112-1 Soc Design Laboratory

Final Project

組別:第 28 組

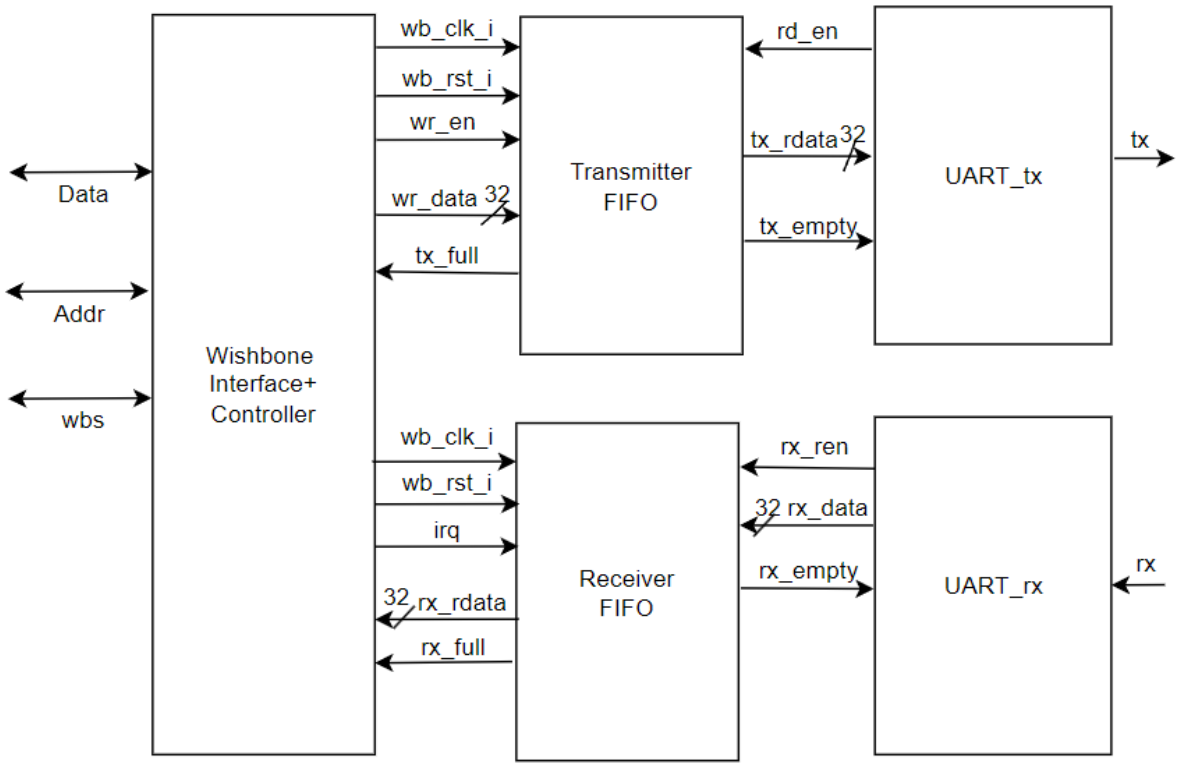
組員:丁緒翰、潘金生、王證皓

1. UART Accelerators

(1) Introduction

在此次 Final Project 中，我們針對 UART 的傳遞做加速，硬體實現的方法為使用 FIFO(Fist In First Out)來接收資料，原因在於 Processor 在執行時，會使用高速的 clock(e.g. 50,000,000Hz)進行操作，但 UART 則使用慢速的 clock(e.g. 9600Hz)，所以一旦 CPU 出現 interrupt 需使用到 UART 進行傳輸時，Processor 就必須等待 UART 全部傳輸完成才能繼續執行其他工作，這種情況會造成效能嚴重下降，因此我們引進 FIFO 來使用高速的 clock 來快速接收資料，接收完成即可執行其他工作，以此來提高效能，並同時執行 fir、matmul、qsort 三種函數來測試我們的效能，最後使用 FPGA 進行驗證。

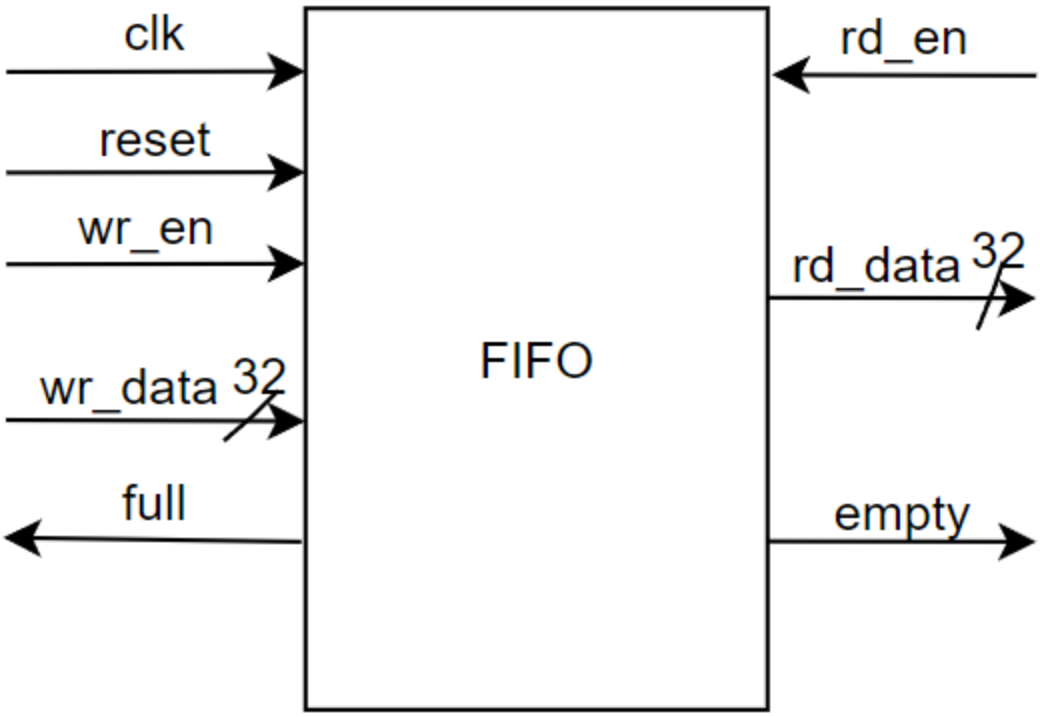
(2) Block Diagram



圖(一)、UART Block Diagram

上圖為我們 UART 整體的架構，可以看到 Wishbone Interface 與 UART_tx 及 UART_rx 中使用兩個 FIFO 來做接收，並將接收的資料與兩個 UART 進行讀寫，其中 Controller 的部分主要是用來將 CPU 傳送過來的 address 與 data 做 decode 並分配給兩個 FIFO，decode 的方法會於 Register specification 做詳細的介紹。

(3) FIFO Function



圖(二)、FIFO Diagram

訊號:

clk:Processor 時脈

reset:Processor reset

wr_en:write enable

wr_data:write data 寫資料

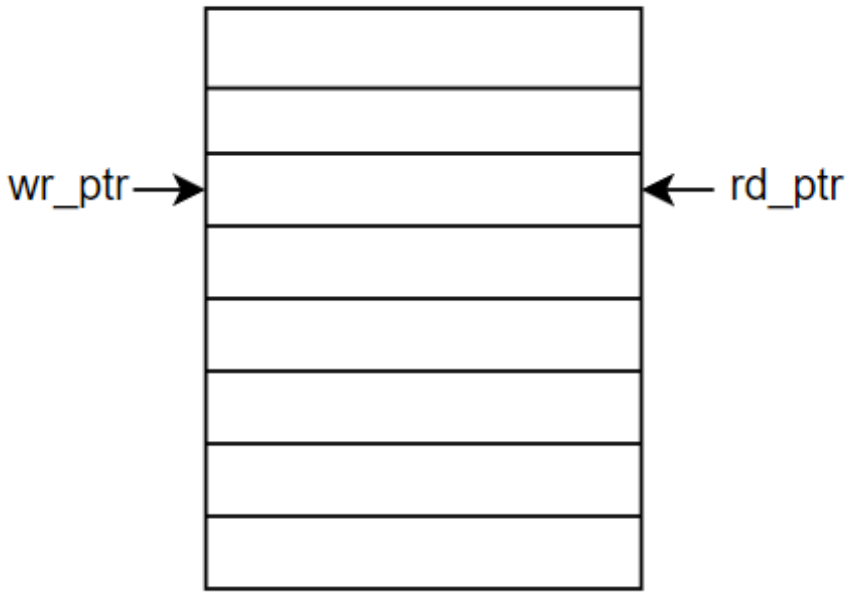
full:判斷 FIFO 是否為滿狀態

rd_en:read enable

rd_data:read data 讀資料

empty:判斷 FIFO 是否為空狀態

以上為我們所設計的 FIFO，其中 depth 我們設為 512，因需傳送 512 筆 character，但我們並不想讓 FIFO 反覆中斷，所以中斷一次即可讓 512 筆資料完整地接收完全部資料，圖(三)為 FIFO 使用的 pointer 示意圖，分別有 write pointer 與 read pointer，判斷空滿的方法如圖(四)，使用 counter 來計算已經儲存多少筆有效的資料，滿足寫入有效的條件為 FIFO 不能為滿，而讀出有效的條件為 FIFO 不能為空，



圖(三)、FIFO Read & Write Pointer

```
70   always@(posedge clk)begin
71     if(reset)begin
72       count<=9'd0;
73     end
74     else if(wr_en&~rd_en&~full) begin
75       count<=count+9'd1;
76     end
77     else if(rd_en&~wr_en&~empty) begin
78       count<=count-9'd1;
79     end
80     else begin
81       count<=count;
82     end
83   end
84
85   assign full=(count==9'd511)?1'b1:1'b0;
86   assign empty=(count==9'd0)?1'b1:1'b0;
87
88 endmodule
```

圖(四)、FIFO 判斷空滿

(4) Register Specification

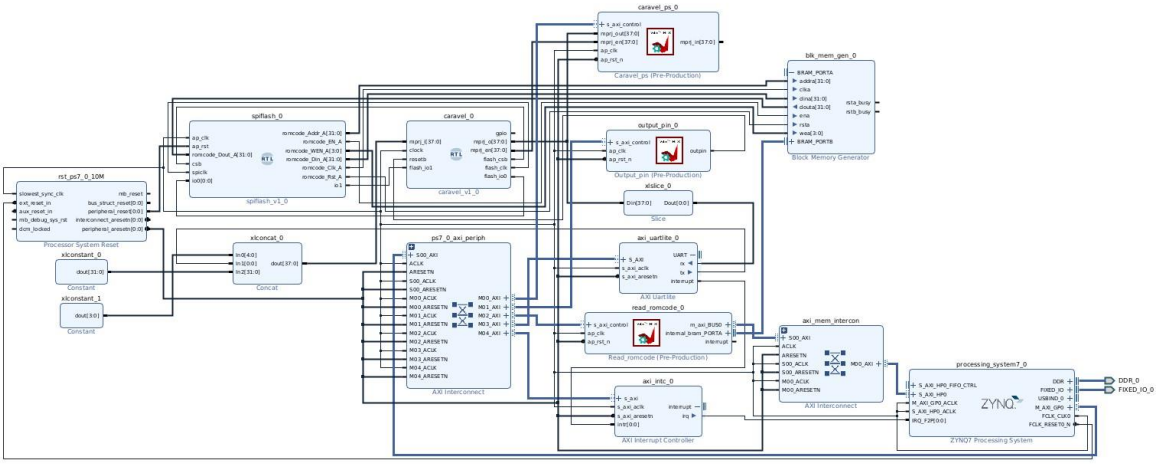
Rx_data:32'h3000_0000

Tx_data:32'h3000_0004

STAT_REG:32'h3000_0008

以上為 Register Specification 的三組 address，其中有 rx_data 與 tx_data，分別代表當 Wishbone 輸入的 address 為此兩組數值時，FIFO 能判斷並拉起 enable 訊號進行接收資料，並將資料傳入 UART 中進行傳輸。

(5) Vivado Synthesis Block Diagram

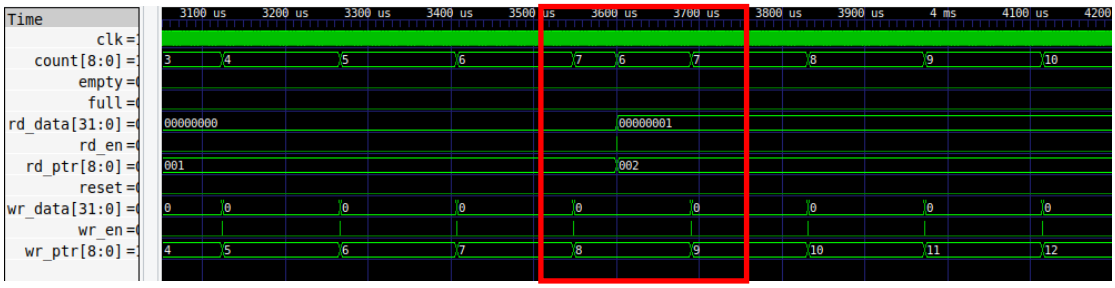


圖(五)、Vivado Synthesis Block Diagram

圖(五)使用 Vivado 進行合成，時脈設置為 40MHz

(6) Simulation Waveform

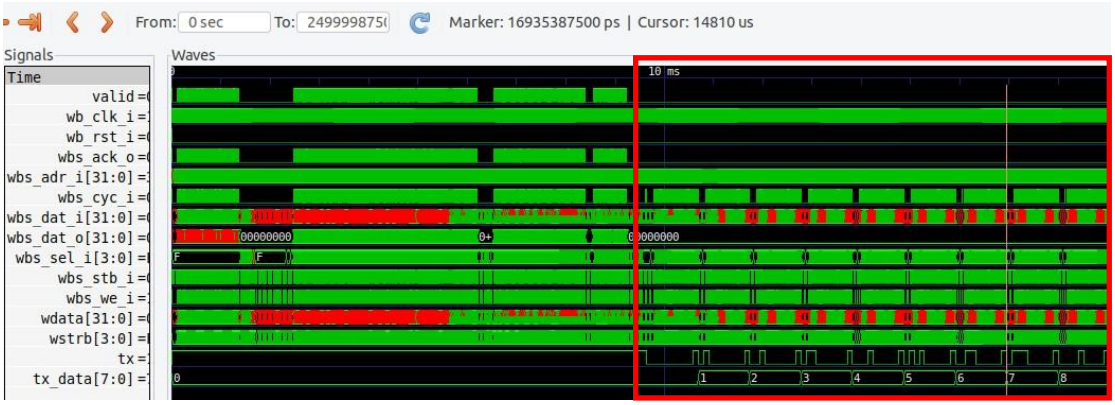
FIFO



圖(六)、FIFO waveform

從圖中可知，當有 wr_en 訊號時，wr_ptr 會加 1，而 count 則代表 FIFO 內部實際的資料量，接收到 wr_en 會加 1，而接收到 rd_en 後會減 1，如紅框所示，count 讀出後從 7 便到 6，而寫入後則加 1 變回 7。

UART Without FIFO



圖(七)、UART without FIFO waveform

圖(七)為無 FIFO 的模擬波形，可以看到當 tx 在傳遞時，Processor 是無法執行其他工作的，從 valid 與 wishbone 的各個訊號都可以看到 Processor 被 stall 住，Latency 為 1042 us x 512=533.747ms。

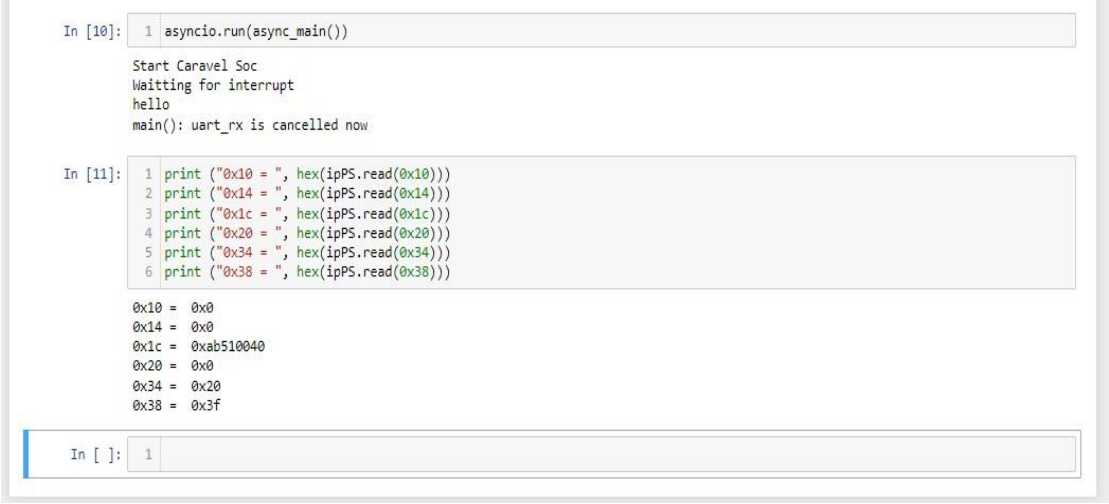
UART With FIFO

圖(八)、UART with FIFO waveform

圖(八)為使用 FIFO 後的波形，可以看到 FIFO 中 wr_ptr 使

用高速的 clock 來寫入接收資料，這邊先以傳送 100 筆資料做測試，其中 CPU 只需等待 FIFO 接收完即可，不須連 rx 也一起等待，而此時 CPU 可繼續執行先前提到的 FIR、matmul、qsort，Latency 為 141.375 us x 512=72.384ms，Metrics 為 72.384 - 500 x (1/9600)=72.332 ms。

(7) Run on FPGA



圖(九)、FPGA 驗證

圖(九)可看到我們使用 UART 傳送”hello”的字串，並從 FPGA 的結果可驗證我們的電路是否正確，而圖中下方則是我們執行完全部 firmware code 後，會輸出一個 0xab51 開頭的位置，代表功能正確。

(8) Quality of Result

With FIFO

Latency=1042 us x 512=533.747ms

Metrics=533.747 - 500 x (1/9600)=0.4817

With FIFO

Latency=141.375 us x 512=72.384ms

Metrics=0.72384 - 500 x (1/9600)=0.0203

比較有無 FIFO 的結果，可觀察到 Latency 明顯大幅下降，從原本的 533.747ms 下降到 72.384 ms，代表著加入 FIFO 確實能對 UART 進行加速，並且 Metrics 也能達到更小的數值。

Github Link:

https://github.com/Alex17898/SOC-Design/tree/main/Final