NCTU-EE IC LAB – Spring 2018

Lab07 Exercise

Design: Cross Domain Clock (CDC)

Data Preparation

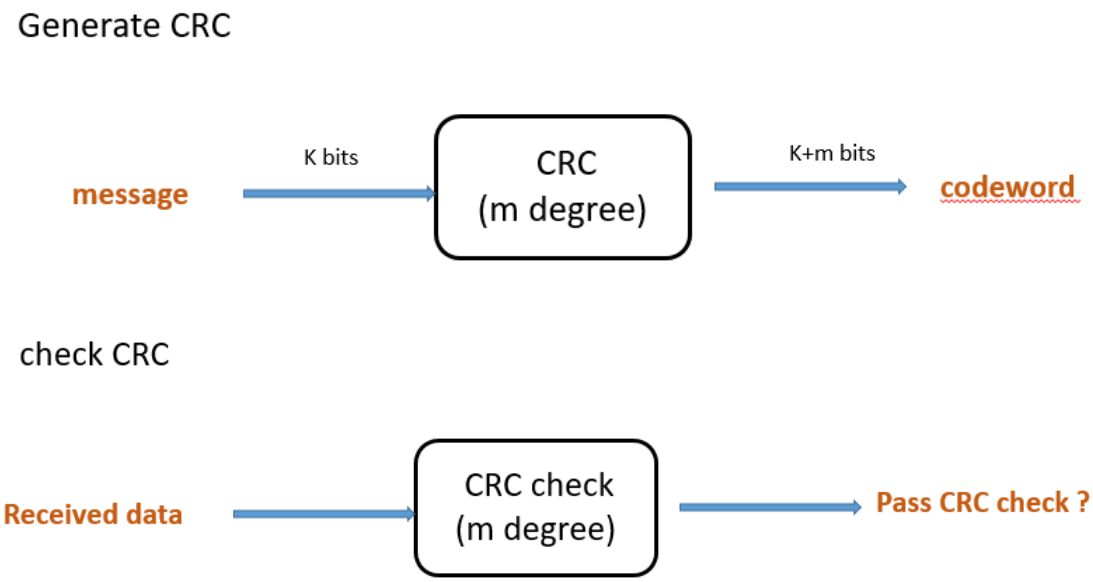
1. Extract test data from TA's directory:
   % tar xvf ~iclabta01/Lab07.tar

Basic Concept

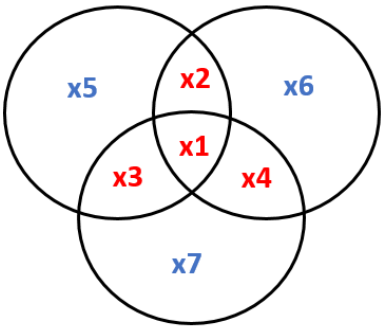
This lab will give you a basic concept about cross domain clock and Error Correction (Detection) code.

A cyclic redundancy check(CRC) is an error-detection code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a **polynomial division** of their contents. On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption.

A (7,4) hamming code is an error-correction code which uses the syndrome to correct the single error in a codeword.



(7,4) Hamming code



Codeword

x1	x2	x3	x4	x5	x6	x7

Legal codeword: the summation in each circle must be an even number.
Illegal codeword: there is an odd summation in a circle.

Example1 for generate the CRC:

The polynomial for the standard is x^3+x+1

message : 1 0 1 0

The degree of polynomial is 3, so we attach extra 3 zero to message

1 0 1 0 0 0 0

The dividend is 1 0 1 0 0 0 0

The divisor is the coefficient of polynomial: 1 0 1 1

Here we replace the subtraction to XOR when we do division.

```
                1 0 0 1
      ---------------
1 0 1 1 ) 1 0 1 0 0 0 0
          1 0 1 1
          -------
          0 0 1 0
          0 0 0 0
          -------
            0 1 0 0
            0 0 0 0
            -------
              1 0 0 0
              1 0 1 1
              -------
                0 1 1
```

Finally, we get our CRC code in remainder: 0 1 1, so codeword is 1 0 1 0 0 1 1

Example2 for detecting the existence of error (CRC check):

The polynomial for the standard is x^3+x+1

Received data: 1 0 1 0 0 1 1

We divide received data by polynomial to check whether is zero or not.

If the remainder is zero, we pass CRC check, otherwise we fail.

The dividend is 1 0 1 0 0 1 1

The divisor is the coefficient of polynomial: 1 0 1 1

Here we also replace the subtraction to XOR when we do division.

```
                1 0 0 1
          ---------------
1 0 1 1 ) 1 0 1 0 0 1 1
          1 0 1 1
          -------
            0 0 1 0
            0 0 0 0
            -------
              0 1 0 1
              0 0 0 0
              -------
                1 0 1 1
                1 0 1 1
                -------
                  0 0 0
```

The remainder is zero, so we pass CRC check.

Design Description

In this lab, you are asked to implement a design called CDC. **CDC will receive [1:0]mode ,[71:0] received_data1,[6:0]received_data2, and[63:0]message, and the functions are as below**

We are going to implement **CRC-8-CCITT standard**, which the polynomial is $x^8+x^7+x^3+x^2+1$

In mode 0, you should generate 8 bits CRC for the 64 bits message but the received_data1 and received_data2 are nothing to do in mode 0. Finally, you should output a 72 bit codeword.

In mode 1, you should check CRC for the 72 bits received_data1 but the message and received_data2 are nothing to do in mode 1.
If CRC check pass, you should output all zeros.
Otherwise, you should output all ones.

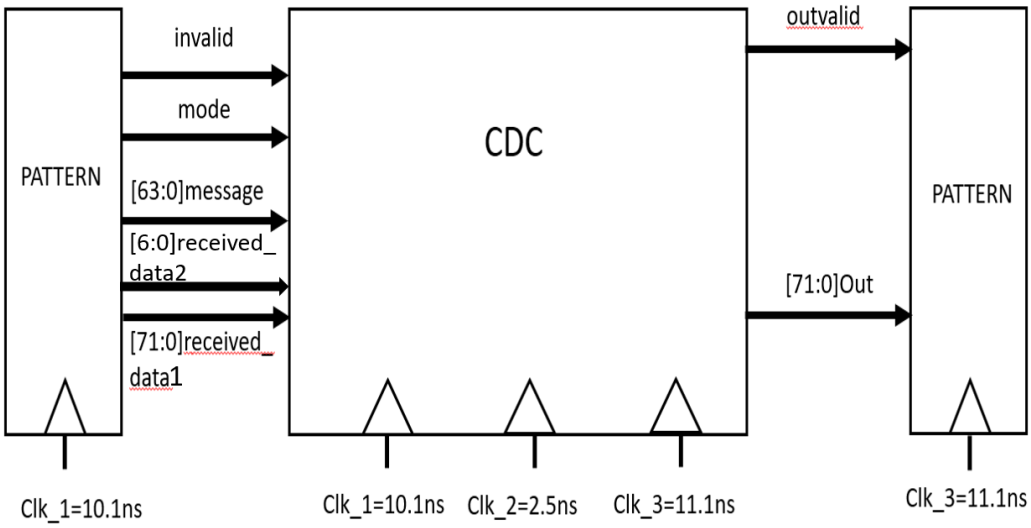
In mode 2, you should correct the 7 bits received_data2 if there is an error, and output the legal codeword. If received_data2 is a legal codeword, you only need to output received_data2.
There is only one error at most in the receiced_data2.
There is nothing to do with received_data1 and message in this mode.

On the other hand, there will be three clock domain in this design, the input pattern will be triggered by clock 1, which is 10.1ns and the output pattern will be triggered by clock 3, which is 11.1ns.

You can follow the instruction below to finish your design, in general, you should use the fastest clock to implement your CRC and ECC.

1. **INPUT pattern:**
   A. Receive: mode , received_data1,received_data2 and message
   B. Clock period is **10.1ns**
2. **CDC:**
   A. CRC module
   B. Clock period is **2.5ns**
3. **OUTPUT pattern:**
   A. Output:out
   B. Clock period is **11.1ns**

The block diagram of this lab is shown below:



Inputs

I/O	Signal name	Bits	Description
Input	**clk_1**	1	Positive edge trigger clock by clock 1 with clock period 10.1ns
Input	**clk_2**	1	Positive edge trigger clock by clock 2 with clock period 2.5ns
Input	**clk_3**	1	Positive edge trigger clock by clock 3 with clock period 11.1ns
Input	**rst_n**	1	Asynchronous reset active low reset
Input	**invalid**	1	**mode, message, received_data1 and received_data2** are valid when **invalid** is high This signal is trigger by **clk_1 for one cycle**
Input	**Mode**	2	This signal is trigger by **clk_1 for one cycle**
Input	**message**	64	This signal is trigger by

			clk_1 for one cycle
Input	**received_data1**	72	This signal is trigger by **clk_1 for one cycle**
Input	**received_data2**	7	This signal is trigger by **clk_1 for one cycle**

Outputs

I/O	Signal name	Bits	Description
output	**outvalid**	1	Should be set to low after reset and not be raised when **invalid** is high. Should set to high when your **out** is ready
output	**out**	72	**out** is synchronous to the positive edge of **clk_3,** and TA's pattern will sample your result at the negative edge of **clk_3** **should not raised more than 1 cycle when you output your answer.** **Mode 0: out = [71:0] codeword** **Mode 1: out =** **72'b00….0 ;pass CRC** **72'b11.…1 ;fail CRC** **Mode2: out = legal codeword (unsigned extension to 72 bits)**

Specifications

1. Top module name : CDC (File name: CDC.v)
2. Input pins : **clk_1, clk_2,clk_3, invalid, [1:0]mode, [63:0]messase, [71:0]received_data1, [6:0]received_data2**
   Output pins : **outvalid, out**
3. Use **asynchronous** reset active low architecture.
4. Input data are synchronous to clk_1, output data are synchronous to clk_3.
5. All your output reg should be set zero after reset.
6. **Changing clock period is prohibited**.
7. After synthesis, check the "CDC.area" and "CDC.timing" in the folder "Report".

The area report is valid only when the slack in the end of "CDC.timing" is **non-negative**.

8. The synthesis result **cannot** contain any **latch**(in syn.log).
9. The output loading is set to 0.05.
10. Input delay and output delay are 0.5*Clock Period.
11. You can't have timing violation in gate-level simulation
12. **You need to write your own .sdc file(hint:"set false path" in Lab06).**
13. Your latency should be smaller than 400 cycles in clk_3
14. Output should not raised more than 1 cycle

Grading Policy

Function correct 70%
Simulation time 20%
Area 10%

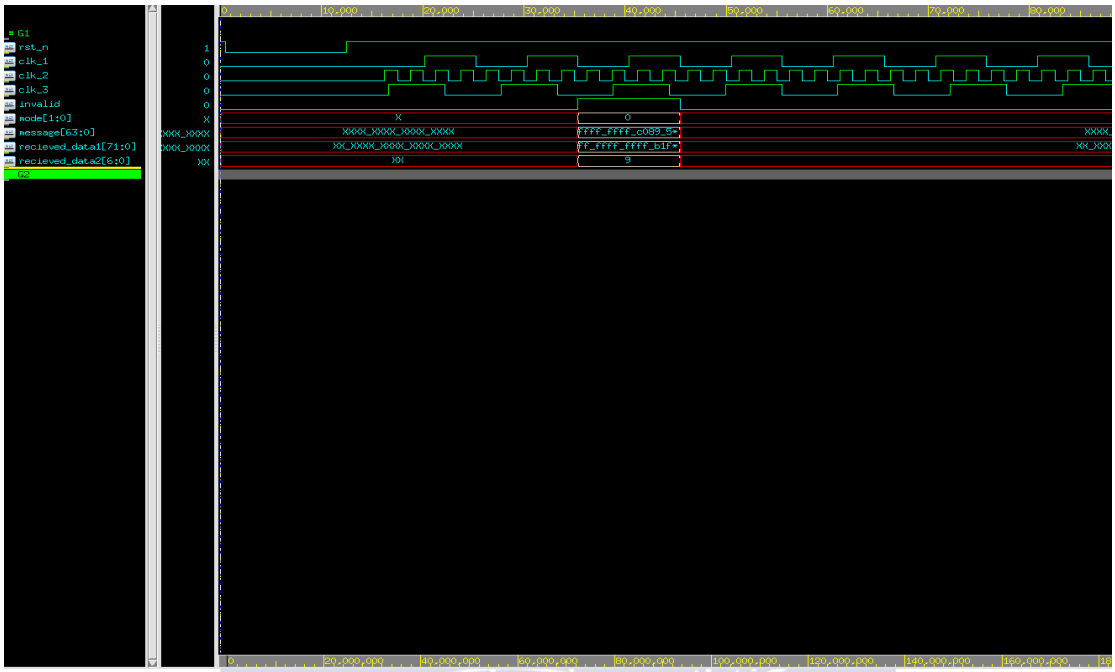
Note

Template folders and reference commands:

1. 01_RTL/ (RTL simulation) **./01_run**
2. 02_SYN/ (Synthesis) **./01_run_dc**
   (Check the design which contains **latch and error** or not in **syn.log**)
   (Clean up 02_SYN folder before you synthesize, or there are errors after you run prime time.)
   ./02_run_pt
   (**set_annotated_check** for the first FF of synchronizer)
   (Check the design's timing in /Report/ CDC.timing)
3. 03_GATE_SIM/ (GL simulation) **./01_run**
4. You can key in **./09_clean_up** to clear all log files and dump files in each folder.
5. **You need to upload your design and CDC.sdc and name them as CDC_iclabxx.v and CDC_iclabxx.sdc**

Waveform Example

Input



Output