Final Report

Group5

Members:

M11207415 陳謝鎧

M11207002 陳泊佑

M11107426 廖千慧

M11207328 吳奕帆

SDRAM 優化

1. Introduce the prefetch scheme

先看是否有 in_valid，若有的話，則把 prefetch_addr 改為 addr +22'd4，如果今天 prefetch_addr 與 addr 相等且 rw 信號為 0 的時候，那就將 prefetch 改為 1，那在 IDLE 的時候，會去判斷是否要 prefetch，如果要的話，就直接把剛剛 dqi_d 的值放入 data_d，並且 out_valid_d 改為 1，進行輸出。

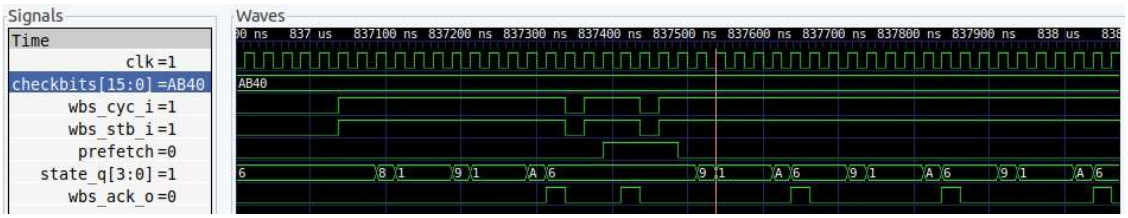
第一次進行 READ 的時候會需要先進行 ACTIVATE，這個部分需要有 3 個 T 的等待時間，而在第二次如果讀取的是同一個 BANK 時，則只需要 3 個 T 的 READ 來讀取資料就可以了，所以 READ_RES 所花的時間都只有 1 個 T，這樣也可以省掉 ACTIVATE 3 個 T 的等待時間，如下圖所示。

- Code

```
///////////////set prefetch condition
    always@(posedge clk)begin
        if(in_valid)
            prefetch_address <= addr + 22'd4;
     else if(!in_valid && out_valid_q)
            prefetch_address <= 0;

     if((prefetch_address == addr) && in_valid && !rw)
            prefetch <= 1;
     else if((prefetch_address != addr) && in_valid & !rw)
            prefetch <= 0;
     else if(rw)
            prefetch <= 0;
    end
```

- wave



2. Introduce the bank interleave for code and data

a. 透過 section 來指定放 code 跟 data 的位置。

b. .接著在利用 wbs_adr_i[31:24]、wbs_stb_i、wbs_cyc_i 來區分 req_code (3800)還是 req_data (3000) ，若 request_code 為 1，表示需要看 code_adr 並依據其數據選擇要哪個 bank，bank 的選擇為 2(10) 或 3(11)。code_addr={wbs_adr_i[22:10],code_bank,wbs_adr_i[7:0]}。若 req_data 為 1，表示需要看 data_addr 的位址，選定要哪一個 bank，bank 的選擇為 0(00) 或 1(01)。data_addr={wbs_adr_i[22:10],data_bank,wbs_adr_i[7:0]}

- section.ids:

```
.data :
{
        . = ALIGN(8);
        _fdata = .;
        *(.data .data.* .gnu.linkonce.d.*)
        *(.data1)
        _gp = ALIGN(16);
        *(.sdata .sdata.* .gnu.linkonce.s.*)
        . = ALIGN(8);
        _edata = .;
} > mprj AT > flash

.bss :
{
        . = ALIGN(8);
        _fbss = .;
        *(.dynsbss)
        *(.sbss .sbss.* .gnu.linkonce.sb.*)
        *(.scommon)
        *(.dynbss)
        *(.bss .bss.* .gnu.linkonce.b.*)
        *(COMMON)
        . = ALIGN(8);
        _ebss = .;
        _end = .;
} > mprj AT > flash
```

```
Disassembly of section .bss:

30000080 <flag>:
30000080:      0000                .2byte  0x0
       ...

30000084 <result>:
       ...

Disassembly of section .mprjram:

38000000 <matmul>:
38000000:      fd010113            addi    sp,sp,-48
38000004:      02112623            sw      ra,44(sp)
38000008:      02812423            sw      s0,40(sp)
3800000c:      03010413            addi    s0,sp,48
38000010:      fe042623            sw      zero,-20(s0)
38000014:      fc042e23            sw      zero,-36(s0)
38000018:      fe042623            sw      zero,-20(s0)
3800001c:      0e80006f            j       38000104 <matmul+0x104>
38000020:      fe042423            sw      zero,-24(s0)
38000024:      0c80006f            j       380000ec <matmul+0xec>
38000028:      fe042023            sw      zero,-32(s0)
3800002c:      fe042223            sw      zero,-28(s0)
38000030:      07c0006f            j       380000ac <matmul+0xac>
38000034:      fec42783            lw      a5,-20(s0)
```

```
Disassembly of section .data:

30000000 <A>:
30000000:      0000                .2byte  0x0
30000002:      0000                .2byte  0x0
30000004:      0001                .2byte  0x1
30000006:      0000                .2byte  0x0
30000008:      0002                .2byte  0x2
3000000a:      0000                .2byte  0x0
3000000c:      00000003            lb      zero,0(zero) # 0 <_DYNAMIC>
30000010:      0000                .2byte  0x0
30000012:      0000                .2byte  0x0
30000014:      0001                .2byte  0x1
30000016:      0000                .2byte  0x0
30000018:      0002                .2byte  0x2
3000001a:      0000                .2byte  0x0
3000001c:      00000003            lb      zero,0(zero) # 0 <_DYNAMIC>
```

- user_code

```
// request signal
    wire req_data;
    wire req_code;
    wire[1:0] data_bank;//data
    wire [22:0] data_addr;
    wire[1:0] code_bank;//code
    wire [22:0] code_addr;
    assign req_data = (wbs_stb_i & wbs_cyc_i &(wbs_adr_i[31:24] == 8'h30)) ? 1 : 0;
    assign req_code = (wbs_stb_i & wbs_cyc_i &(wbs_adr_i[31:24] == 8'h38)) ? 1 : 0;
    assign data_bank = (wbs_adr_i[9:8]>2'b01) ? (wbs_adr_i[9:8]-2'b10) : wbs_adr_i[9:8];
    assign data_addr = (req_data) ? {wbs_adr_i[22:10], data_bank, wbs_adr_i[7:0]} : 0;
    assign code_bank = (wbs_adr_i[9:8]<2'b10) ? (wbs_adr_i[9:8]+2'b10) : wbs_adr_i[9:8];
    assign code_addr = (req_code) ? {wbs_adr_i[22:10], code_bank, wbs_adr_i[7:0]} : 0;
```

3. Result

a. 這邊我們是使用 O2 來優化 compiler，並透過 checkbit 檢查是否有完成矩陣功能跟 qs 排序功能。

b. 目前我們只有完成 qs 跟矩陣乘法使用 sdram 來做優化，主要因為在整合過程中還有尚未解決的問題所以還沒有把它完全整合起來。

- counter_la_mm

```
ubuntu@ubuntu2004:~/Desktop/Final_test-main/testbench/counter_la_qs$ source run_clean
ubuntu@ubuntu2004:~/Desktop/Final_test-main/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
counter_la_qs_tb.uut.mprj.mprj.user_bram : at time            848838000 ERROR: Bank is not Activated fo
r Read
counter_la_qs_tb.uut.mprj.mprj.user_bram : at time            989863000 ERROR: Bank is not Activated fo
r Read
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, 0x0ca1
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x1787
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x2371
LA Test 2 passed
```

- counter_la_qs

```
ubuntu@ubuntu2004:~/Desktop/soclab-sdram/testbench/counter_la_mm$ source run_clean
ubuntu@ubuntu2004:~/Desktop/soclab-sdram/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
counter_la_mm_tb.uut.mprj.mprj.user_bram : at time           4896263000 ERROR: Bank is not Activated for Read
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
```

UART_FIFO:

1. 程式碼設計部分:

為了要使其有 FIFO 的功能主要新增或改善了以下幾個於 RTL 的 USER 中的 module，首先建立一個關於 FIFO 的 module，在這個 module 主要用於 FIFO 的設計，包含將資料寫入 FIFO 內，以及將資料從 FIFO 中讀取，同時也包含了 fifo 是否為空或是否為滿以及中斷信號 irq 的控制。可以看到此張圖所顯示的即為 synchronous_fifo 這個 module 中關於如何將資料進行寫入的設計，當 write 為 enable 且 fifo 處在沒有滿的狀態時將 data_in 寫進 fifo。接著這張則為資料如何從 fifo 中讀取出來的程式設計，當 read 為 enable 且 fifo 不為空的狀態時，將 fifo 內的資料賭取出來為 data_out。

接著第二個調整的 module 為 uart_receive 這個 module，因為我們這次是針對 SOC Rx 的部分加了 FIFO，因此也要調整原本 UART 中這個 module 使其可以有 fifo 的功能。

最後為控制 UART 的 module。透過這個 module 將 UART 的許多信號加以控制。其中包含中斷信號、fifo 是 empty 還是 full 等等。讓我們再 Rx 所添加的 fifo 可以順利執行。

- Module name: synchronous_fifo

```
module synchronous_fifo #(parameter DEPTH=4, DATA_WIDTH=8) (
  input clk, rst_n,
  input W_en, R_en,
  input [DATA_WIDTH-1:0] Data_in,
  output reg [DATA_WIDTH-1:0] Data_out,
  output fifo_full, fifo_empty, irq_sig,
  output [$clog2(DEPTH)-1:0] data_num
);
```

將資料寫入 FIFO

```
/* Write data to fifo */
always@(posedge clk) begin
  if(W_en & !fifo_full)begin
    fifo[W_ptr] <= Data_in;
    W_ptr <= W_ptr + 1;
  end
end
```

將資料讀取出 FIFO

```
/* Read data from fifo */
always@(posedge clk) begin
  if(R_en & !fifo_empty) begin
    Data_out <= fifo[R_ptr];
    R_ptr <= R_ptr + 1;
  end
end
```

- Module name: uart_receive

```
module uart_receive (
  input wire        rst_n,
  input wire        clk,
  input wire [31:0] clk_div,
  input wire        rx,
  //output reg        irq,
  output reg        o_rx_done,
  output reg [7:0]  rx_data,
  //input wire        rx_finish,
  input wire        i_ctrl_done,
  output reg        frame_err,
  output reg        busy
);
```

- Module name: ctrl:

```
module ctrl(
    input wire        rst_n,
    input wire        clk,
    input wire [31:0] clk_div,
    input wire        i_wb_valid,
    input wire [31:0] i_wb_adr,
    input wire        i_wb_we,
    input wire [31:0] i_wb_dat,
    input wire [3:0]  i_wb_sel,
    output reg        o_wb_ack,
    output reg [31:0] o_wb_dat,
    input wire [7:0]  i_rx,
    input wire        i_rx_done,  // i_byte_finish
    //input wire        i_irq,   //
    input wire        i_rx_busy,
    input wire        i_frame_err,
    output reg        o_ctrl_done,  // o_ctrl_byte_finish
    output reg [7:0]  o_tx,
    input wire        i_tx_start_clear,
    input wire        i_tx_busy,
    output reg        o_tx_start,
    output            o_irq
);
```

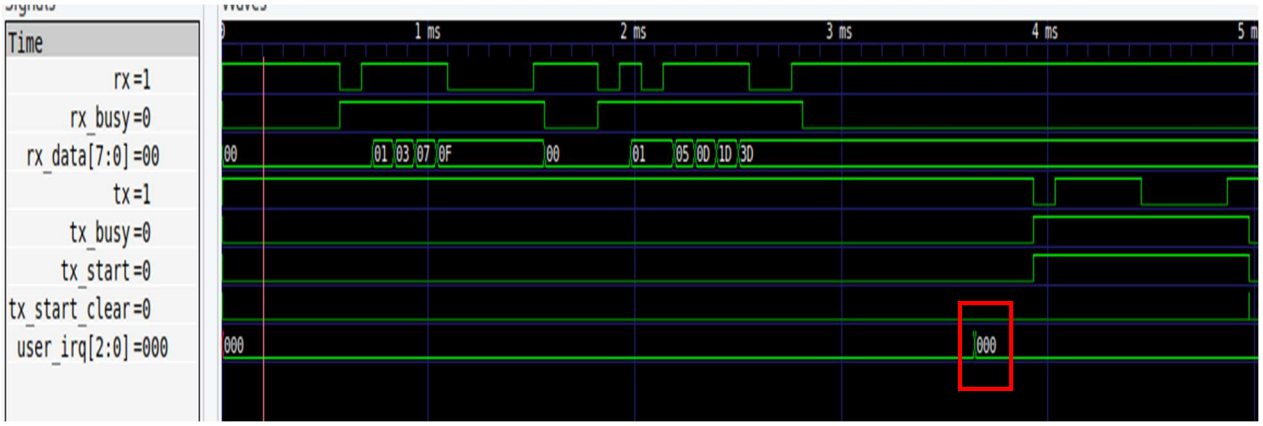
2. 執行結果:

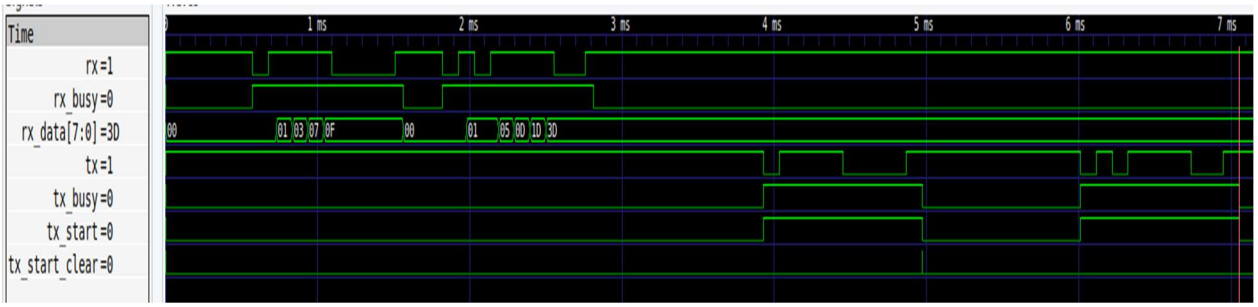
此圖是我們依照上述所提到的程式碼經由終端機執行得結果，由此圖可以看到所傳的兩筆資料皆有被完整的接收。

```
ubuntu@ubuntu2004:~/Desktop/Final_test-main/testbench/uart$ source run_clean
ubuntu@ubuntu2004:~/Desktop/Final_test-main/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: 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
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: 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
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
```

3. 波型:

第一張波型圖中可以看到 rx 總共傳了兩筆資料後又過了一小段時間，中斷信號 irq 才升起(紅色框框處)，使 CPU 只有進行一次中斷，而非原先的每傳完一筆資料就需要先中斷一次，達成 FIFO 一次傳多筆資料後才進行中斷的結果。





4. 總結:

我們這次做了 UART FIFO 和 SDRAM＋QS 和 MM，由於再做這個 project 的時候遇到了蠻多的問題，且有些尚未解決整合一直無法順利地進行，雖然處處碰壁但還是學到很多。