國立清華大學
NATIONAL TSING HUA UNIVERSITY

EE 6250 VLSI Testing

Homework#3

資工系碩士班一年級

趙奕誠/ Yi-Cheng,Chao/105062600

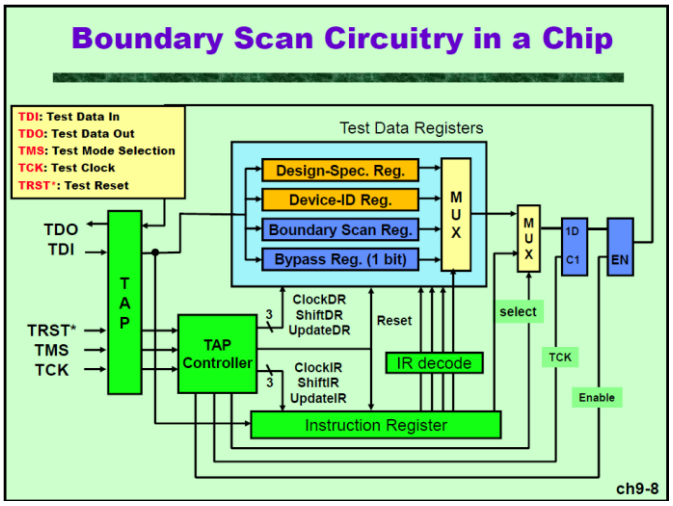
2017.01.19

VLSI Testing Homework #3

Yi-Cheng,Chao January 19, 2017

Source Code (Environment: NC-verilog)

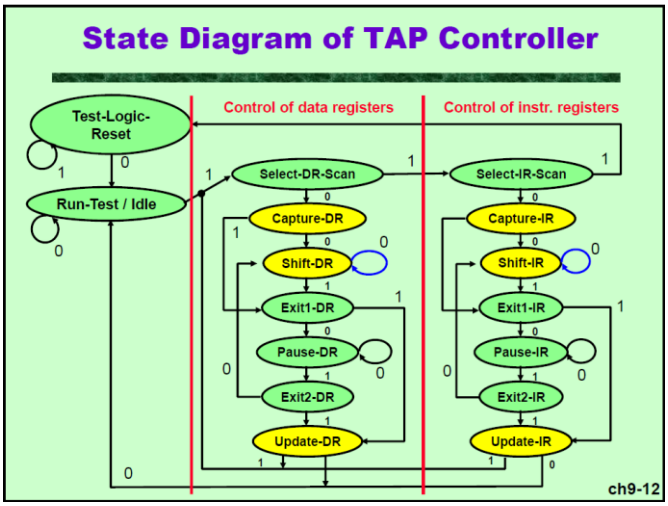
主要架構參考自課本第九章提供的 IEEE 1149.1 架構來製作 BSC，如下圖所示：



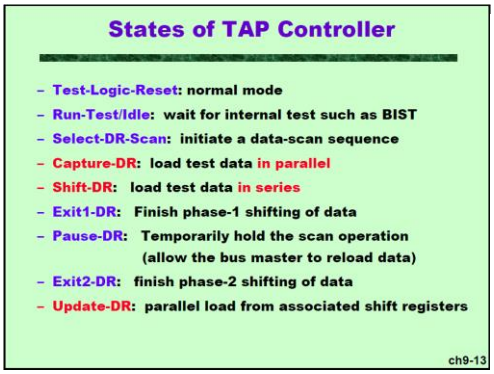
各細部 module 將在底下詳述其功能

(1) TAP-Controller

根據講義提供之狀態圖為其設定 FSM，訊號皆由 TMS 來進行控制。




```
always @(posedge tck)
    begin
        case(state)
            Run_Test_Idle:  if(tms == 0)    state = Run_Test_Idle;
                else                        state = Select_Dr;
            Select_Dr:  if(tms == 0)        state = Capture_Dr;
                else                        state = Select_Ir;
            Capture_Dr: if(tms == 0)        state = Shift_Dr;
                else                        state = Exit1_Dr;
            Shift_Dr:   if(tms == 0)        state = Shift_Dr;
                else                        state = Exit1_Dr;
            Exit1_Dr:   if(tms == 0)        state = Pause_Dr;
                else                        state = Update_Dr;
            Pause_Dr:   if(tms == 0)        state = Pause_Dr;
                else                        state = Exit2_Dr;
            Exit2_Dr:   if(tms == 0)        state = Shift_Dr;
                else                        state = Update_Dr;
            Update_Dr:  if(tms == 0)        state = Run_Test_Idle;
                else                        state = Select_Dr;
            Select_Ir:  if(tms == 0)        state = Capture_Ir;
                else                        state = Reset;
            Capture_Ir: if(tms == 0)        state = Shift_Ir;
                else                        state = Exit1_Ir;
            Shift_Ir:   if(tms == 0)        state = Shift_Ir;
                else                        state = Exit1_Ir;
            Exit1_Ir:   if(tms == 0)        state = Pause_Ir;
                else                        state = Update_Ir;
            Pause_Ir:   if(tms == 0)        state = Pause_Ir;
                else                        state = Exit2_Ir;
            Exit2_Ir:   if(tms == 0)        state = Shift_Ir;
                else                        state = Update_Ir;
            Update_Ir:  if(tms == 0)        state = Run_Test_Idle;
                else                        state = Select_Dr;
            Reset:  if(tms == 0)            state = Run_Test_Idle;
                else                        state = Reset;
            default:                        state = Reset;
        endcase
    end
```




```
// Output Assignments
assign reset = (state == 4'b1000);
assign select = state[3]; // 1 for IR, 0 otherwise

assign capture_ir = (state == Capture_Ir);
assign shift_ir = (state == Shift_Ir);
assign update_ir = (state == Update_Ir);
assign clock_ir = ((capture_ir || shift_ir || update_ir) && ~tck);

assign capture_dr = (state == Capture_Dr);
assign shift_dr = (state == Shift_Dr);
assign update_dr = (state == Update_Dr);
assign clock_dr = ((capture_dr || shift_dr || update_dr) && ~tck);

assign enable = shift_ir || shift_dr;
```

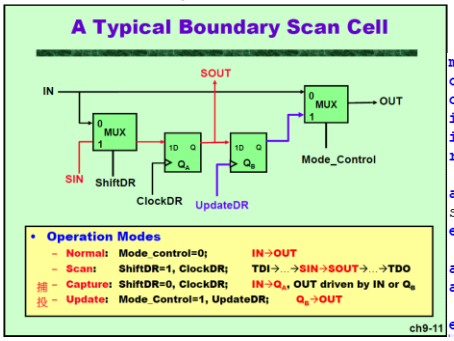
根據目前的 state 輸出其訊號為 1，否則便為 0。

(2) Bypass-Register

```
module  Bypass_Register(scan_out, scan_in, shiftDR, clockDR);
  output     scan_out;
  input      scan_in, shiftDR, clockDR;
  reg        scan_out;

  always @ (posedge clockDR) scan_out <= scan_in & shiftDR;

endmodule
```

Bypass-Register 則為收到 shiftDR 的訊號後去傳遞 scan_in 的值到 scan_out，若沒收到則不傳遞 scan_in 的值到 scan_out (即 hold 住原本 scan_out 的值)。

(3) Boundary Scan Cell




```
module BSC_Cell  (data_out, scan_out, data_in, mode, scan_in, shiftDR, updateDR, clockDR);
output  data_out;
output  scan_out;
input   data_in;
input   mode, scan_in, shiftDR, updateDR, clockDR;
reg     scan_out, update_reg;

always @ (posedge clockDR) begin
scan_out <= shiftDR ? scan_in : data_in;
end

always @ (posedge updateDR) update_reg <= scan_out;
assign data_out = mode ? update_reg : data_in;

endmodule
```

BSC 部分亦是根據講義架構完成，如上圖所示，主要概念就是用 shiftDR 及 mode_control 去做兩個 multiplier 的控制，以及分別由 ClockDR 及 UpdateDR 控制兩個 DFF 的訊號進出。

(4) Instruction Decoder

```
module Instruction_Decoder (mode, select_BR, shift_BR, clock_BR, shift_BSC_Reg, clock_BSC_Reg, update_BSC_Reg, instruction, shiftIR, clockIR, updateIR);
  parameter     IR_size             = 2;
  output                    mode, select_BR, shift_BR, clock_BR;
  output                    shift_BSC_Reg, clock_BSC_Reg, update_BSC_Reg;
  input                  shiftIR, clockIR, updateIR;
  input     [IR_size -1: 0]     instruction;
  parameter BYPASS      = 2'b00;
  parameter EXTEST          = 2'b01;
  parameter INTEST          = 2'b01;
  parameter SAMPLE_PRELOAD  = 2'b11;
  reg       mode, select_BR, clock_BR, clock_BSC_Reg, update_BSC_Reg;

  assign    shift_BR = shiftIR;
  assign        shift_BSC_Reg = shiftIR;

  always @ (instruction or clockIR or updateIR) begin
    mode = 0; select_BR = 0;        // default is test-data register
    clock_BR = 1; clock_BSC_Reg = 1;
    update_BSC_Reg = 0;

    case (instruction)
      BYPASS:       begin select_BR = 1; clock_BR = clockIR; end        // 1 selects bypass reg
      EXTEST:       begin mode = 1; clock_BSC_Reg = clockIR; update_BSC_Reg = updateIR; end
      INTEST:       begin mode = 1; clock_BSC_Reg = clockIR; update_BSC_Reg = updateIR; end
      SAMPLE_PRELOAD:   begin  clock_BSC_Reg = clockIR; update_BSC_Reg = updateIR; end
      default:      begin select_BR = 1; end                    // 1 selects bypass reg

    endcase
  end
endmodule
```

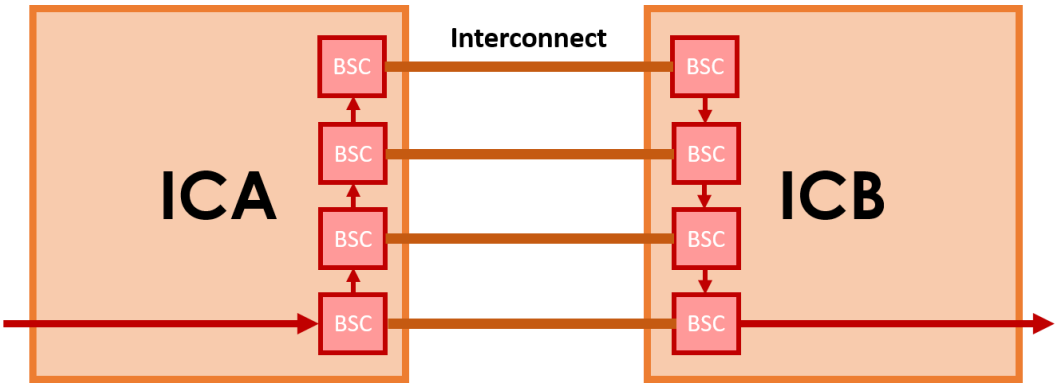
收到 ClockIR、ShiftIR、UpdateIR 訊號後根據收到的訊號進行判斷，傳遞所要傳遞的訊號給 BSC。

(5) Instruction Register

```
module Instruction_Register (data_out, data_in, scan_out, scan_in,  shiftIR, clockIR, updateIR, reset_bar);
parameter IR_size = 2;
output  [IR_size -1: 0] data_out;
output          scan_out;
input   [IR_size -1: 0] data_in;
input           scan_in;
input           shiftIR, clockIR, updateIR, reset_bar;
reg [IR_size -1: 0] IR_Scan_Register, IR_Output_Register;

assign          data_out = IR_Output_Register;
assign          scan_out = IR_Scan_Register [0];

always @ (posedge clockIR) IR_Scan_Register <= shiftIR ? {scan_in, IR_Scan_Register [IR_size - 1: 1]} : data_in;

always @ ( posedge updateIR or negedge reset_bar)
  if (reset_bar == 0) IR_Output_Register <= ~(0);       // Fills IR with 1s for BYPASS instruction
  else IR_Output_Register <= IR_Scan_Register;

endmodule
```

儲存收到的訊號指令，並且根據 ShiftIR 訊號判斷是否要將新收到的指令傳遞出去。

但因為之後發現在實作的一些困難，因此我簡化省略了 IR-Cell 的部分，直接從 Host 端控制 shiftDR，mode 等訊號，而且原題目指要求傳遞 4 條 IC 間的 interconncet，因此最後設計僅使用 Boundary Scan Cell 來進行 Boundary Scan Test，而因為有四條 interconnect，因此需要八組 BSC，如下圖所示：



下圖為實際的程式碼，interconnect 的部分使用 4bit 的 data，每個 bit 代表一條 interconnect，並且我可以直接從 out 端看到我輸出的結果方便我進行 pattern 是否完整傳輸的驗證。

```
module TOP (    output [3:0]OUT,
                output TDO_A, TDO_B,
                input [3:0]IN,
                input TDI_A, Mode_Control_A, ShiftDR_A, UpdateDR_A, ClockDR_A,
                input TDI_B, Mode_Control_B, ShiftDR_B, UpdateDR_B, ClockDR_B,
                input RST
);
    wire [3:0] data;

    IC A(.TDO(TDO_A), .data_out(data), .TDI(TDI_A), .data_in(IN), .Mode_Control(Mode_Control_A),
         .ShiftDR(ShiftDR_A), .UpdateDR(UpdateDR_A), .ClockDR(ClockDR_A), .RST(RST));
    IC B(.TDO(TDO_B), .data_out(OUT), .TDI(TDI_B), .data_in(data), .Mode_Control(Mode_Control_B),
         .ShiftDR(ShiftDR_B), .UpdateDR(UpdateDR_B), .ClockDR(ClockDR_B), .RST(RST));
endmodule
```

```
module IC (TDO, data_out, TDI, data_in, Mode_Control, ShiftDR, UpdateDR, ClockDR, RST);
    output [3:0] data_out;
    output TDO;
    input [3:0] data_in;
    input TDI, Mode_Control, ShiftDR, UpdateDR, ClockDR, RST;
    wire scan1, scan2, scan3, scan4, q1;

    BSC_Cell B1(.data_out(data_out[0]), .scan_out(scan1), .data_in(data_in[0]), .scan_in(TDI),
                .mode(Mode_Control), .shiftDR(ShiftDR),
                .updateDR(UpdateDR), .clockDR(ClockDR));
    BSC_Cell B2(.data_out(data_out[1]), .scan_out(scan2), .data_in(data_in[1]), .scan_in(scan1),
                .mode(Mode_Control), .shiftDR(ShiftDR),
                .updateDR(UpdateDR), .clockDR(ClockDR));
    BSC_Cell B3(.data_out(data_out[2]), .scan_out(scan3), .data_in(data_in[2]), .scan_in(scan2),
                .mode(Mode_Control), .shiftDR(ShiftDR),
                .updateDR(UpdateDR), .clockDR(ClockDR));
    BSC_Cell B4(.data_out(data_out[3]), .scan_out(scan4), .data_in(data_in[3]), .scan_in(scan3),
                .mode(Mode_Control), .shiftDR(ShiftDR),
                .updateDR(UpdateDR), .clockDR(ClockDR));


endmodule
```

每個 IC 有 4 個 Boundary Scan Cell，並且 in/out 端連接 bit-by-bit 的連接 data in/data out，並且用四條 wire(scan1~scan4)串聯形成 scan chain，Boundary Scan Cell 部分上面設計以闡述過故不在贅述。

Simulation Result (Environment: nWave)

由 ICA 輸入 TDI，之後使用 shiftDR 移動 TDI 輸入的 pattern 進行 scan chain 的移動，之後由 ICB data in 端接收到的資訊，之後再 scan out 出 pattern 完成一次的 pattern communication。
之後就由 TDI_a 進行灌入 pattern 的動作，設定 clock period 為 10ns，因此灌入一次完整的 pattern 需要 40ns，如下圖為每組的 pattern 詳細參數：

```
//iuput pattern 0000
#10 tdi_a = 0;
#10 tdi_a = 0;
#10 tdi_a = 0;
#10 tdi_a = 0;
//input pattern 0001
#10 tdi_a = 0;
#10 tdi_a = 0;
#10 tdi_a = 0;
#10 tdi_a = 1;
//input pattern 0010
#10 tdi_a = 0;
#10 tdi_a = 0;
#10 tdi_a = 1;
#10 tdi_a = 0;
//input pattern 0011
#10 tdi_a = 0;
#10 tdi_a = 0;
#10 tdi_a = 1;
#10 tdi_a = 1;
```

```
//input pattern 0100
#10 tdi_a = 0;
#10 tdi_a = 1;
#10 tdi_a = 0;
#10 tdi_a = 0;
//input pattern 0101
#10 tdi_a = 0;
#10 tdi_a = 1;
#10 tdi_a = 0;
#10 tdi_a = 1;
//input pattern 0110
#10 tdi_a = 0;
#10 tdi_a = 1;
#10 tdi_a = 1;
#10 tdi_a = 0;
//input pattern 0111
#10 tdi_a = 0;
#10 tdi_a = 1;
#10 tdi_a = 1;
#10 tdi_a = 1;
```

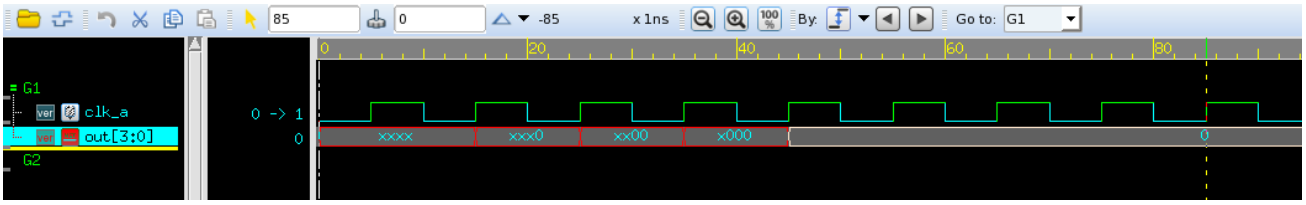
```
//input pattern 1000
#10 tdi_a = 1;
#10 tdi_a = 0;
#10 tdi_a = 0;
#10 tdi_a = 0;
//input pattern 1001
#10 tdi_a = 1;
#10 tdi_a = 0;
#10 tdi_a = 0;
#10 tdi_a = 1;
//input pattern 1010
#10 tdi_a = 1;
#10 tdi_a = 0;
#10 tdi_a = 1;
#10 tdi_a = 0;
//input pattern 1011
#10 tdi_a = 1;
#10 tdi_a = 0;
#10 tdi_a = 1;
#10 tdi_a = 1;
```

```
//input pattern 1100
#10 tdi_a = 1;
#10 tdi_a = 1;
#10 tdi_a = 0;
#10 tdi_a = 0;
//input pattern 1101
#10 tdi_a = 1;
#10 tdi_a = 1;
#10 tdi_a = 0;
#10 tdi_a = 1;
//input pattern 1110
#10 tdi_a = 1;
#10 tdi_a = 1;
#10 tdi_a = 1;
#10 tdi_a = 0;
//input pattern 1111
#10 tdi_a = 1;
#10 tdi_a = 1;
#10 tdi_a = 1;
#10 tdi_a = 1;
#40
```

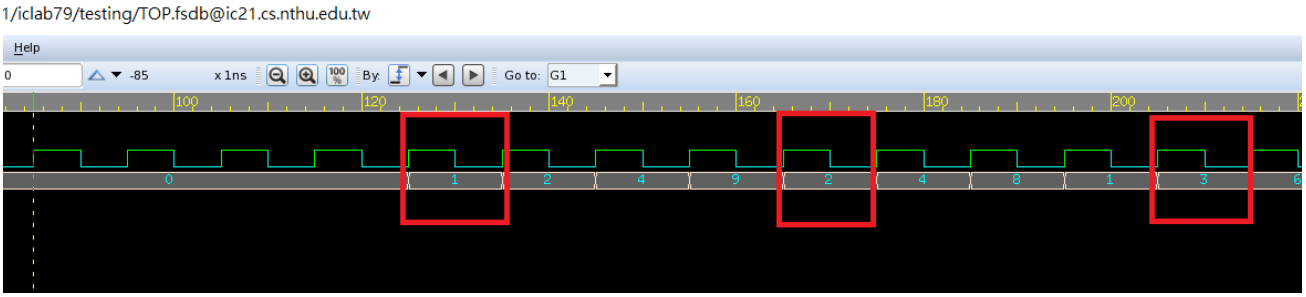
因此需要每 40ns 才可以在 out 端看到完整的結果，在使用 nWave 的時候需要多以下指令來產生 nWave 觀看波型圖時所使用的.sfdb 檔，如下所示：

```
initial begin
$fsdbDumpfile("TOP.fsdb");
$fsdbDumpvars;
end
```

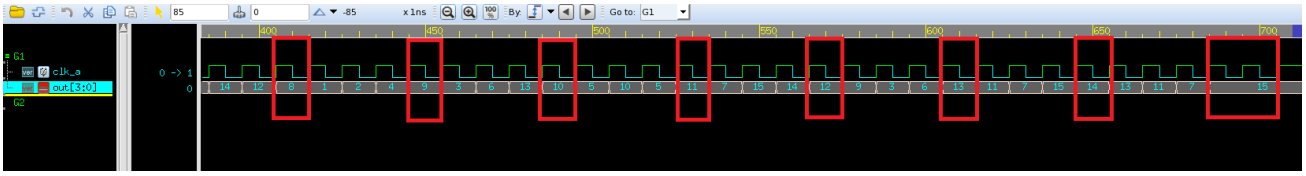
模擬後結果如下圖所示：



因為 pattern 從 ICA 的第一個 BSC 到 ICB 的第四個 BSC 須要經過 8 個完整的 positive edge 的時間，外加 dfilp-flop 本身需要一個 clock cylce 的時間，因此黃色虛線的位置為第一個從 ICB 的 data out 完整吃到 pattern 產生的結果。



之後每個四個 clock cycle 便可以看到完整 pattern output 的結果，如上圖所示依序為 1,2,3…



之後結果依序如上圖所示，值到完整個 16 組 pattern 產生出來後，模擬便會結束。

此外為了方便助教操作，我有附上 Makefile 可以直接使用 make sim 來進行模擬，產生的 nWave 觀看波型用的 fsdb 檔也一併附上。