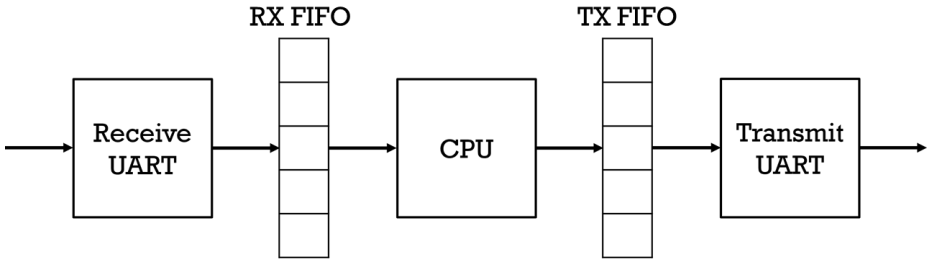
SOC Final Project
UART FIFO Optimization

14 組
111061560 吳俊鋌
111061534 陳　翀

一、Block Diagram:



在 UART 的 receive 和 transmit 端各加一個 FIFO，減少 UART 傳送多筆資料時，須向 CPU interrupt 的次數，增加傳輸效率。

二、Firmware Design:

1.ISR.C modification

在 CPU 接收到 interrupt 訊號時，由原先一次讀寫一筆 UART 的資料，改為一次連續讀寫 9 筆，直到 RX_FIFO 為空，達到 CPU 連續傳輸的效果，此外 CPU 在傳輸時，UART 也可以同時進行 receive 和 transmit 的動作，使 CPU 與 UART 能並行工作，減少 UART 停頓的時間。

```
23  void isr(void)
24  {
25      int i;
26
27  #ifndef USER_PROJ_IRQ0_EN
28
29      irq_setmask(0);
30
31
32  #else
33      uint32_t irqs = irq_pending() & irq_getmask();
34      int buf;
35
36      if ( irqs & (1 << USER_IRQ_0_INTERRUPT)) {
37          user_irq_0_ev_pending_write(1); //Clear Interrupt Pending Event
38          for(i=0;i<10;i++){
39              buf = uart_read();
40              uart_write(buf);
41          }
42
43      }
44  #endif
45
46      return;
47
48  }
```

2.UART.C Optimization

我們發現 CPU 在讀取時，會對 reg_uart_stat 訊號進行兩次取值與一些額外的運算，因此我們將其改為把 reg_uart_stat 讀取到的訊號先存進我們自己設的變數 state 中，再以 state 參數進行計算，此外原先需右移四和五的動

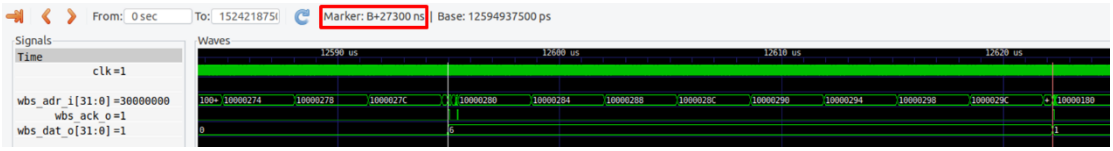
作也改為直接與 0x0030 進行 and，減少 CPU 傳輸所需讀寫與運算的次數，增加傳輸效能，下圖為 UART.C 優化結果比較。

```
int __attribute__ ( ( section ( ".mprj" ) ) ) uart_read()
{
    int num;
    if((((reg_uart_stat>>5) | 0) == 0) && (((reg_uart_stat>>4) | 0) == 0)){
        for(int i = 0; i < 1; i++)
            asm volatile ("nop");

        num = reg_rx_data;
    }

    return num;
}
```

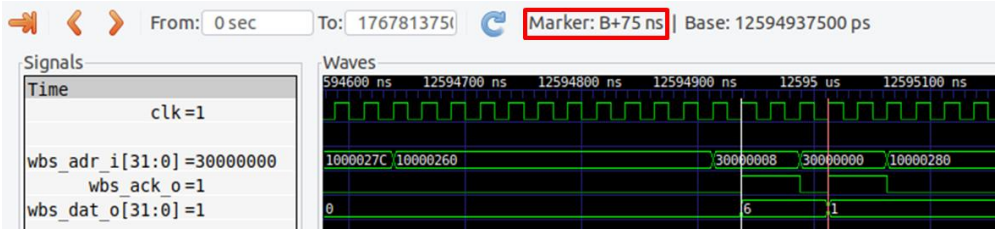
圖一、UART.c 原始程式碼

```
42  int __attribute__ ( ( section ( ".mprj" ) ) ) uart_read()
43  {
44      int num;
45      int state;
46      state = reg_uart_stat;
47      if( (state & 0x0030) | 0 == 0 ){
48          for(int i = 0; i < 1; i++)
49              asm volatile ("nop");
50
51          num = reg_rx_data;
52      }
53
54      return num;
55  }
```

圖一、UART.c 優化後程式碼



圖二、CPU 接收 reg_uart_stat 至 reg_rx_data 原先時間差



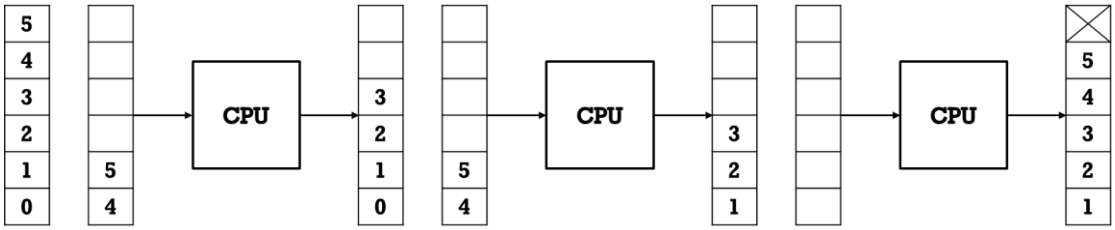
圖二、CPU 接收 reg_uart_stat 至 reg_rx_data 優化後時間差

由上兩圖可知，優化計算 reg_uart_stat 訊號需花費 27300ns，優化後可減為只需 75ns 即可完成所有 reg_uart_stat 的計算，傳送每筆資料可以減少 27225ns 的時間，大幅減少 CPU 傳輸所需的延遲。

三、Hardware Design:

FIFO Depth

因為 CPU 取值與 UART 傳輸的時間有差距，且 CPU 快很多，因此如果能找到兩者時間差之整數倍，就能夠使得 CPU 取值並存進 FIFO 後 UART 傳輸一筆資料，接著 CPU 再次執行取值之功能，這樣可以使得 TX FIFO 的長度能夠減少。



經實驗後發現，CPU 傳輸速度約為 UART 的 9 點多倍，因此我們將 RX FIFO 長度設為 10，TX FIFO 長度設為 9，使 CPU 準備將第十筆資料傳進 TX FIFO 時，UART 也正好將一筆資料由 TX FIFO 傳出，使 TX FIFO 達到可以減少一個 memory 的效果。

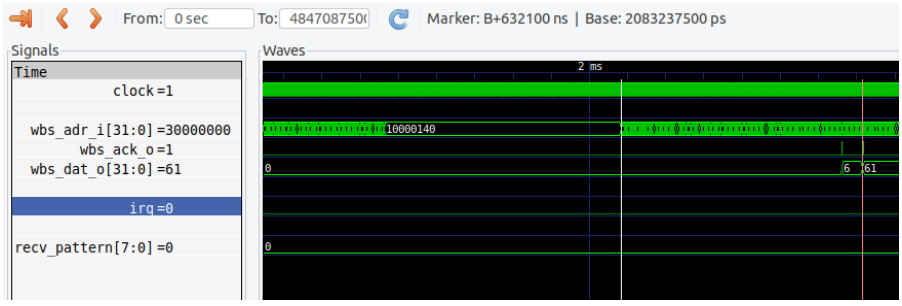
四、Experimental Result:

CPU Latency

在加了 FIFO 與優化 Firmware 後，CPU 傳輸每筆資料可以由原先的 25284 cycles，減為 6002 cycles，達到超過四倍的效能提升。

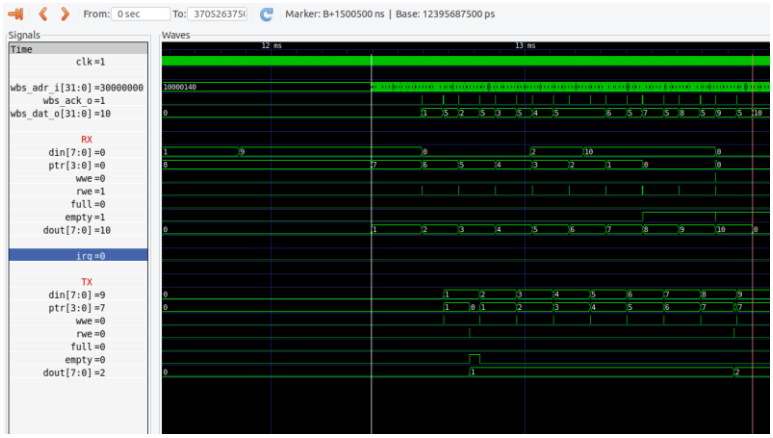
Baseline :

CPU time : ISR time + read/write time = 25284 cycles per data



Our work :

CPU time : ISR time + read/write time*10 = 60020 cycles = 6002 cycles per data.

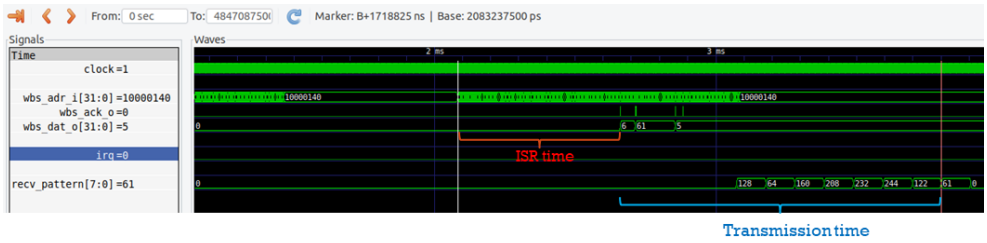


Total Latency

CPU 加上 UART 後的傳輸時間也由原先的每筆 68753 cycles 減為 42291 cycles，由此可知，加上 FIFO 確實對 UART 需傳輸多筆資料的情形下有相當大的時間效益。

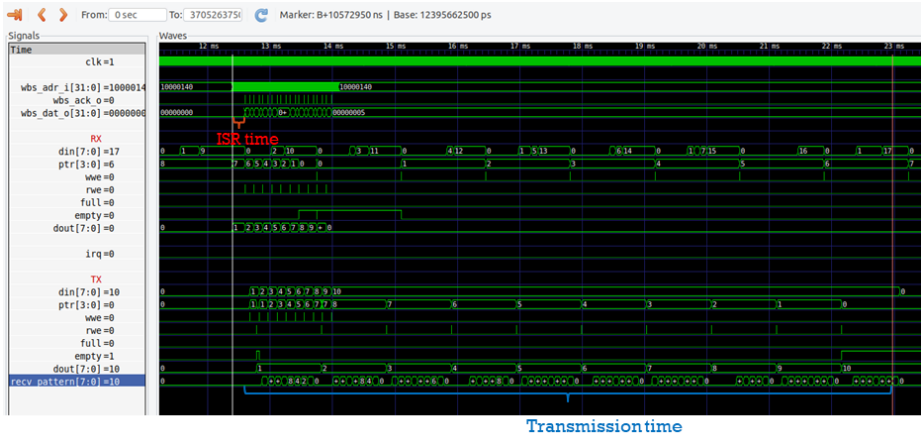
Baseline :

UART time : ISR time + Transmission time = 68753 cycles per data



Our work :

UART time : ISR time + Transmission time*10 = 422918 cycles = 42291 cycles per data



Quality of result

在 baud rate 9600 傳輸 512 筆 characters 的情形下，latency 可以由原先的 880038.4 us 減為 541324.8 us，而 metric 也由 826705.066 us 減少為 487991.466 us，都有大幅的效能提升。

Baseline :

Latency = (68753*512*25)/1000 = 880038.4 us

Metric = 880038.4 us – 512*(1/9600) = 826705.066 us

Our work :

Latency = (42291*512*25)/1000 = 541324.8 us

Metric = 541324.8 us - 512*(1/9600) = 487991.466 us

五、Synthesis:

AREA :

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	4080	0	0	53200	7.67
LUT as Logic	4026	0	0	53200	7.57
LUT as Memory	54	0	0	17400	0.31
LUT as Distributed RAM	16	0			
LUT as Shift Register	38	0			
Slice Registers	4234	0	0	106400	3.98
Register as Flip Flop	4157	0	0	106400	3.91
Register as Latch	77	0	0	106400	0.07
F7 Muxes	168	0	0	26600	0.63
F8 Muxes	47	0	0	13300	0.35

2. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	5	0	0	140	3.57
RAMB36/FIFO*	2	0	0	140	1.43
RAMB36E1 only	2				
RAMB18	6	0	0	280	2.14
RAMB18E1 only	6				

TIME：

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             8.465ns  (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:        6.003ns  (logic 0.374ns (6.230%)  route 5.629ns (93.770%))
  Logic Levels:           3  (BUFG=1 LUT1=1 LUT6=1)
  Clock Path Skew:        2.657ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.657ns = ( 27.657 - 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_X0Y21       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=5494, routed)        2.079    15.873    design_1_i/caravel_0/inst/gpio_control_in_1[7]/clock
  SLICE_X49Y97         LUT6 (Prop_lut6_I3_O)        0.124    15.997 f  design_1_i/caravel_0/inst/gpio_control_in_1[7]/mprj_o[15]_INST_0/O
                       net (fo=1, routed)           1.252    17.249    design_1_i/caravel_0/inst/gpio_control_in_1[7]/mprj_o[0]
  SLICE_X26Y97         LUT1 (Prop_lut1_I0_O)        0.149    17.398 f  design_1_i/caravel_0/inst/gpio_control_in_1[7]/mprj_o[0]_hold_fix/O
                       net (fo=1, routed)           1.105    18.503    design_1_i/caravel_ps_0/inst/control_s_axi_U/mprj_o[0]_hold_fix_1_alias_1
  SLICE_X49Y97         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_X0Y21       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=5494, routed)        1.478    27.657    design_1_i/caravel_ps_0/inst/control_s_axi_U/ap_clk
  SLICE_X49Y97         FDRE                                       r  design_1_i/caravel_ps_0/inst/control_s_axi_U/int_ps_mprj_out_reg[15]/C
                       clock pessimism              0.000    27.657
                       clock uncertainty           -0.377    27.280
  SLICE_X49Y97         FDRE (Setup_fdre_C_D)       -0.312    26.968    design_1_i/caravel_ps_0/inst/control_s_axi_U/int_ps_mprj_out_reg[15]
-------------------------------------------------------------------
                       required time                         26.968
                       arrival time                         -18.503
-------------------------------------------------------------------
                       slack                                 8.465
```

六、Improve after 1/18:

經老師提醒，因這個設計不能保證 UART 的接收端一定能有值傳入，若是 UART 的接收端沒有值傳入的話我們的功能就會傳 0，但這樣其實是不符合功能的，此設計有缺陷。

因此我們嘗試修改 isr.c 使其能夠偵測 RX 是否一直繼續傳才執行此功能，但就必須在 isr.c 的讀取 uart rx fifo 以及寫入 uart tx fifo 之前額外做 polling 去進行判斷，而此功能會增加額外的 cycles。

經過嘗試後我們發現只能減少一個 RX FIFO 深度的好處不足以用如此多的額外 latency 去交換，因此決定改成只能減少 TX FIFO 深度的設計來作為最後的成果，此設計的好處是能夠 optimize UART 的 FIFO 深度。