Optional Lab9 - Creeping Line

Tsotne Putkaradze

May 2015

Contents

List of Figures

Introduction

This document is created on a purpose of proving the understanding of blablabla

1 Problem Statement

The task given is following: we are required to write a single code, that will be uploaded to N number of FPGAs, which are connected with each other via serial connection, and which should construct marquee of text using existing Seven Segment Displays, in other words: moving text on Seven Segment Displays. main things that should be considered [1]

- each FPGA should be able to be configured into Ma, Slave or By pass mode: on run
- system should be scalable from 1 to N number of FPGAs
- we should provide some algorithm of error checking (CRC, parity,..)
- there should be only one master in the whole chain, if there i s more than one, stop process
- initialize some form of hand-shaking between FPGAs

2 Solution

Divide and Conquer ! In order to make life easy and things possible, we should break the whole system into smaller parts, design them : assemble. We decided to use:

- Serial Connection
- one way Handshaking with header
- CRC error checking

For this methodology we created top module Creeping Line Serial which have following components:

1. Reading Memory
    - Data Memory
    - Output Buffer
2. Input Buffer
3. Output Buffer
4. Seven Segment Controller
5. clock stuff
6. constraints

2.1 General Description

before we start describing each sub-part of system, lets talk about the behaviour of system, in general. data flow of system is following:

1. we are waiting for header
2. after getting right header we take input that is consisted of CRC code and actual data that should be printed
3. we check CRC, if there is an error we changed damaged data with bit-vector, that prints "dot" on seven segment display
4. - we print data
   - we shift data on seven segments
   - we send old data on output

the general overview of system looks like this:

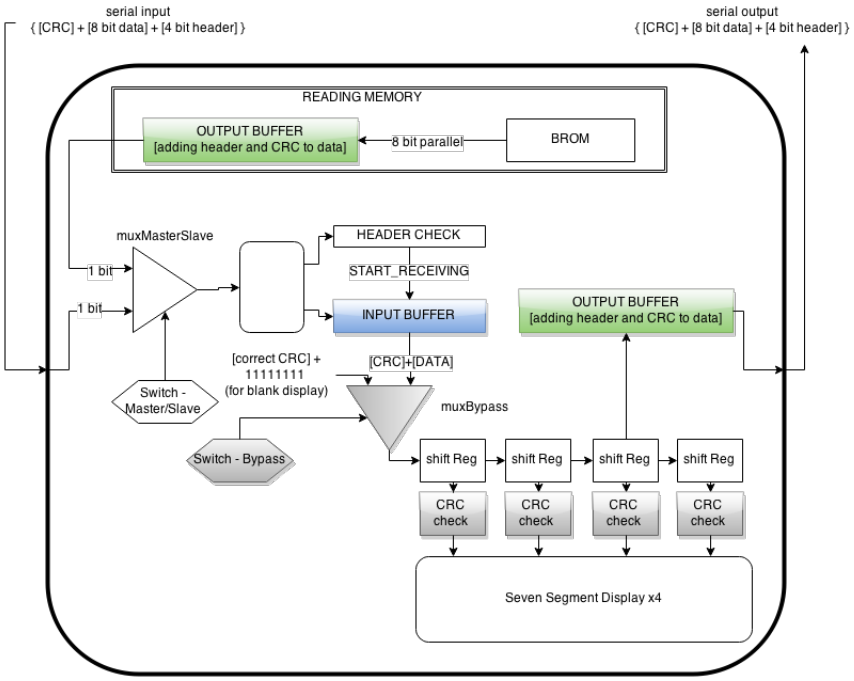

Figure 1: Data Flow Diagram of Creeping Line

2.2 Input Buffer

in input buffer element we have following inputs and output:

```
entity InBuffer is
    generic(inputSZ : integer;
            countSZ : integer
    );
    port(
        Received_Data    : out STD_LOGIC_VECTOR(inputSZ - 1 downto 0);
        DataIsReceived_F : out std_logic;
        StartReceiving_S : in  std_logic;
        PMOD_in          : in  std_logic;
        clk              : in  std_logic;
        rst              : in  std_logic
    );
end entity InBuffer;
```

Figure 2: Input Buffer ports

it takes input from PMOD and start signal, and and gives out received data with a flag notifying that data-reading-process has completed. please note that once start signal is given to the input buffer, nothing, except reset signal will affect/stop the process.

2.3 Output Buffer

in this buffer we have following ports

```
entity OutBuffer is
    generic(outputSZ : integer;
            countSZ  : integer
    );
    port(
        DataIsSent_F   : out std_logic;
        PMOD_out       : out std_logic;
        Data_To_Send   : in  STD_LOGIC_VECTOR(outputSZ - 1 downto 0);
        StartSending_S : in  std_logic;
        clk            : in  std_logic;
        rst            : in  std_logic
    );
end entity OutBuffer;
```

Figure 3: Output Buffer ports

in this buffer, we take start signal and data that has a bitwidth of outputSZ, once we start outputting, the process can't be stopped. but if we change "Data To Send" signal during the process, changed data will be send, this is because, otherwise we should have introduced another register with a size of outputSZ, that would get the value of "Data To Send" when Start signal would be 1, but i did not do that because it takes 1 clock cycle for this operation to start. but i wanted to make it in a way that, sending will start, immediately i provide the signal. the reason for that is Input buffer. lets say, we are reading header that was the one we expected, so then, next bit is data already, and that means,

we should NOT introduce 1 clock cycle delay between header check and data reading, now, if we introduce this delay in output buffer, we can assume that, if:

- receiving - takes 10 clock cycle
- sending - takes 10+1 clock cycle

so this creates imbalance and that might cause buffer overflow after some period of time. for example, if we have 100 clock cycles between sending of two data, in 100 clock cycle we will have buffer overflow.

2.4 Reading Memory

port description can be seen on the diagram below:

```
entity ReadingMemory is
    generic(
        headerPattern : STD_LOGIC_VECTOR (3 downto 0) := "1111";
        dataSZ    : integer := 8;
        memSZ     : integer := 2;
        countSZ   : integer := 3;
        headerSZ  : integer := 4
    );
    Port(
        mem_data_bit_out  : OUT STD_LOGIC;
        MemIsRead_F       : OUT STD_LOGIC;
        data_Addrs_in     : in  STD_LOGIC_VECTOR(memSZ - 1 downto 0);
        StartReadingMem_S : in  STD_LOGIC;
        rst               : in  STD_LOGIC;
        clk               : in  STD_LOGIC
    );
end ReadingMemory;
```

Figure 4: Reading Memory ports description

in this module we have instantiated two components:

- Data Memory, that contains actual 8 bit data:

```
entity DataMemory is
    generic(
        headerPattern : STD_LOGIC_VECTOR (3 downto 0);
        dataSZ    : integer;
        memSZ     : integer;
        headerSZ  : integer
    );
    Port(
        mem_data_out  : OUT STD_LOGIC_VECTOR(dataSZ + headerSZ -1 downto 0);
        data_Addrs_in : in  STD_LOGIC_VECTOR(memSZ-1 downto 0);
        rst           : in  STD_LOGIC;
        clk           : in  STD_LOGIC
    );
end DataMemory;
```

Figure 5: Data Memory

- and Output Buffer, that takes data from data memory and sends it bit-by-bit 3

in this part, i tried to be smart and did following:

- created memory - i will make BROM later
- and started to read data from memory in a linear way

i did this because in this way, my data path will be same in both cases:

- if we are master and we are reading from memory
- or if our board is in Slave mode, and we are just receiving data from PMOD

you can view the diagram here: 1

2.5 Other Elements

in file

- Seven Segment Controller - we take four 8-bit inputs and print them on four seven segment displays
- clock stuff - we take reference clock and generate every kind of clock we might ever need in our design:
  - clock for Seven Segment Display
  - clock for data transmission
  - clock for button debouncer - in case we will use any buttons and etc.
- constraints - in this file we simply map our input-output ports on the pins of Nexys 3 board that has Spartan 6 FPGA.
- and finally, our top module looks like this:

```
entity CreepingLineMasterSlave is
    generic(
        memSZ         : integer                     := 3;
        inputSZ       : integer                     := 8;
        dataSZ        : integer                     := 8;
        countSZ       : integer                     := 3;
        outputSZ      : integer                     := 12;
        headerSZ      : integer                     := 4;
        headerPattern : STD_LOGIC_VECTOR(3 downto 0) := "1111"
    );
    Port(
        an                  : out STD_LOGIC_VECTOR(3 downto 0);
        SevSeg              : out STD_LOGIC_VECTOR(7 downto 0);
        LEDS_Header         : out STD_LOGIC_VECTOR(headerSZ - 1 downto 0);
        led8th_btn_deb_out  : out STD_LOGIC;
        led7th_data_received : out STD_LOGIC;
        led6th_clks_out     : out STD_LOGIC;
        PMOD_out            : out STD_LOGIC;
        clk_wire_out        : out STD_LOGIC;
        clk_wire_in         : in  STD_LOGIC;
        masterSwitch        : in  STD_LOGIC;
        PMOD_in             : in  STD_LOGIC;
        realCLK             : in  STD_LOGIC;
        btn_in              : in  STD_LOGIC;
        rst                 : in  STD_LOGIC
    );
end CreepingLineMasterSlave;
```

Figure 6: Creeping Line ports

3 Conclusion / Results

Benefits, Results and Conclusions from this project are following:

1. understanding the principles of Serial Communication
2. principles, drawbacks and benefits of handshake communication
3. importance of synchronous operations, data-sending and receiving cycles/times

References

[1] Dimitry Mikhailov Muhammad Adeel Tajammul. *Lab 9 Creeping Line.* TTU, 2015.