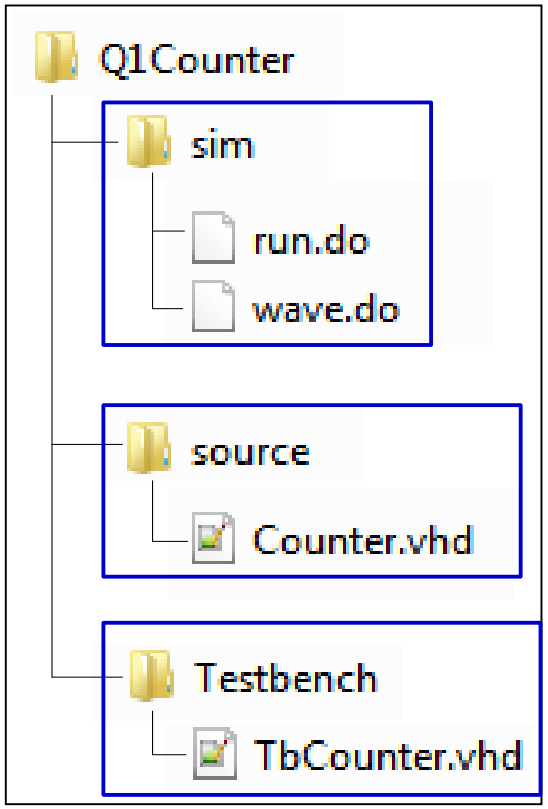
DIGITAL DESIGN WITH FPGA CAMP

DAY 2 TESTBENCH LAB

SIMULATION BY MODELSIM

Modelsim คือ โปรแกรมที่ใช้ในการ simulation ตัว HDL code ที่เราออกแบบ
เพื่อให้ง่ายต่อการเข้าใจจึงแนะนำโครงสร้าง folder ที่ใช้ simulation ดังนี้



sim folder: เป็น working directory ของ Modelsim ที่เป็น tool สำหรับ simulation
run.do : เป็น script file ที่บอกลำดับคำสั่งของ modelsim เช่น source code ที่จะใช้งาน ช่วงเวลาที่ต้องการ run simulation ว่า run นานเท่าไหร่
wave.do : เป็น save list ของสัญญาณที่ต้องการจะดู

source folder: เป็น folder สำหรับเก็บ source code ที่ออกแบบเพื่อทำงานบน hardware จริง และต้องการนำมาทดสอบการทำงาน

Testbench folder: เป็น folder สำหรับเก็บ source code ที่ใช้สำหรับ simulation เช่น testbench หรือ package ที่เรียกใช้สำหรับ simulation เท่านั้น

RUN.DO SCRIPT

run.do

```
1  ##
2  quit -sim
3  vlib work
4
5  #--------------------------------#
6  #--      Compile Source        --#
7  #--------------------------------#
8  vcom -work work ../source/Counter.vhd
9
10 #--------------------------------#
11 #--     Compile Test Bench     --#
12 #--------------------------------#
13 vcom -work work ../Testbench/TbCounter.vhd
14
15 vsim -t 100ps -novopt work.TbCounter
16 view wave
17
18 do wave.do
19
20 view structure
21 view signals
22
23 #for other Tb
24 run 100 us
25
```

สั่งจบการทำงานที่อาจจะทำค้างไว้ก่อนหน้านี้

สร้าง work folder เพื่อเก็บไฟล์ที่ได้ compile แล้วของ modelsim
(หากมี work อยู่แล้ว จะลบของเดิมทิ้ง และสร้าง folder ใหม่แทน)

Compile HDL code ที่ต้องการ simulate ใส่ work directory

Compile testbench ใส่ work directory

เริ่ม Simulate ตาม Testbench ที่เขียนไว้

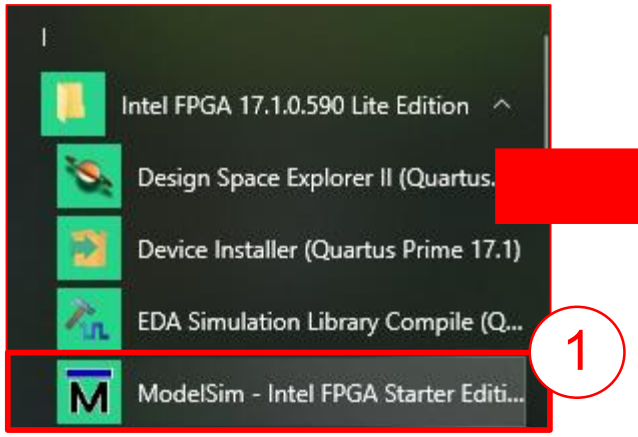
เรียก list ของ waveform ที่เก็บไว้ เพื่อให้สัญญาณเรียงตามลำดับที่ต้องการ

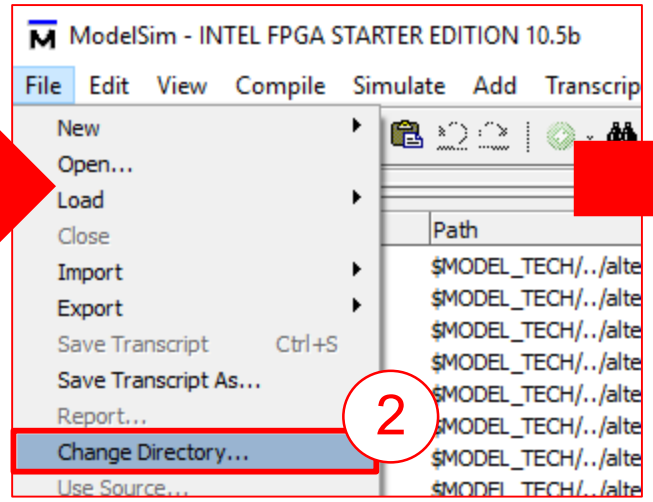
wave.do : อาจจะเริ่มสร้างจากไฟล์เปล่า แล้วค่อยกด save ทับไปอีกที หลังเรียงสัญญาณที่ต้องการเสร็จแล้ว

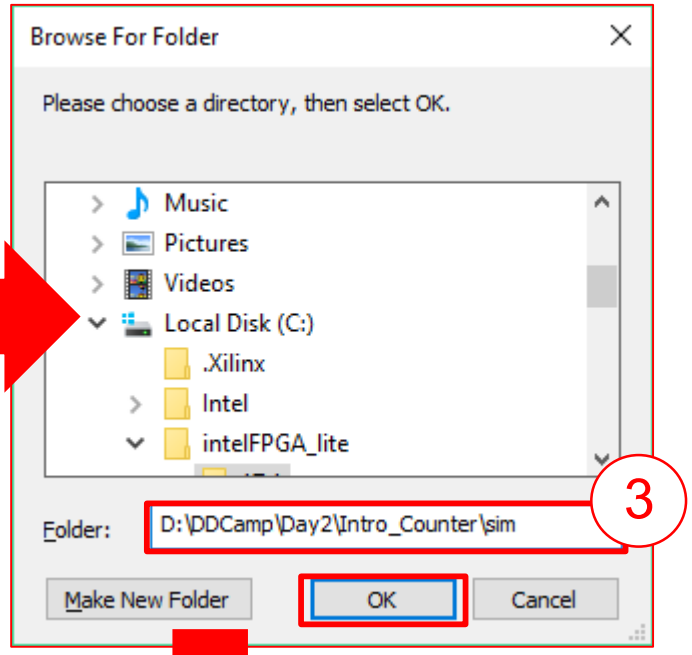
สั่ง run ทั้งหมด 100 us

STEP TO SIMULATE

STEP1: RUN MODELSIM

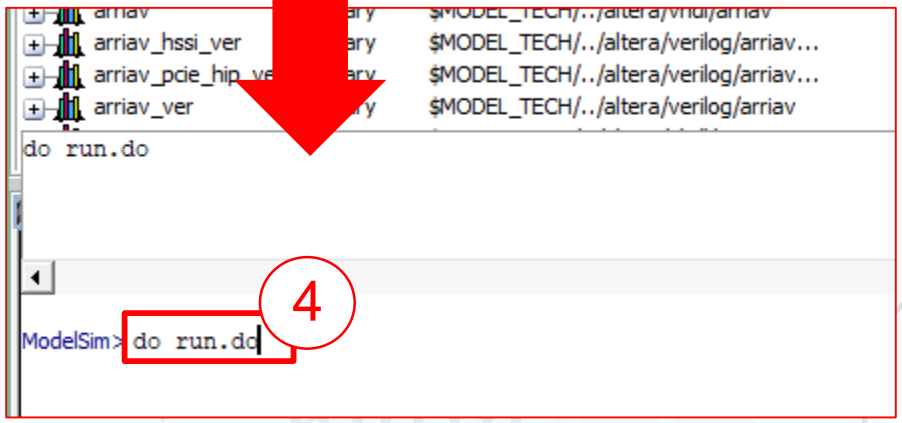




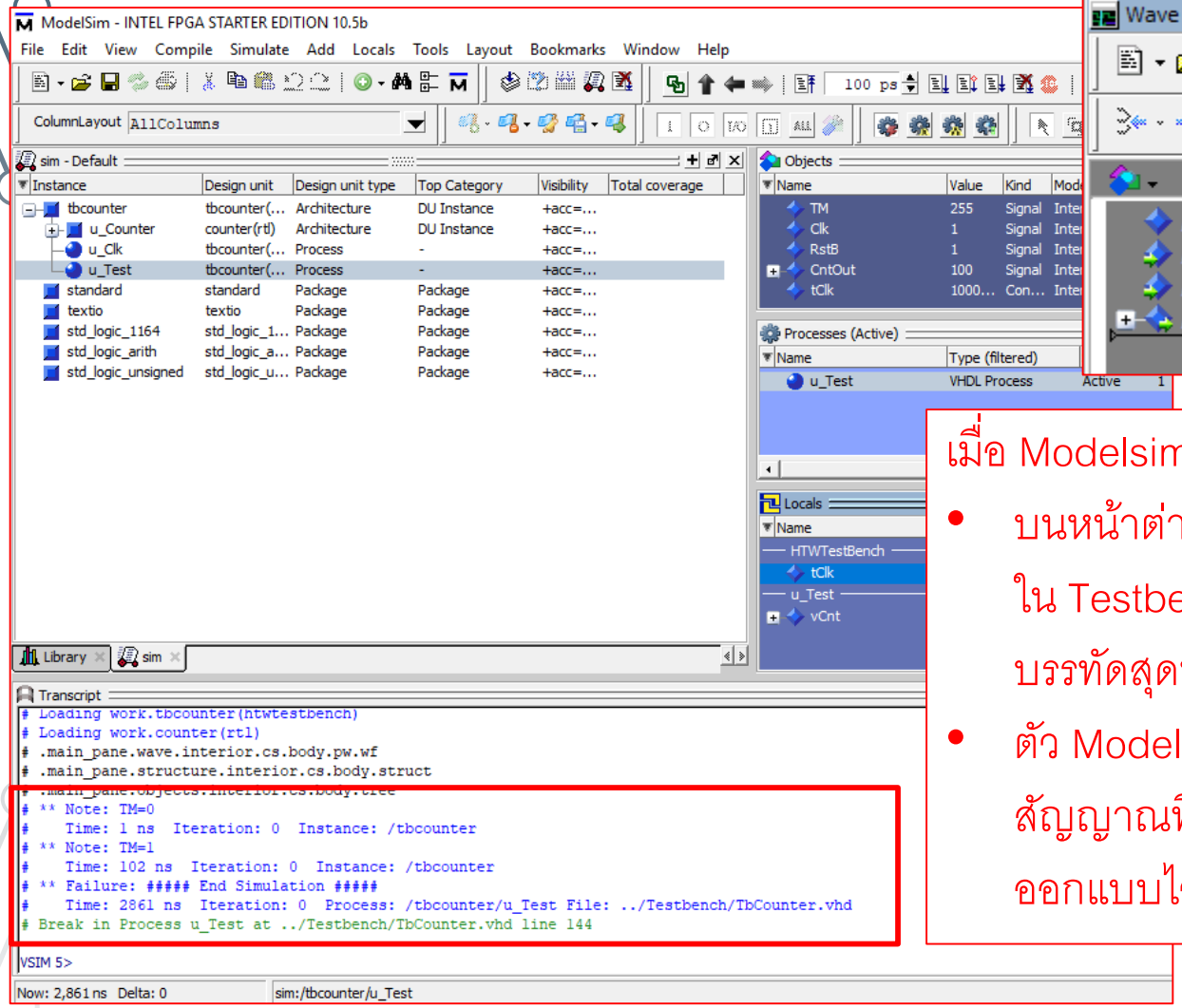


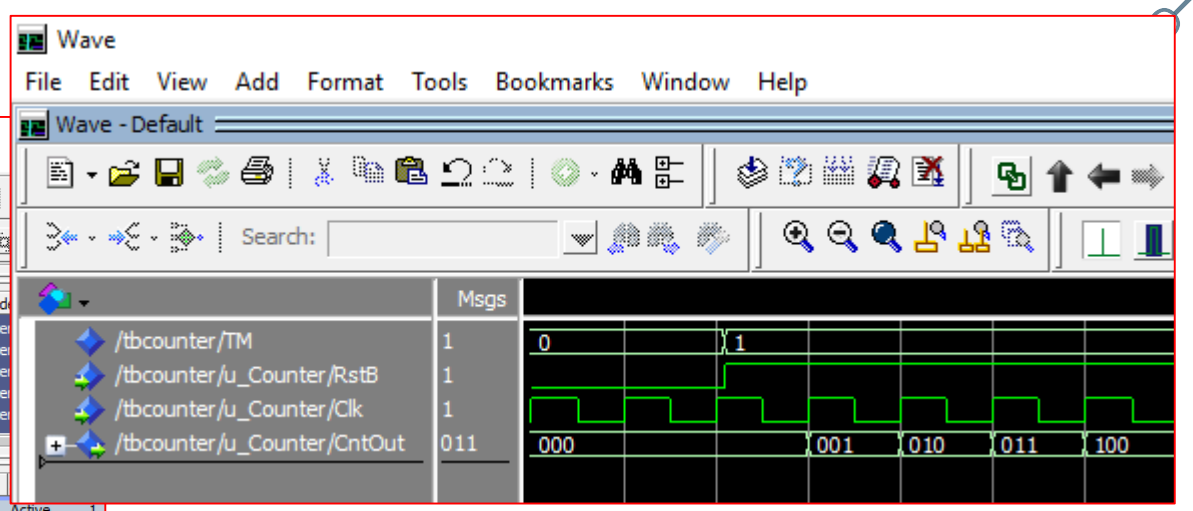
ขั้นตอนทั่วไปสำหรับ simulation

1. เปิดโปรแกรม ModelSim
2. Setup working directory โดยเลือกที่ File -> Change Directory ..
3. เลือกไปที่ sim folder ของโปรเจคตัวอย่าง แล้วกด OK
4. ในหน้าต่าง transcript พิมพ์คำสั่ง do run.do + enter เพื่อเริ่มทำงาน



STEP2: MODELSIM RESULT

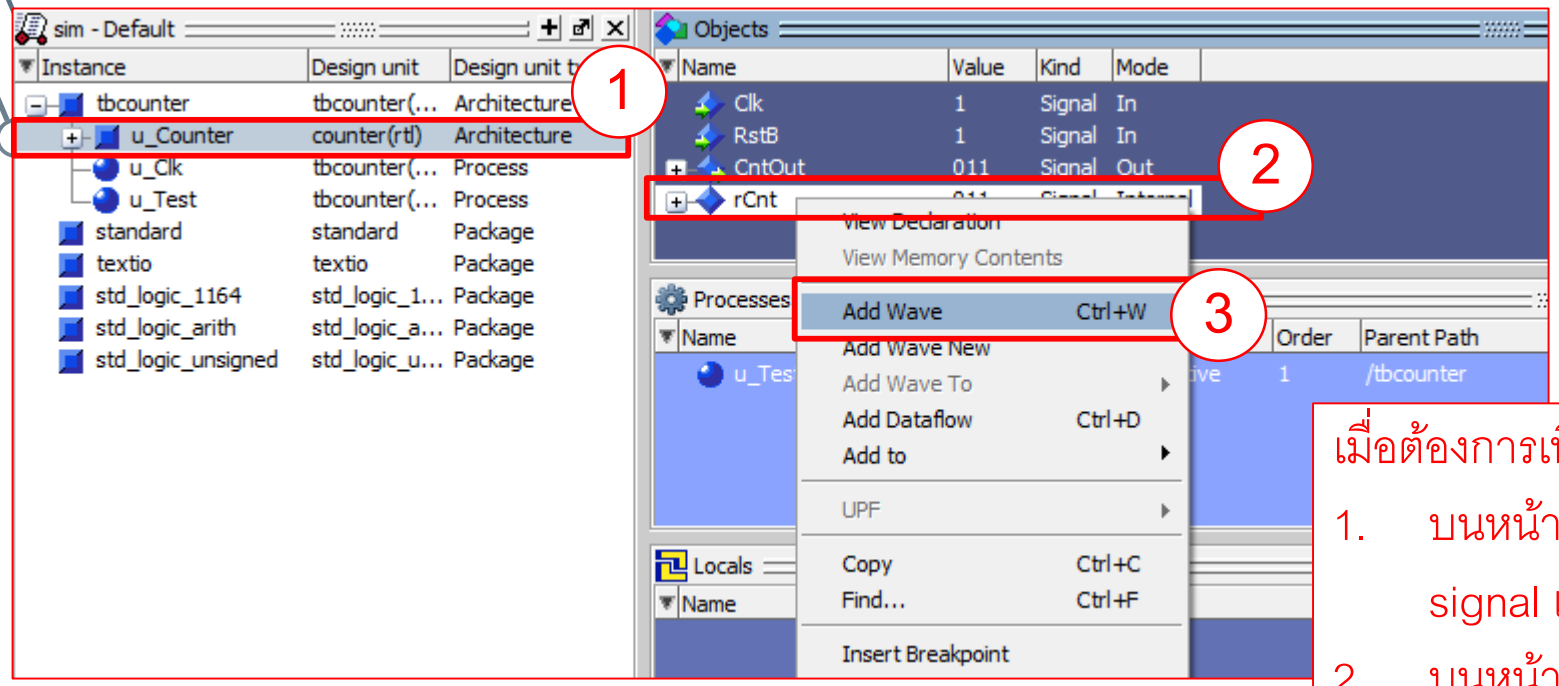




เมื่อ Modelsim ทำงานเสร็จ

- บนหน้าต่าง Transcript จะมีข้อความ End Simulation (เป็นข้อความที่เขียนไว้ใน Testbench code เมื่อจบการทำงาน) แสดงถึงว่า Testbench ทำงานจนถึงบรรทัดสุดท้ายของ Process หลักใน Testbench แล้ว
- ตัว Modelsim จะแสดงหน้าต่าง Wave ขึ้นมาด้วย เพื่อแสดง waveform ของสัญญาณที่เราได้เลือกไว้ สำหรับดู timing diagram ว่าทำงานได้ถูกต้องตามที่ออกแบบไว้ไหม ในตัวอย่างคือ วงจร Count-up ธรรมดา

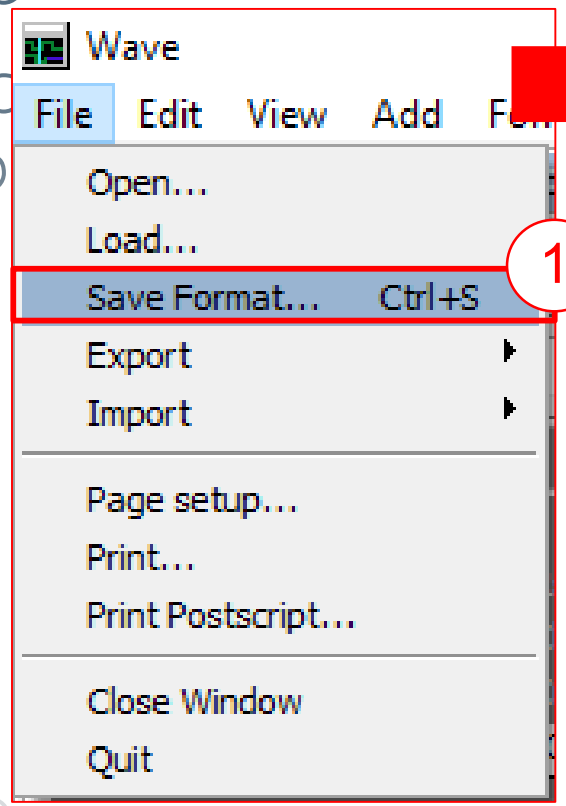
STEP3: ADD SIGNAL TO WAVEFORM

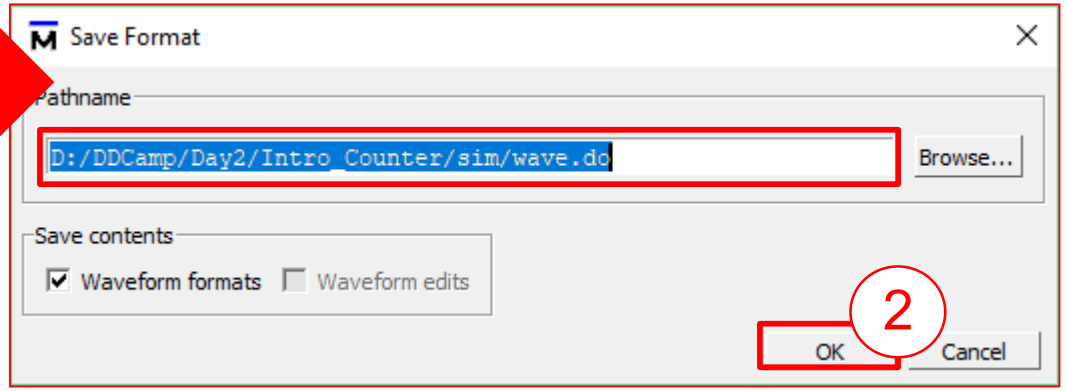


เมื่อต้องการเพิ่มสัญญาณที่จะดู waveform

1. บนหน้าต่าง sim เลือกชื่อ module ที่ต้องการจะดู signal เพิ่มเติม
2. บนหน้าต่าง Objects จะแสดงชื่อสัญญาณทั้งหมดใน module นั้นขึ้นมา เลือกสัญญาณที่ต้องการจะดู
3. คลิ้กขวาแล้วเลือก Add Wave เพื่อเพิ่มสัญญาณที่ต้องการไปที่หน้าต่าง wave

STEP4: SAVE NEW WAVEFORM





หลังจากที่เพิ่มสัญญาณที่ต้องการจะดูไปที่ waveform จนครบแล้ว ให้กด save waveform แบบใหม่ทับไปที่ไฟล์ wave.do เดิม ดังนี้

1. บนหน้าต่าง wave ให้เลือกเมนู File -> Save Format ..
2. Default path ของ wave.do ที่ tool ให้มา จะเป็น working directory/wave.do หากต้องการ save ไปที่ directory อื่น ให้แก้ชื่อ path ให้เรียบร้อย (path นี้ต้องตรงกับ wave.do ที่เรียกใช้ใน run.do ด้วย) แล้วจึงกดปุ่ม OK เพื่อยืนยันการแก้ไข wave.do

จากนั้นลองพิมพ์ do run.do เพื่อลอง simulate วงจรอีกครั้ง หน้าต่าง wave ตัวใหม่ จะมี signal ที่เพิ่มเข้าไป พร้อม waveform แสดง timing diagram เรียบร้อย

EXAMPLE TESTBENCH

NOTE FOR TESTBENCH CODING

สิ่งที่สำคัญก่อนที่เราจะออกแบบ Testbench เพื่อทดสอบการทำงานของวงจรของเรา

- ผู้ออกแบบจะต้องรู้ Specification ของวงจรที่เราต้องการจะตรวจสอบ รู้ข้อจำกัด รู้ลำดับการทำงานที่ถูกต้อง รู้ความสัมพันธ์ของ input และ output ที่ต้องได้มา เพื่อตรวจสอบได้ว่า การทำงานของระบบที่ถูกออกแบบมานี้นั้น ถูกต้องหรือไม่ ดังนั้น ควรจะมี timing diagram แสดงพฤติกรรมของวงจรนี้อยู่ในหัวหรือในมือแล้ว

- การออกแบบ Testbench ที่ดี ควรจะต้องป้อน input แบบต่าง ๆ ให้ครบทุกเงื่อนไขที่เป็นไปได้ หรืออย่างน้อยต้องครอบคลุมการทำงานที่สำคัญ ๆ หรือช่วงที่เป็น critical ที่สุดไว้ เพื่อทดสอบว่า วงจรสามารถทำงานได้ถูกต้อง

TESTBENCH HEADER

TbCounter.vhd

```
45
46  library IEEE;
47  use IEEE.STD_LOGIC_1164.all;
48  use IEEE.STD_LOGIC_ARITH.all;
49  use IEEE.STD_LOGIC_UNSIGNED.all;
50  USE STD.TEXTIO.ALL;
51
52  Entity TbCounter Is
53  End Entity TbCounter;
54
55  Architecture HTWTestBench Of TbCounter Is
56
```

1
2
3

ส่วนประกอบของไฟล์ testbench ใน code ตัวอย่าง ไล่ตั้งแต่บรรทัดแรก ๆ มามีดังนี้

1. การประกาศ library ที่ใช้งาน ซึ่งจะเหมือนกับ library ที่เราใช้ในการออกแบบ hardware ยกเว้นที่มีเพิ่มเข้ามาคือ STD.TEXTIO ซึ่งเป็น library ที่มีฟังก์ชันเพิ่มเติม เช่น การเขียน/อ่านไฟล์ได้ (ใน camp นี้จะไม่ได้แสดงตัวอย่างในส่วนนี้ให้ดู ให้ลองศึกษาเพิ่มเติมได้)
2. ชื่อ Entity ที่จะใช้งาน เรามักจะตั้งชื่อขึ้นต้นด้วย Tb แล้วตามด้วยชื่อ module ที่เราต้องการจะทดสอบ ในที่นี้คือ TbCounter เพื่อบอกว่าเป็นไฟล์ testbench สำหรับทดสอบ module ชื่อ Counter (ชื่อนี้จะต้องตรงกับ script ไฟล์ใน run.do)
3. ส่วนตัวของ Architecture เพื่อบอกว่า เป็นการเริ่มต้น code ของ Testbench หลังจากนี้

SIGNAL DECLARATION IN TESTBENCH

```
-------------------------------------------------------------
-- Constant Declaration
-------------------------------------------------------------
    constant    tClk            : time := 10 ns;
-------------------------------------------------------------
-- Component Declaration
-------------------------------------------------------------
    Component Counter Is
    Port
    (
        RstB        : in    std_logic;
        Clk         : in    std_logic;

        CntOut      : out   std_logic_vector( 7 downto 0 )
    );
    End Component Counter;
-------------------------------------------------------------
-- Signal Declaration
-------------------------------------------------------------
    signal  TM          : integer   range 0 to 65535;

    signal  Clk         : std_logic;
    signal  RstB        : std_logic;
    signal  CntOut      : std_logic_vector( 7 downto 0 );
Begin
```

1 2 3

หลังจากประกาศ Architecture แล้ว ใน code บรรทัดต่อไปจะเป็นการประกาศสัญญาณที่เราจะใช้งานใน testbench รวมถึงการประกาศชื่อ module ที่เราต้องการจะทดสอบในรูปแบบของ Component ดังนี้

1. การประกาศ constant ที่จะใช้งานใน testbench ในที่นี้อย่างน้อยจะเป็นคาบเวลาของสัญญาณ Clk ที่ใช้ทดสอบ
2. การประกาศ component ที่จะใช้งานใน testbench ซึ่งอย่างน้อยต้องประกอบด้วย module ที่เราจะทดสอบ ในที่นี้คือวงจร Counter นั่นเอง แต่ในระบบที่ใหญ่ขึ้น อาจจะมี module มากกว่านี้ เช่น model ที่เราสร้างมา เพื่อต่อกับ input/output ของระบบที่เราจะทดสอบ สำหรับการสร้าง input หรือการตรวจสอบ output
3. การประกาศ list ของสัญญาณที่จะใช้ทั้งหมด อย่างน้อยจะประกอบด้วย input/output ทั้งหมดที่มีของ module ที่เราจะทดสอบ ในที่นี้มีเพิ่มสัญญาณชื่อ TM ซึ่งจะระบุลำดับการ Test ว่ากำลังทำถึง issue ไหนแล้ว

COMPONENT MAPPING IN TESTBENCH

TbCounter.vhd

```
89  ----------------------------------------
90  -- Concurrent signal
91  ----------------------------------------
92
93      u_Clk : Process
94      Begin
95          Clk     <= '1';
96          wait for tClk/2;
97          Clk     <= '0';
98          wait for tClk/2;
99      End Process u_Clk;
100
101     u_Counter : Counter
102     Port map
103     (
104         Clk             => Clk      ,
105         RstB            => RstB     ,
106
107         CntOut          => CntOut
108     );
109
```

1

2

หลังจากคำสั่ง Begin จะเริ่มส่วนที่เป็น HDL code ของ testbench จากการสร้างสัญญาณพื้นฐานคือ Clock และทำ Component mapping เพื่อเชื่อมสัญญาณที่ประกาศใน testbench เข้ากับ module ที่เราจะทดสอบ

1. Process สำหรับสร้าง Clock โดยจะเป็น Clock ที่มีคาบเวลาตามที่ระบุไว้ในค่า tClk ที่ประกาศเป็นค่า constant ไว้ (code ที่มีคำสั่ง wait for แบบนี้ ใช้ใน testbench เท่านั้น ไม่สามารถสังเคราะห์เป็น logic ได้)
2. การ map สัญญาณ บน testbench เข้าไปที่ input/output port ของ component ทีละสัญญาณ โดยด้านซ้ายมือจะเป็นชื่อ port ของ component ส่วนด้านขวามือเป็นชื่อสัญญาณบน testbench (เพื่อความง่ายในตัวอย่าง ไม่ให้สับสนกัน จึงตั้งชื่อทุกสัญญาณเหมือนกับชื่อ port)

TESTBENCH PROCESS

```
u_Test : Process
variable    vCnt    : std_logic_vector( 7 downto 0 );
Begin
    ------------------------------------------
    -- TM=0 : Reset
    ------------------------------------------
    TM <= 0; wait for 1 ns;
    Report "TM=" & integer'image(TM);
    RstB        <= '0';
    wait for 10*tClk;

    ------------------------------------------
    -- TM=1 : Check counter value
    ------------------------------------------
    TM <= 1; wait for 1 ns;
    Report "TM=" & integer'image(TM);

    RstB        <= '1';
    vCnt        := x"00";
    For i in 0 to 258 loop
        Assert (vCnt=CntOut)
        Report "ERROR: Counter is invalid"
        Severity Failure;

        wait until rising_edge(Clk);
        wait for 1 ns;
        vCnt    := vCnt + 1;
    End loop;

    ------------------------------------------------------
    TM <= 255; wait for 1 ns;
    wait for 20*tClk;
    Report "##### End Simulation #####" Severity Failure;
    wait;

End Process u_Test;

End Architecture HTWTestBench;
```

1

2

ส่วนสุดท้ายสำหรับตัวอย่าง เป็น Process สำหรับการทดสอบวงจร Counter

1. ตัว Process โดยทั่วไปจะใช้กำหนดค่า input ให้มีค่าต่าง ๆ เพื่อทดสอบการทำงานของระบบว่าถูกต้องไหม แต่เนื่องจากในตัวอย่างนี้ เป็นวงจร Counter ที่มี input แค่ Clk และ RstB (สัญญาณ Reset) เท่านั้น ตัวอย่าง จึงแบ่งออกเป็น 2 TM คือ TM=0 เป็นช่วงที่ Reset ระบบ และ TM=1 เป็นช่วงที่ Counter เริ่มทำงาน แต่ในตัวอย่างนี้ แสดงการตรวจสอบ output อัตโนมัติว่าทำงานถูกต้องไหม โดยการสร้าง variable ขึ้นมา เพื่อ compare ว่า output จาก Counter นั้นค่อย ๆ นับขึ้นทีละ 1 ตามจังหวะ Clk จริงหรือไม่ ถ้าไม่จริงตัว modelsim จะหยุดการทำงาน (จากคำสั่ง Severity Failure) พร้อมแจ้ง error message (จากคำสั่ง Report)
2. การประกาศ End Architecture เพื่อระบุการจบ coding ของ Testbench

CHALLENGE 1 : VGA GENERATOR TESTBENCH

สร้างสัญญาณ Reset และ Clk ป้อนให้วงจร VGA Generator ที่ให้สร้างใน Day1 Lab
แล้วตรวจสอบดูสัญญาณ output ว่าเป็นไปตามที่ออกแบบไว้ไหม

LAB1: TESTBENCH CODING

EXAMPLE HDL CODE

```
Entity CntUpDwn Is
Port
(
    RstB        : in    std_logic;
    Clk         : in    std_logic;

    CntUpEn     : in    std_logic;
    CntDwnEn    : in    std_logic;

    CntOut      : out   std_logic_vector( 7 downto 0 )
);
End Entity CntUpDwn;
```

การทำงานของ module ชื่อ CntUpDwn

- เมื่อ RstB='0' สัญญาณ CntOut จะถูกเคลียร์ค่าเป็น 0
- เมื่อ CntUpEn='1' สัญญาณ CntOut จะเพิ่มค่าขึ้นทีละ 1 ตามจังหวะขอบขาขึ้นของ Clk
- เมื่อ CntDwnEn='1' สัญญาณ CntOut จะลดค่าลงทีละ 1 ตามจังหวะขอบขาขึ้นของ Clk
- หากสัญญาณ CntUpEn='1' พร้อมกับ CntDwnEn='1' สัญญาณ CntOut จะคงค่าเดิมเหมือนกับกรณีที่ CntUpEn='0' และ CntDwnEn='0'

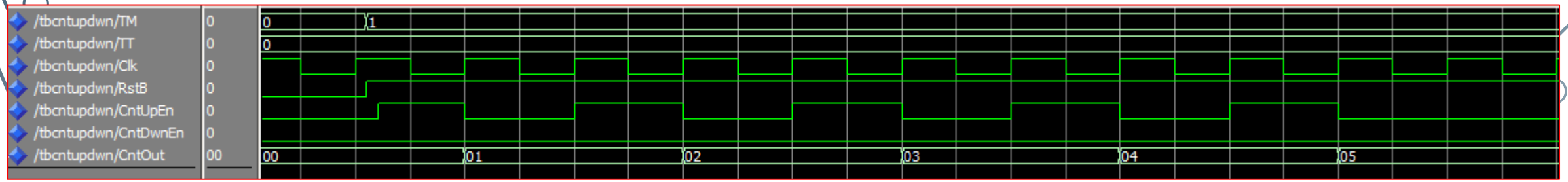
Q1: GENERATE ENABLE FOR 1 CYCLE

```
-----------------------------------------
-- TM=1 : Generate enable for 1 cycle
-----------------------------------------
TM <= 1; TT <= 0; wait for 1 ns;
Report "TM=" & integer'image(TM) & " TT=" & integer'image(TT);

------------------------------
-- Check increment feature

-- Enable 1 clock
CntUpEn <= '1';
wait until rising_edge(Clk);
CntUpEn <= '0';
wait until rising_edge(Clk);

-- Enable 1 clock for 4 times
For i in 0 to 3 loop
    CntUpEn <= '1';
    wait until rising_edge(Clk);
    CntUpEn <= '0';
    wait until rising_edge(Clk);
End loop;
wait for 10*tClk;
```

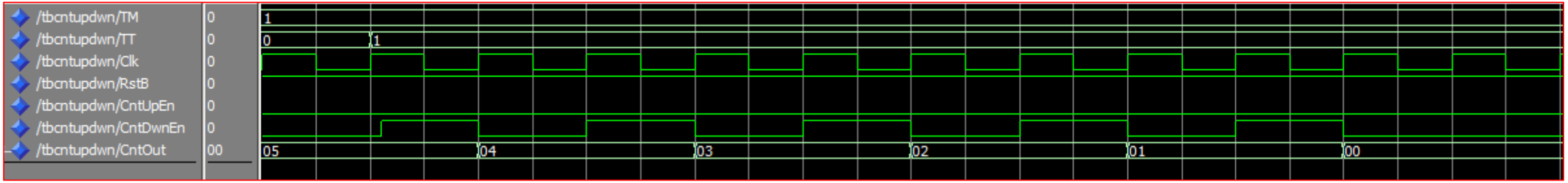
โจทย์

1. Coding TT=0 ตามรูป และลอง simulate ดู waveform ว่าถูกต้องหรือไม่
2. เพิ่ม code ที่ TT=1 โดยแก้ไขจาก CntUpEn เป็นสัญญาณ CntDwnEn แล้วดู waveform ว่าถูกต้องหรือไม่ (เปลี่ยนจากนับขึ้นเป็นนับลง)

WAVEFORM RESULT OF Q1



ที่ TM=1 และ TT=0 สัญญาณ CntOut จะนับขึ้นทีละ 1 ตามจังหวะของ CntUpEn ที่ enable ทีละ 1 cycle



ที่ TM=1 และ TT=1 สัญญาณ CntOut จะนับลงทีละ 1 ตามจังหวะของ CntDwnEn ที่ enable ทีละ 1 cycle

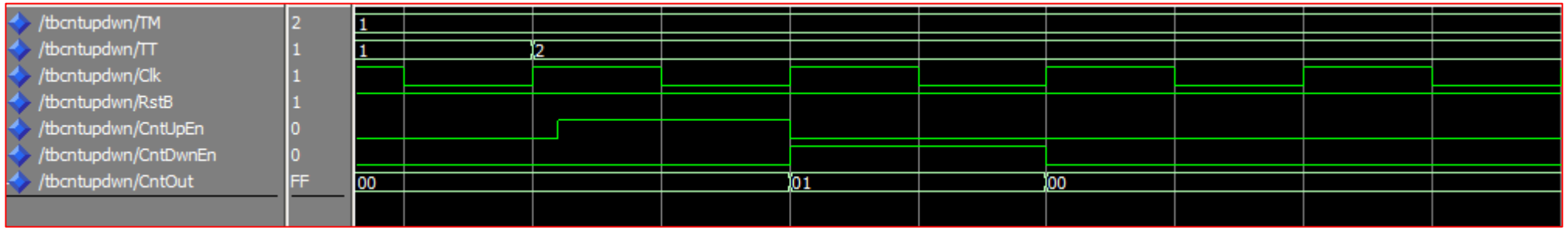
Q2.1: USE FOR LOOP

```
-------------------------------
-- Check increment/decrement feature
TT <= 2; wait for 1 ns;
Report "TM=" & integer'image(TM) & " TT=" & integer'image(TT);

-- Enable 1 clock
CntUpEn  <= '1';
CntDwnEn <= '0';
wait until rising_edge(Clk);
CntUpEn  <= '0';
CntDwnEn <= '1';
wait until rising_edge(Clk);
CntUpEn  <= '0';
CntDwnEn <= '0';
wait for 10*tClk;
```

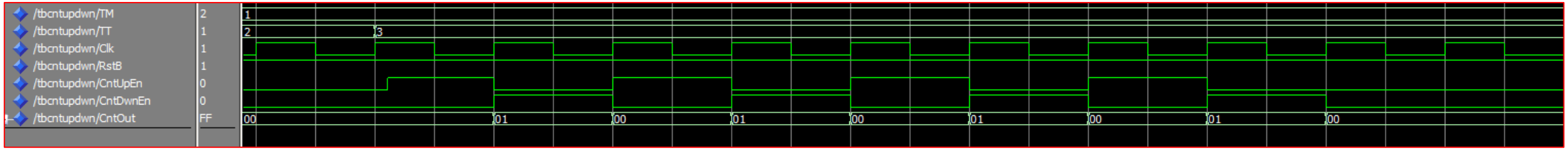
โจทย์

1. Coding TT=2 ตามรูป และลอง simulate ดู waveform ว่าถูกต้องหรือไม่
2. เพิ่ม code ที่ TT=3 โดยให้ทำงานเหมือน code ที่ TT=2 (ตั้งแต่สร้าง CntUpEn='1' อย่างเดียว จนถึงจังหวะที่ทั้ง CntUpEn และ CntDwnEn มีค่าเป็น '0' ทั้งคู่) ทั้งหมด 4 รอบด้วยคำสั่ง For loop แล้วดู waveform ว่าถูกต้องหรือไม่

WAVEFORM RESULT OF Q2.1



ที่ TM=1 และ TT=2 สัญญาณ CntOut จะนับขึ้น/ลง ตามจังหวะของ CntUpEn/CntDwnEn ที่ enable ทีละ 1 cycle



ที่ TM=1 และ TT=3 จะทำงานเหมือน TT=2 แต่ทำทั้งหมด 4 รอบ

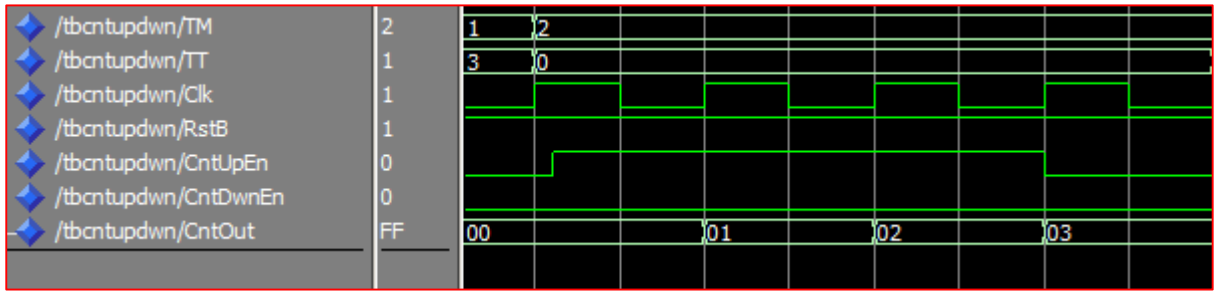
Q2.2: USE FOR LOOP

```
-----------------------------------------
-- TM=2 : Generate enable more than 1 cycle
-----------------------------------------
TM <= 2; TT <= 0; wait for 1 ns;
Report "TM=" & integer'image(TM) & " TT=" & integer'image(TT);

------------------------------
-- Generate multiple cycles by simple method
CntUpEn  <= '1';
wait until rising_edge(Clk);
wait until rising_edge(Clk);
wait until rising_edge(Clk);
CntUpEn  <= '0';
wait until rising_edge(Clk);
wait for 10*tClk;
```

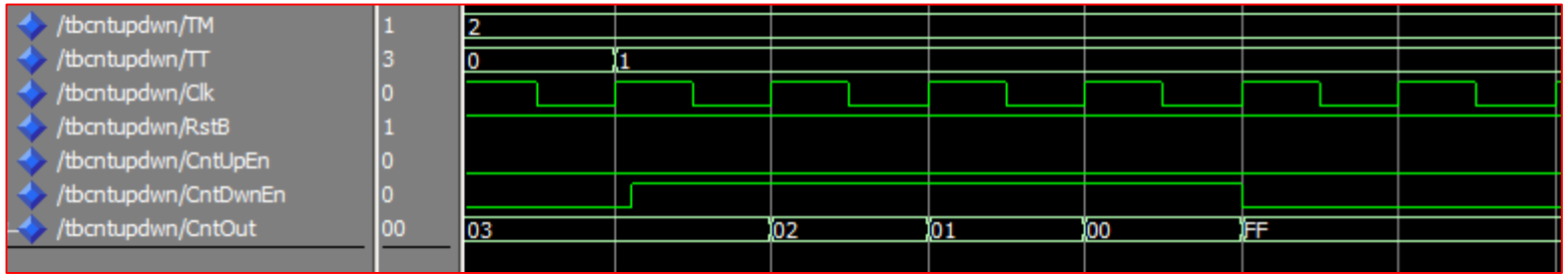
โจทย์

1. Coding TM=2 ตามรูป และลอง simulate ดู waveform ว่าถูกต้องหรือไม่
2. เพิ่ม code ที่ TT=1 แก้ code จาก TT=0 ซึ่งใช้คำสั่ง wait until rising_edge(Clk) ทั้งหมดหลาย ๆ ครั้ง ให้แก้ไขโดยใช้คำสั่ง For loop เพื่อควบคุมสัญญาณ CntDwnEn ให้ set เป็น '1' ทั้งหมด 4 clock จากนั้นตรวจสอบ waveform ว่าถูกต้องหรือไม่

WAVEFORM RESULT OF Q2.2



ที่ TM=2 และ TT=0 สัญญาณ CntOut จะนับขึ้นจาก 0 เป็น 3 เพราะสัญญาณ CntUpEn นั้น set เป็น ‘1’ ทั้งหมด 3 clock



ที่ TM=2 และ TT=1 สัญญาณ CntOut จะนับลงจากค่า 3 จน 0 และกลายเป็น 0xFF (underflow) ไป เพราะสัญญาณ CntDwnEn นั้น set เป็น ‘1’ ทั้งหมด 4 clock

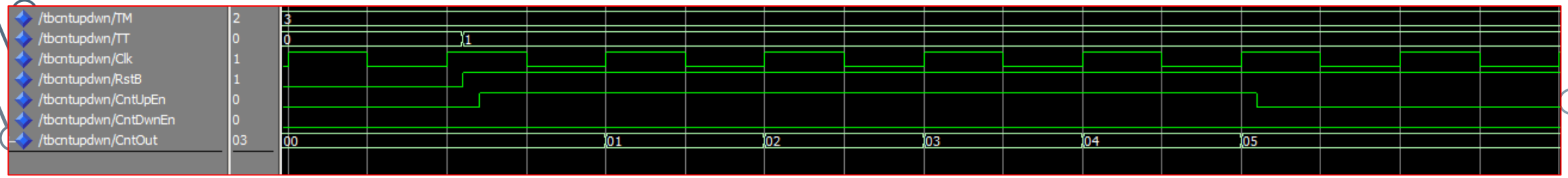
Q3: USE LOOP UNTIL

```
-----------------------------------------
-- TM=3 : Generate signal with output condition
-----------------------------------------
TM <= 3; TT <= 0; wait for 1 ns;
Report "TM=" & integer'image(TM) & " TT=" & integer'image(TT);

-- Reset logic
RstB    <= '0';
wait for 10*tClk;
RstB    <= '1';

-- Count up until CntOut=5
TT <= 1; wait for 1 ns;
Report "TM=" & integer'image(TM) & " TT=" & integer'image(TT);

iTmp    := 0;
Loop
    CntUpEn <= '1';
    iTmp    := iTmp + 1;
    wait until rising_edge(Clk);
    wait for 1 ns; -- Delay to wait CntOut valid
    if ( CntOut=5 ) then
        exit;
    end if;
end loop;
CntUpEn <= '0';
Report "Total CntUpEn cycle=" & integer'image(iTmp);
wait until rising_edge(Clk);
wait for 10*tClk;
```

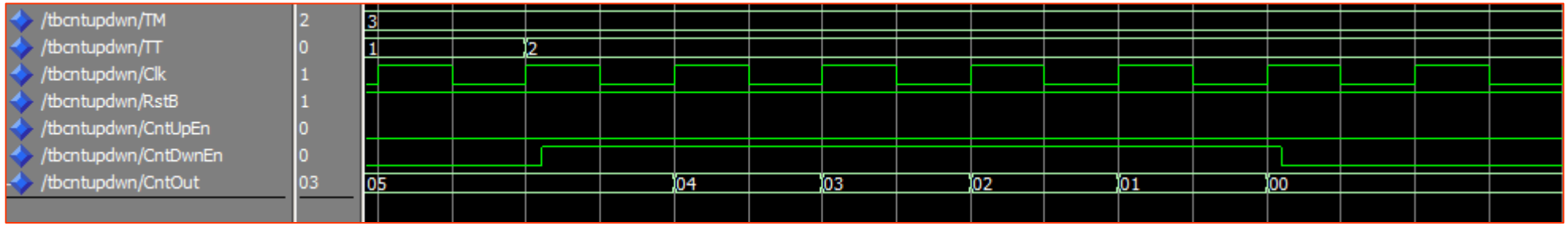
โจทย์

1. Coding TM=3 ตามรูป และลอง simulate ดู waveform ว่าถูกต้องหรือไม่
2. เพิ่ม code ที่ TT=2 โดยปรับ code ให้สร้างสัญญาณ CntDwnEn=‘1’ จนกว่าค่า CntOut จะนับลงจนมีค่าเป็น 0 (ตรวจสอบความถูกต้องโดยดูจาก message ว่า CntDwnEn cycle มีค่าเป็น 5 หรือไม่) พร้อมดู waveform ว่าถูกต้องหรือไม่

WAVEFORM RESULT OF Q3



ที่ TM=3 และ TT=1 สัญญาณ CntUpEn จะถูก set เป็น ‘1’ จนกว่า CntOut จะมีค่าเท่ากับ 5 ดังนั้น CntUpEn จะต้อง set ค่าเป็น ‘1’ ทั้งหมด 5 clock



ที่ TM=3 และ TT=2 จะทำงานเหมือน TT=1 แต่เปลี่ยนจากสัญญาณ CntUpEn เป็น CntDwnEn และตัว CntOut ก็จะนับลงจนกระทั่งมีค่าเป็น 0 ดังนั้นตัว CntDwnEn ต้องทำงานทั้งหมด 5 clock

LAB2: TESTBENCH DESIGN

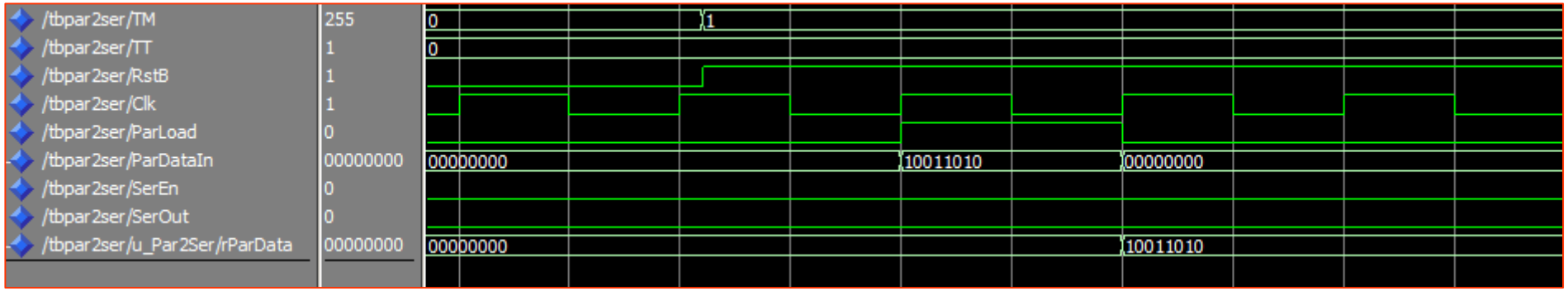
EXAMPLE HDL CODE

```
Entity Par2Ser Is
Port
(
    RstB        : in    std_logic;
    Clk         : in    std_logic;

    ParLoad     : in    std_logic;
    ParDataIn   : in    std_logic_vector( 7 downto 0 );
    SerEn       : in    std_logic;

    SerOut      : out   std_logic
);
End Entity Par2Ser;
```

การทำงานของ module ชื่อ Par2Ser

- เมื่อ RstB=‘0’ สัญญาณ SerOut จะถูกเคลียร์ค่าเป็น 0
- เมื่อ ParLoad=‘1’ สัญญาณ ParDataIn ทั้ง 8 bit จะถูกอ่านไปเก็บไว้ที่ Shift register ขนาด 8-bit ที่อยู่ภายใน และเฉพาะ LSB (bit 0) จะส่งออกมาที่ SerOut
- เมื่อ SerEn=‘1’ ตัว Shift register ภายในจะค่อย ๆ shift ขวาเพื่อทยอยส่ง data ออกมาทาง SerOut ทีละ bit เริ่มจาก LSB (bit 0) ไปจนถึง MSB (bit 7) หลังจากนั้นหากยังมี SerEn เข้ามา สัญญาณที่ส่งออกมาจะมีค่าเป็น ‘0’ หมด
- หากสัญญาณ ParLoad และ SerEn มีค่าเป็น ‘1’ พร้อมกัน ให้เลือกทำงานโหลดค่า (ParLoad มี Priority สูงกว่า SerEn)

TM=1 TT=0: TEST PARLOAD

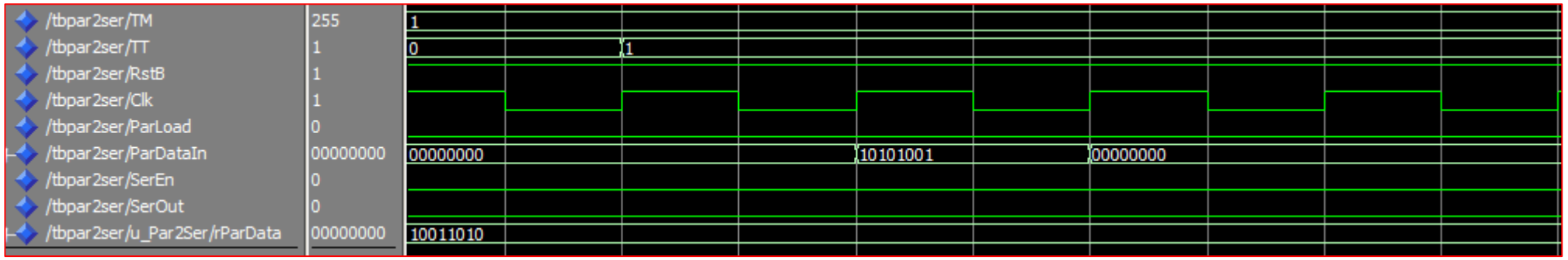


โจทย์

เขียน code ช่วง TM=1 TT=0 โดยมีลำดับดังนี้

1) ให้ ParLoad=‘1’ และ ParDataIn=x”9A” เป็นเวลา 1 clock cycle
2) ให้ ParLoad เปลี่ยนค่าเป็น ‘0’ และ ParDataIn=x”00”
3) ตรวจสอบดู waveform ว่า rParData ซึ่งเป็นสัญญาณภายในมีค่าถูกต้องคือ เปลี่ยนเป็น x”9A” หลังจากเจอสัญญาณ ParLoad และ hold ค่านี้ไว้จนจบ TM=1 และ TT=0

TM=1 TT=1: ADDITIONAL TEST FOR PARLOAD

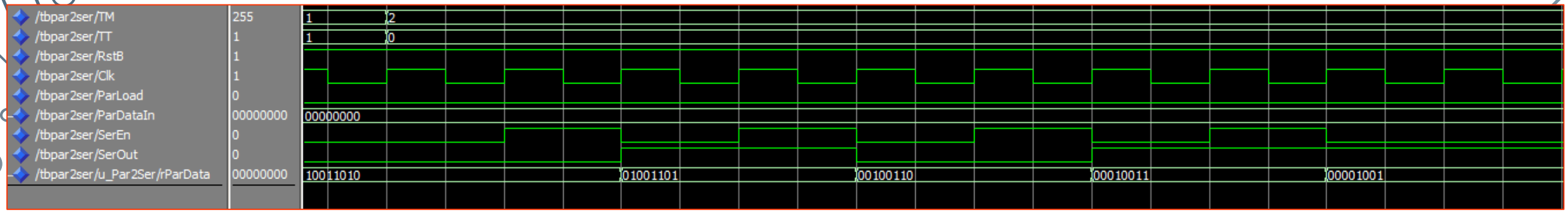


โจทย์

เขียน code ช่วง TM=1 TT=1 โดยมีลำดับดังนี้

1) ให้ ParLoad='0' และเปลี่ยนค่า ParDataIn=x"A9" เป็นเวลา 1 clock cycle
2) ให้ ParDataIn=x"00"
3) ตรวจสอบดู waveform ว่า rParData ซึ่งเป็นสัญญาณภายในมีค่าถูกต้องคือ ต้องคงค่าเป็น x"9A" ตลอด ไม่เปลี่ยนค่าตาม ParDataIn

TM=2 TT=0: TEST SEREN

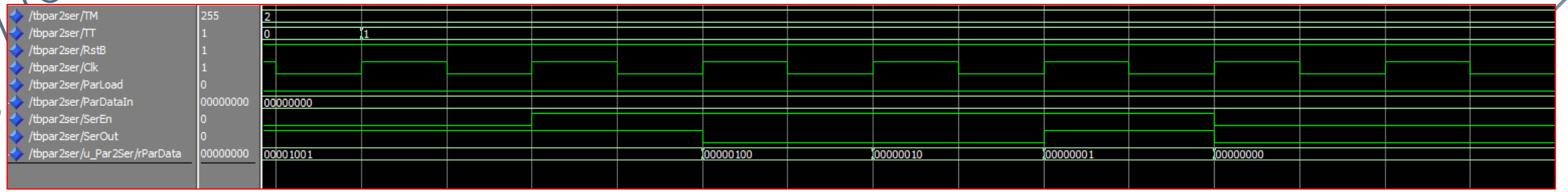


โจทย์

เขียน code ช่วง TM=2 TT=0 โดยมีลำดับดังนี้

1) ให้ SerEn='1' สลับกับ SerEn='0' อย่างละ 1 clock cycle ทั้งหมด 4 ครั้ง
2) ตรวจสอบดู waveform ว่า rParData ซึ่งเป็นสัญญาณภายในมีค่าถูกต้องคือ สัญญาณจะ shift ไปทางขวาทีละ 1 bit หลังจากได้สัญญาณ SerEn มา และ MSB ของ rParData จะถูก fill ด้วยค่า '0'
3) ตรวจสอบดู SerOut ว่าเป็นค่าที่ถูก shift ออกมาจากค่าตั้งต้น โดยเริ่มจาก LSB ตามจังหวะ SerEn='1' เช่น เมื่อค่าเริ่มต้น 9A และมี SerEn='1' 4 clock สัญญาณ SerOut ที่ส่งออกมาจะมีค่า 0 1 0 1 (ตัว A แบบถอยหลัง)

TM=2 TT=1: ADDITIONAL TEST FOR SEREN

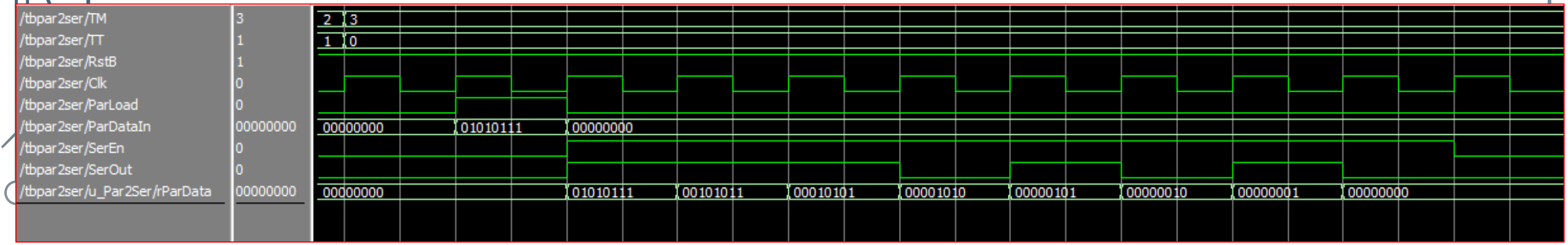


โจทย์

เขียน code ช่วง TM=2 TT=1 โดยมีลำดับดังนี้

1) ให้ SerEn='1' กว้าง 4 clock cycles แล้วเคลียร์ค่าเป็น '0' เพื่อ shift ข้อมูลอีก 4 bit ที่เหลือออกมา
2) ตรวจสอบดู waveform ว่า rParData ซึ่งเป็นสัญญาณภายในมีค่าถูกต้องคือ สัญญาณจะ shift ไปทางขวาทีละ 1 bit ตามสัญญาณ SerEn='1' ค่าสุดท้ายกลายเป็นค่า 0 หมด
3) ตรวจสอบดู SerOut ว่ามีค่าเปลี่ยนตามจังหวะ SerEn โดยส่งค่า 4 bit ที่เหลืออยู่คือ 9 ออกมาตามลำดับจาก LSB ได้แก่ 1 0 0 1 (ตัว 9 แบบถอยหลัง)

TM=3 TT=0: LOAD AND SHIFT CONTINUOUSLY

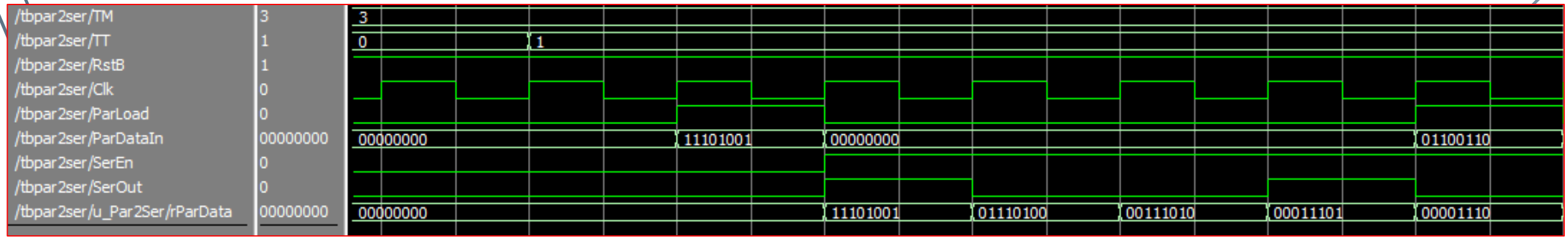


โจทย์

เขียน code ช่วง TM=3 TT=0 โดยมีลำดับดังนี้

1) ให้ ParLoad=‘1’ พร้อม ParDataIn=x”57” เป็นเวลา 1 clock
2) ให้ ParLoad=‘0’ พร้อม ParDataIn=x”00” และให้ SerEn=‘1’ เพื่อเริ่ม shift data ทันที หลังจาก load ค่าใหม่มาแล้ว
3) ให้ SerEn=‘1’ ทั้งหมด 8 clock เพื่อ shift data ทั้งหมดออกมา แล้วจึงเปลี่ยนค่ากลับเป็น ‘0’
4) ตรวจสอบดู waveform ว่า rParData ซึ่งเป็นสัญญาณภายในมีค่าถูกต้องคือ โหลดค่า ParDataIn เข้าไปเมื่อเจอ ParLoad=‘1’ และหลังจากนั้นจะ shift ไปทางขวาทีละ 1 bit ตามจังหวะ SerEn=‘1’ จนครบ ทำให้ค่าสุดท้ายมีค่าเป็น 0 หมด
5) ตรวจสอบดู SerOut ว่ามีค่าเปลี่ยนตามจังหวะ SerEn=‘1’ โดยส่งค่า x”57” ออกมา เริ่มจาก LSB ได้แก่ 1 1 1 0 1 0 1 0

TM=3 TT=1: LOAD AND SHIFT AT THE SAME TIME (1)

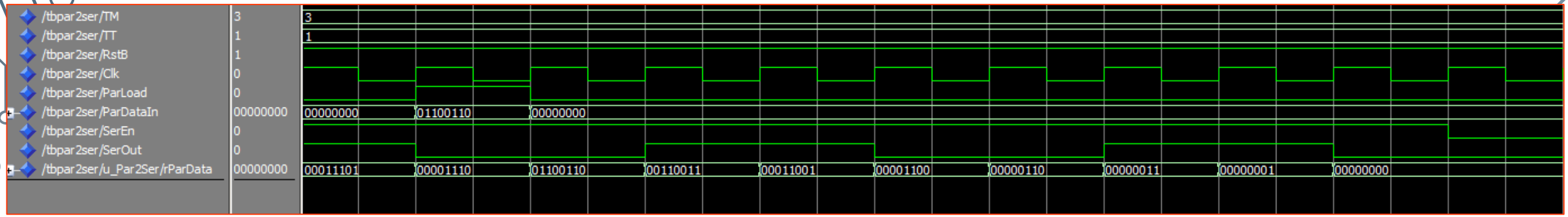


โจทย์

เขียน code ช่วง TM=3 TT=1 โดยมีลำดับดังนี้

1) ให้ ParLoad=‘1’ พร้อม ParDataIn=x”E9” เป็นเวลา 1 clock
2) ให้ ParLoad=‘0’ พร้อม ParDataIn=x”00” และให้ SerEn=‘1’ เพื่อเริ่ม shift data ออกมาทีละ bit
3) ให้ SerEn=‘1’ รวมทั้งหมด 4 clock
4) หลังจากนั้นให้ค่า ParLoad เปลี่ยนเป็น ‘1’ โดยที่สัญญาณ SerEn ยังคงมีค่าเดิมอยู่ (‘1’)
5) ตรวจสอบดู waveform ว่า rParData และ SerOut ว่ามีการทำงานที่ถูกต้อง (load และ shift data ออกมา 4 bit)

TM=3 TT=1: LOAD AND SHIFT AT THE SAME TIME (2)



โจทย์

เขียน code ต่อจากข้อที่แล้วคือ

1) ให้ ParLoad=‘1’ พร้อม ParDataIn=x”66” เป็นเวลา 1 clock โดยที่ SerEn=‘1’ อยู่ เพื่อทดสอบว่า rParData ให้ Priority กับการโหลดข้อมูลใหม่
2) ให้ ParLoad=‘0’ พร้อม ParDataIn=x”00” โดย SerEn=‘1’ เหมือนเดิมไปอีก 8 clock เพื่อ shift data ทั้งหมดออกมา
3) ตรวจสอบดู waveform ว่า rParData และ SerOut ว่ามีการทำงานที่ถูกต้อง (โหลดค่าใหม่เป็น x”66” หลังจากเจอ load และเริ่ม shift data ทั้ง 8 bit ออกมา เรียงลำดับคือ 0 1 1 0 0 1 1 0)

LAB3: TESTBENCH FOR DEBUG

EXAMPLE HDL CODE

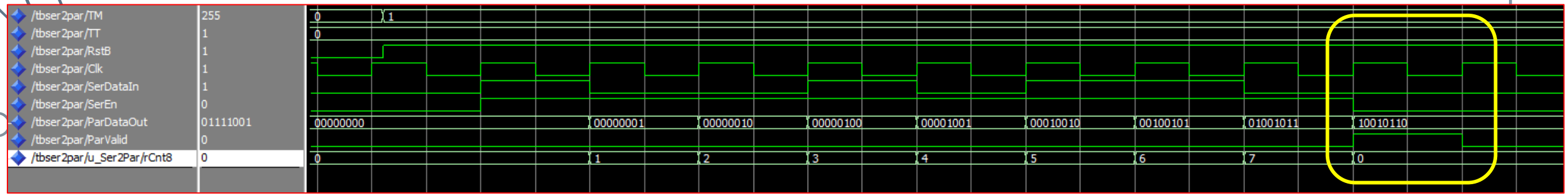
```
Entity Ser2Par Is
Port
(
    RstB        : in    std_logic;
    Clk         : in    std_logic;

    SerDataIn   : in    std_logic;
    SerEn       : in    std_logic;

    ParDataOut  : out   std_logic_vector( 7 downto 0 );
    ParValid    : out   std_logic
);
End Entity Ser2Par;
```

การทำงานของ module ชื่อ Ser2Par

- เมื่อ RstB='0' สัญญาณ ParDataOut และ ParValid จะถูกเคลียร์ค่าเป็น 0
- เมื่อ SerEn='1' สัญญาณ SerDataIn จะถูก shift เข้าไปเก็บใน register ภายในขนาด 8 bit โดยข้อมูลตัวแรกจะถูกเก็บไว้ที่ MSB (bit ที่ 7) ไล่ไปจนถึงตัวสุดท้ายจะเก็บไว้ที่ LSB (bit ที่ 0)
- ตัว logic ภายใน ออกแบบด้วยการใช้ shift register ค่อย ๆ รับข้อมูลเข้ามา และ shift ไปทางซ้าย เพื่อให้ข้อมูลที่เข้ามาก่อนเลื่อนไปอยู่ตำแหน่ง bit ที่สูงขึ้นไป จนครบ 8 ตัว
- เมื่อได้ data ครบ 8 ตัว ข้อมูลทั้ง 8 bit จะส่งออกมาที่ ParDataOut พร้อมทั้งกระดกสัญญาณ ParValid='1' เป็นเวลา 1 Clk เพื่อบอกว่าได้ข้อมูลครบแล้ว เอาไปใช้งานได้ หลังจากนั้นจะเริ่มต้นรอบการทำงานใหม่

TM=1 TT=0: TEST SEREN

จังหวะสุดท้าย ParDataOut จะมีค่าเท่ากับ SerDataIn ที่ส่งมา 8 Clk ต่อกัน คือ 10010110 พร้อมกับ ParValid='1' เป็นเวลา 1 Clk



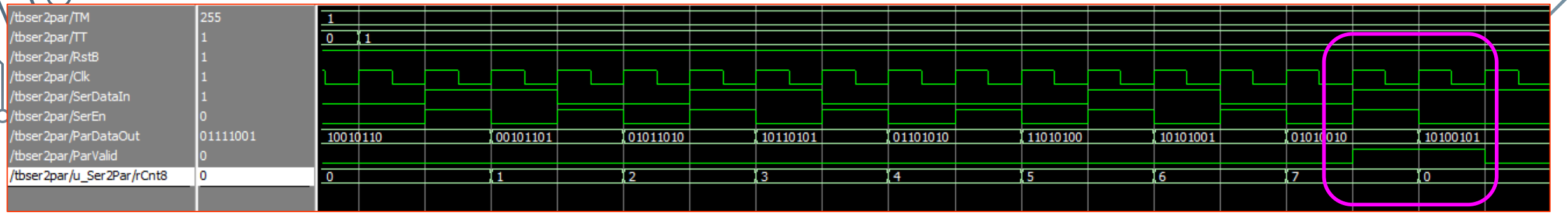
โจทย์

เขียน code ช่วง TM=1 TT=0 โดยมีลำดับดังนี้

1) ให้ SerEn='1' ทั้งหมด 8 Clk และส่งค่า SerDataIn ทั้งหมด 8 ค่า ให้รวมกันกลายเป็น x"96" โดยเริ่มจาก bit ที่ 7 ไล่ไปจนถึง bit ที่ 0 ตามลำดับ (1 0 0 1 0 1 1 0)
2) ให้ SerEn='0' หลังจากจบ Clk ที่ 8
3) ตรวจสอบดู waveform ว่า ParDataOut นั้นได้ค่าออกมาถูกต้องตามที่ส่งเข้าไปไหม (x"96") และ ParValid มีค่าเป็น '1' ทั้งหมด 1 Clk หรือไม่

TM=1 TT=1: TEST SEREN (NOT CONTINUOUS)

ParValid=‘1’ ทั้งหมด 2 Clk ผิด!!!



โจทย์

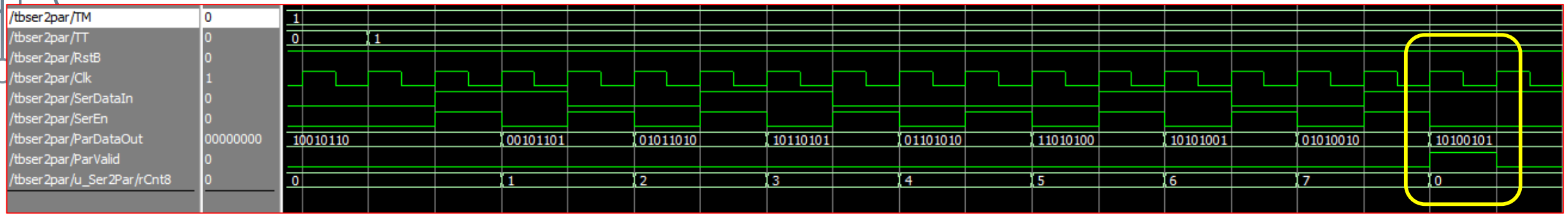
เขียน code ช่วง TM=1 TT=1 โดยมีลำดับดังนี้

1) ให้ SerEn=‘1’ สลับกับ ‘0’ อย่างละ 1 cycle ทั้งหมด 8 รอบ และทยอยส่งค่า SerDataIn ทั้งหมด 8 ค่า ตามจังหวะ SerEn=‘1’ ให้รวมกันกลายเป็น x”A5” โดยเริ่มจาก bit ที่ 7 ไล่ไปจนถึง bit ที่ 0 ตามลำดับ (1 0 1 0 0 1 0 1)
2) ให้ SerEn=‘0’ หลังจากส่งครบ 8 รอบ
3) ตรวจสอบดู waveform ว่า ParDataOut นั้นได้ค่าออกมาถูกต้องตามที่ส่งเข้าไปไหม (x”A5”) และ ParValid มีค่าเป็น ‘1’ ทั้งหมด 1 Clk หรือไม่

*** จะพบว่า code ที่ให้ไว้สร้างสัญญาณ ParValid=‘1’ ทั้งหมด 2 Clk แทนที่จะเป็น 1 Clk *** **(ทำงานผิด)**

โจทย์เพิ่มเติม : หาที่ผิดใน Ser2Par.vhd และแก้ไขให้ถูกต้อง

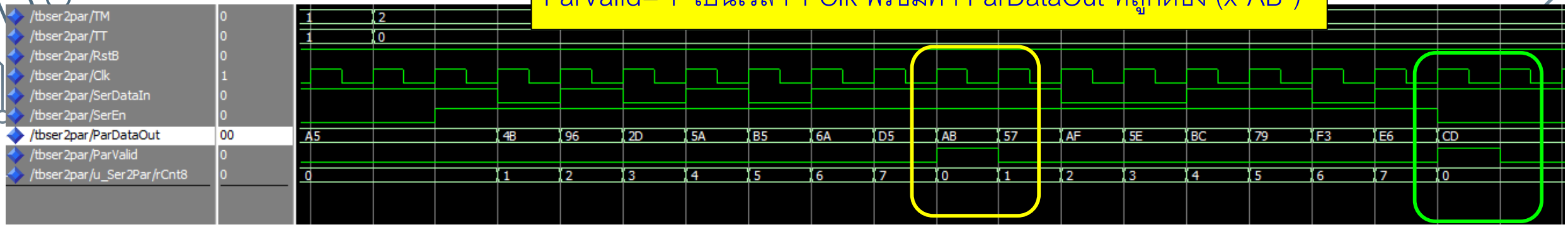
TM=1 TT=1: CORRECT WAVEFORM



ParValid=‘1’ เป็นเวลา 1 Clk พร้อมค่า ParDataOut ที่ถูกต้อง (x”A5”)

TM=2 TT=0: TEST SEREN FOR 16 CLOCK

ParValid=‘1’ เป็นเวลา 1 Clk พร้อมค่า ParDataOut ที่ถูกต้อง (x”AB”)



ParValid=‘1’ เป็นเวลา 1 Clk พร้อมค่า ParDataOut ที่ถูกต้อง (x”CD”)

โจทย์

เขียน code ช่วง TM=2 TT=0 โดยมีลำดับดังนี้

1) ให้ SerEn=‘1’ ทั้งหมด 16 Clk และส่งค่า SerDataIn ทั้งหมด 16 ค่า ให้รวมกันกลายเป็น x”ABCD” โดยเริ่มจาก bit ที่ 15 ไล่ไปจนถึง bit ที่ 0 ตามลำดับ (1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 1)
2) ให้ SerEn=‘0’ หลังจากจบ Clk ที่ 16
3) ตรวจสอบดู waveform ว่า ParDataOut นั้นได้ค่าออกมาถูกต้องตามที่ส่งเข้าไปไหม (x”AB” และ x”CD” ตามลำดับ) และ ParValid มีค่าเป็น ‘1’ ทั้งหมด 2 ครั้ง เมื่อได้ data ครบ 8 ตัว และ 16 ตัว ตามลำดับ

TM=2 TT=1: TEST SEREN FOR 16 CLOCK (NOT CONTINUOUS)

โจทย์

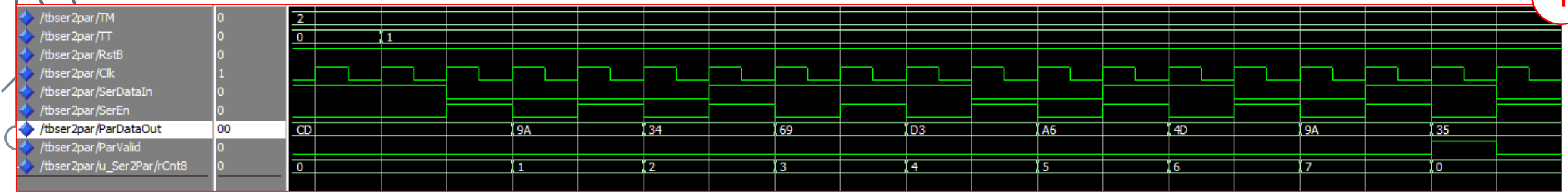
เขียน code ช่วง TM=2 TT=1 (รวม TM=1 TT=2 และ TM=2 TT=0 เข้าด้วยกัน โดยการส่งข้อมูล 16 ตัวต่อเนื่องกัน แต่ SerEn จะมีค่าเท่ากับ ‘1’ และ ‘0’ สลับกันอย่างละ 1 Clk) ดังต่อไปนี้

1) ให้ SerEn=‘1’ สลับกับ ‘0’ อย่างละ 1 cycle ทั้งหมด 16 รอบ และทยอยส่งค่า SerDataIn ทั้งหมด 16 ค่า ตามจังหวะ SerEn=‘1’ ให้รวมกันกลายเป็น x”3579” โดยเริ่มจาก bit ที่ 15 ไล่ไปจนถึง bit ที่ 0 ตามลำดับ (0 0 1 1 0 1 0 1 0 1 1 1 1 0 0 1)
2) ให้ SerEn=‘0’ หลังจากส่งครบ 16 ตัว
3) ตรวจสอบดู waveform ว่า ParDataOut นั้นได้ค่าออกมาถูกต้องตามที่ส่งเข้าไปไหม (x”35” และ x”79” ตามลำดับ) และ ParValid มีค่าเป็น ‘1’ ทั้งหมด 2 ครั้ง เมื่อได้ data ครบ 8 ตัว และ 16 ตัว ตามลำดับ

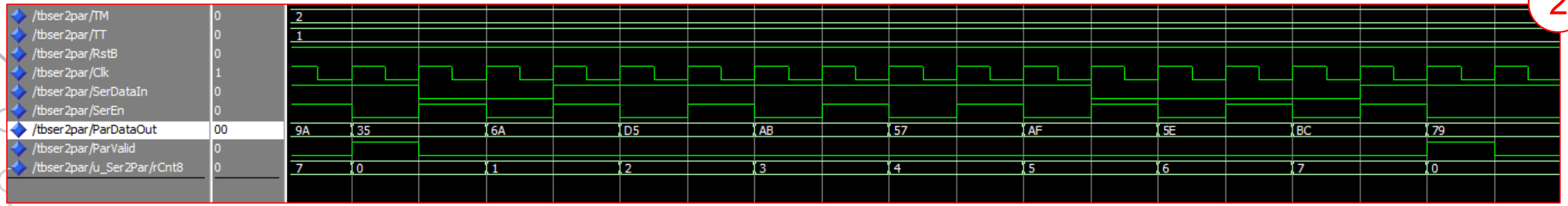
*** จะต้องแก้ code ที่ให้ไว้ ถึงจะได้ ParValid=‘1’ ทั้งหมด 1 clock ตามที่ต้องการ หากไม่แก้ จะได้ 2 clock ซึ่งผิดอยู่ ***

TM=2 TT=1: TEST SEREN (NOT CONTINUOUS) WAVEFORM

1



2



รูป waveform ที่ถูกต้องใน TM=2 TT=1 เพื่อรับข้อมูลทั้งหมด 16 ตัวต่อกัน โดย data จะถูกส่งมาทีละตัว ในทุก ๆ 2 clock

CHALLENGE 2 : UPDATE SER2PAR/PAR2SER

1. แก้ code ของ Ser2Par และ Par2Ser โดยลบ SerEn ทิ้ง และให้แก้ code ให้ shift ข้อมูลอัตโนมัติทุก ๆ 16 Clock แทน และลองตรวจสอบโดยใช้ testbench ดูว่าทำงานถูกต้องไหม
2. แก้ code เหมือนข้อ 1 แต่ปรับให้ shift ข้อมูลอัตโนมัติทุก ๆ 100 Clock แทน และลองตรวจสอบโดยใช้ testbench ดูว่าทำงานถูกต้องไหม

Hint: ย้ายสัญญาณ SerEn จาก port ที่รับภายนอก ไปเป็นสัญญาณภายใน และลองสร้างวงจรควบคุมสัญญาณ SerEn เพื่อจะได้ควบคุมจังหวะการ shift ข้อมูล เป็นทุก ๆ 16 Clock หรือ 100 Clock ตามที่โจทย์กำหนด

CHALLENGE 3 :
TESTPATT SIMULATION

ลองออกแบบ testbench และลอง simulate เพื่อดูการทำงาน waveform และ timing diagram ของ TestPatt

Hint: อ่านบทความเพิ่มเติมของ TestPatt รวมถึงตัวอย่าง testbench ได้ที่

https://forfpgadesign.wordpress.com/2017/12/11/10-2-example-design2/

Q & A

https://www.facebook.com/DigitalDesignThailand/

https://forfpgadesign.wordpress.com/