NCTU-EE IC LAB – Fall 2019

Lab07 Exercise

Design: Cross Domain Clock (CDC)

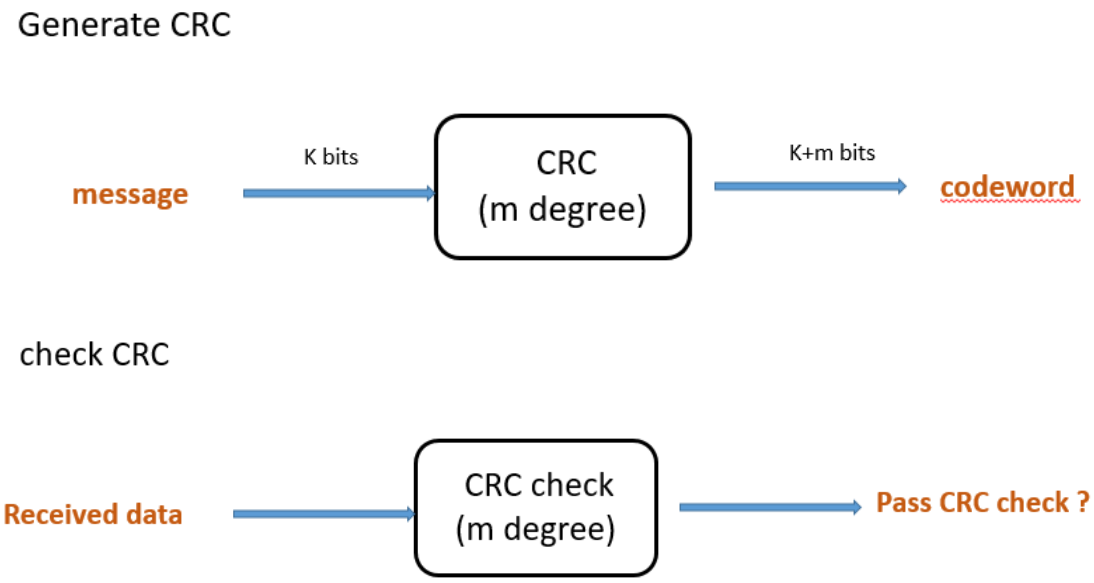
Data Preparation

1. Extract test data from TA's directory:
   % tar xvf ~iclabta01/Lab07.tar
2. The extracted LAB directory contains:
   a. **00_TESTBED**
   b. **01_RTL**
   c. **02_SYN**
   d. **03_GATE**

Basic Concept

This lab will give you a basic concept about cross domain clock and Cyclic redundancy check(CRC).

A cyclic redundancy check (CRC) is an error-detecting 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.



Example 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

Example for detecting the existence of error:

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

```
                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'b mode signal, 1'b CRC signal and [57:0] message signal, and the functions are as below**

We are going to implement **CRC-5-CCITT standard and CRC-5-USB standard.** The CRC-5-CCITT standard has the polynomial $x^5 + x^3 + x^1 + 1$ and CRC-5-USB standard has the polynomial $x^5 + x^2 + 1$.

If mode = 1'b0, you should generate 53 bits CRC for the lower 53 bits message signal ([52:0]) and the upper 5 bit signal will be all zeros ([57:53]). Finally, you should output a 58 bit codeword.

If mode = 1'b1, you should check CRC for the 58 bits message signal. If CRC check pass, you should output all zeros. Otherwise, you should output all ones .

If **CRC = 1'b0**, you should compute for **CRC-5-CCITT ($x^5 + x^3 + x^1 + 1$).**
If **CRC = 1'b1**, you should compute for **CRC-5-USB ($x^5 + x^2 + 1$).**

On the other hand, there will be three clock domain in this design. Two of them are for the IO clock speed. The input pattern will be triggered by clock 1, which is 14.1ns and the output pattern will be triggered by clock 3, which is 11.1ns. The last clock signal is designed for computation which is clock 2 that is 2.5ns.

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

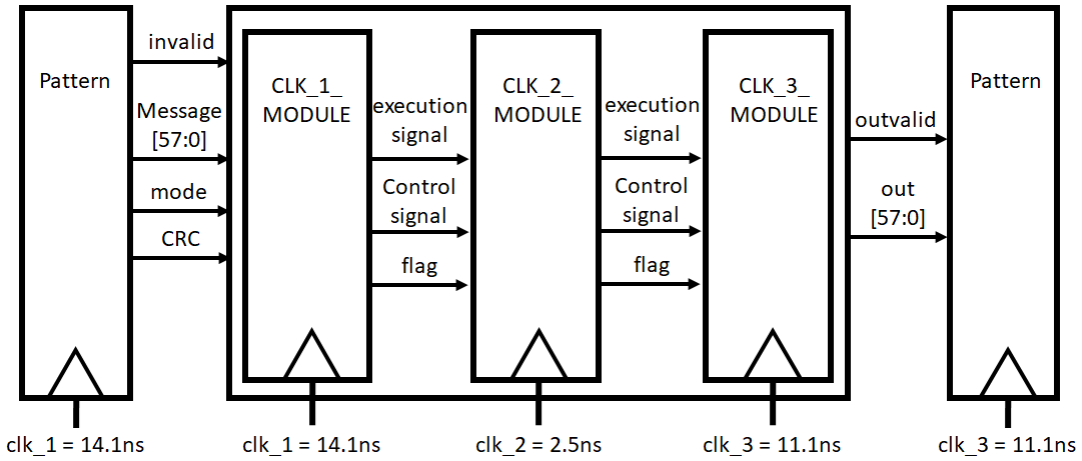
The design is already divided the into three modules, you should write your design and synchronizer in the three modules and send the signal to other modules through the ports provided by TA.

1. **CLK_1_MODULE: INPUT pattern:**
   A. Receive: mode, CRC type and message
   B. Clock period is **14.1ns** (additional clk_2 = 2.5ns for synchronizer)
2. **CLK_2_MODULE CRC:**
   A. CRC module
   B. Clock period is **2.5ns** (additional clk_3 = 11.1ns for synchronizer)
3. **CLK_3_MODULE 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 14.1ns (sent into **CLK_1_MODULE**)
Input	**clk_2**	1	Positive edge trigger clock by clock 2 with clock period 2.5ns (sent into **CLK_1_MODULE**) (sent into **CLK_2_MODULE**)
Input	**clk_3**	1	Positive edge trigger clock by clock 3 with clock period 11.1ns (sent into **CLK_2_MODULE**) (sent into **CLK_3_MODULE**)
Input	**rst_n**	1	Asynchronous reset active low reset (sent into **all three modules**)
Input	**invalid**	1	**mode, message** are valid when **invalid** is high This signal is trigger by **clk_1 for one cycle** (sent into **CLK_1_MODULE**)
Input	**CRC**	1	If **CRC=0**, compute for polynomial $x^5 + x^3 + x^1 + 1$. If **CRC=1**, compute for polynomial $x^5 + x^2 + 1$. (sent into **CLK_1_MODULE**)

Input	**mode**	1	This signal is trigger by **clk_1 for one cycle** (sent into **CLK_1_MODULE**)
Input	**message**	58	**In mode 0, [52:0] signal** represent un-coded signal. [57:53] signal will remain zeros. **In mode 1, [57:0] signal** represent the coded signal. (sent into **CLK_1_MODULE**)

Outputs

I/O	Signal name	Description
output	**outvalid**	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 form **CLK_3_MODULE**)
output	**out**	**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 = [57:0] codeword** **Mode 1: out = 58'b00….0 ;pass CRC** **58'b11…1 ;fail CRC** (output form **CLK_3_MODULE**)

Specifications

1. Top module name : CDC (File name: CDC.v)
2. Submodule name : CLK_1_MODULE, CLK_2_MODULE, CLK_3_MODULE
   (File name: DESIGN_MODULE.v)
3. Input pins : **clk_1, clk_2, clk_3, invalid, mode, [58:0]messase, CRC**
   Output pins : **outvalid, out**
4. Use **asynchronous** reset active low architecture.
5. Input data are synchronous to clk_1, output data are synchronous to clk_3.
6. The reset signal (rst_n) would be given only once at the beginning of simulation. All output signals should be reset after the reset signal is asserted.
7. The out should be correct when outvalid is high.
8. The out should be reset after your outvalid is pulled down.
9. **Changing clock period is prohibited.**
10. **Changing top module is prohibited.**

11. 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**.
12. The synthesis result **cannot** contain any **latch**(in syn.log).
13. The output loading is set to 0.05.
14. Input delay and output delay are 0.5*Clock Period.
15. You can’t have timing violation in gate-level simulation
16. **You need to write your own .sdc file.**
17. Your latency should be smaller than 400 cycles in clk_3
18. Output should not be raised more than 1 cycle
19. The next input pattern will come in 1~3 clk_1 cycles after **outvalid** falls.

Grading Policy

Function correct 70%

Latency * Area 30%

(Latency is calculated in clk_3)

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**)
   ./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 DESIGN_MODULE_iclabxx.v and CDC_iclabxx.sdc to new E3 before Nov. 18th 12:00 pm for 1st demo or before Nov. 20th 12:00 pm for 2nd demo.**

Waveform Example