `axi-uartlite` 和 `iInterrupt-Control` IP，以及一系列的軟硬體互動，實現了 UART 傳輸和中斷處理的功能，並透過綜合的測試方法確保了系統的穩定和功能的正確性。

Github link

- <https://github.com/Sheng08/SoC-Lab-lab6>
-



補充