Embedded Systems Lab 2 Report

Hong-Bin Yang, Tz-Ching Yu

November 27, 2022

1 Introduction

The goal of Lab 2.2 is to design a custom programmable interface facilitating the control of an external component. We choose to design a custom interface for WS2812. In the following sections, we will introduce the mechanism of WS2812, the proposed design for the interface, and the implementation details of the design.

1.1 WS2812

WS2812 is a single package with a control circuit and RGB LEDs. It is capable of latching the first few bits of signals received and propagating the rest to downstream components. Therefore, it's usually used to build a daisy chain of LEDs by cascading the input and output of multiple WS2812.

The component follows non-return-to-zero (NRZ) transmission protocol where signals of 0 and 1 are encoded as specific conditions. For the component, bit 0 is encoded as 0.35 us of high and 0.8 us of low, and bit 1 is encoded as 0.7 us of high and 0.6 us of low. The waveform is as shown in Fig.1.

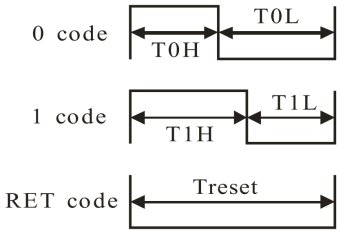

Figure 1: The encoding scheme of WS2812 for 0, 1 and reset

As mentioned previously, the component consists of an array of RGB LEDs. The color of each LED is controlled by 8 bits, and the transmission order is G, R then B and MSB first. The typical waveform is shown in Fig. 2. After each cycle of data transmission, a reset code follows which is encoded as more than 50 us of low.

2 Interface Design

After understanding the working of WS2812, the design of the interface becomes quite clear. The essential idea of the interface is to output varying intervals of high and low based on current bit to transmit, which can be done quite easily through a counter. There are many possible designs could achieve the functionality. For example, a design using a single data register storing the data received

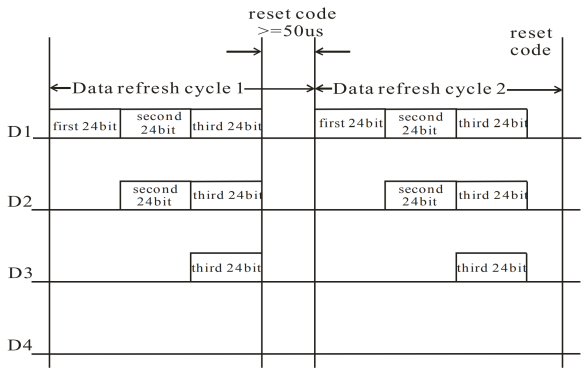

Figure 2: Example waveform for 4-pixel systems

from the microcontroller and outputs signals corresponding to the bits of the data. However, this kind of design makes the interface harder to use from a software perspective. We would like to abstract away the details of WS2812 by allowing the programmers to only have to program the colors they want to display and the interface will handle the rest. Therefore, our final design of the custom Avalon slave interface consists of 9 registers and the block diagram is shown below.

The **RegData** is an array of 8 registers to store the color code for each pixel. The **RegStart** is used to initiate the transmission of data after data registers have been set properly. The register map is summarized as below.

Address	Field	Type	Description
0	RegStart	R/W	control the start of signal transmission
1	RegData[0]	R/W	data register
2	RegData[1]	R/W	data register
3	RegData[2]	R/W	data register
4	RegData[3]	R/W	data register
5	RegData[4]	R/W	data register
6	RegData[5]	R/W	data register
7	RegData[6]	R/W	data register
8	RegData[7]	R/W	data register

2.1 Control Logic

The control of the interface can be summarized as a finite-state machine (FSM) consisting of 8 states. The state diagram is as shown in Fig. 3. The overall logic is as follows. After **RegStart** is set, the MSB of first register of **RegData** is read, and state transitions to S_2 or S_3 depending on the bit. The output of the interface is set to 1 in both S_2 and S_3. After counter reaches the specific values, state transitions to S_4 or S_5 and the output becomes 0 in these states. Once again, after counter reaches the specific values, state transitions to S_6. In S_6, it determines if every bit of the current register has been transmitted and transitions to S_1 starting to transmit another register of **RegData**, or read another bit of the register and moves to S_1. The cycle continues until every bit of every register has

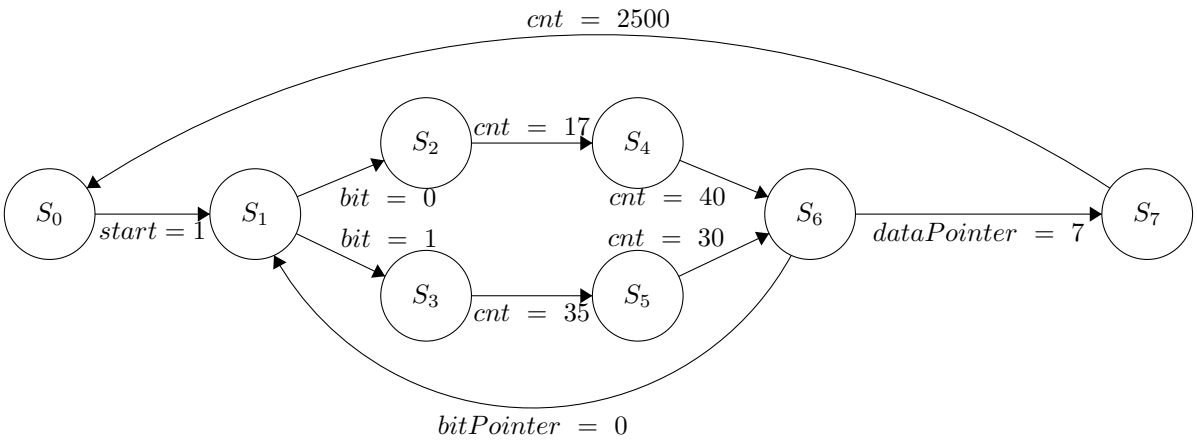

Figure 3: State diagram of the interface where **bit** is the current bit of the current register and **bitPointer** and **dataPointer** are used to keep track of the bit and register numbers of the current bit.

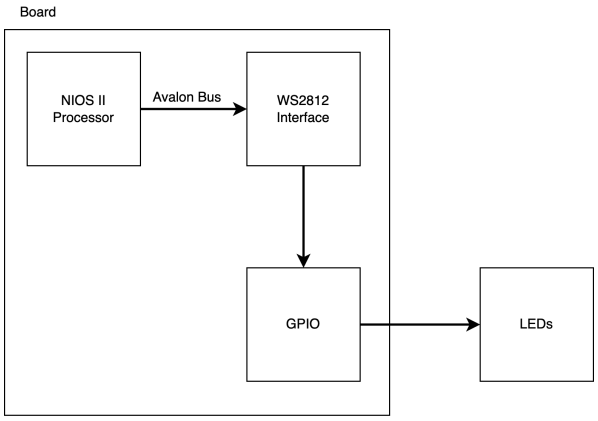

Figure 4: Overall system block diagram

been transmitted. It then moves to S_7 where the output is set to 0 which corresponds to reset code, and this concludes a single data refresh cycle.

3 Implementation

The overall system on the FPGA board consists of a NIOS II core, 128 KB on-chip memory, JTAG UART and our custom interface for WS2812. The system connection is as shown in Fig.4. The output of our interface is directly connected to the pin 1 of the **GPIO_0** on board. We implemented the aforementioned custom interface in VHDL. Please refer to the original VHDL source files in the zip file for detailed implementation.

3.1 Simulation

To verify the design before actually program the FPGA, we wrote a testbench to check the interface works as we expected. The simulation process of the test bench can be found below.

```
simulation: process
    begin
        nReset <= '0';
        wait for 20 ns;
        nReset <= '1';

        -- storing colors to display in registers
        address <= "0001";
        write <= '1';
        writedata <= std_logic_vector(to_unsigned(1, writedata'length));
        wait for 20 ns;

        address <= "0010";
        writedata <= std_logic_vector(to_unsigned(2, writedata'length));
        wait for 20 ns;

        address <= "0011";
        writedata <= std_logic_vector(to_unsigned(4, writedata'length));
        wait for 20 ns;

        address <= "0100";
        writedata <= std_logic_vector(to_unsigned(8, writedata'length));
        wait for 20 ns;

        address <= "0101";
        writedata <= std_logic_vector(to_unsigned(16, writedata'length));
        wait for 20 ns;

        address <= "0110";
        writedata <= std_logic_vector(to_unsigned(32, writedata'length));
        wait for 20 ns;

        address <= "0111";
        writedata <= std_logic_vector(to_unsigned(64, writedata'length));
        wait for 20 ns;

        address <= "1000";
        writedata <= std_logic_vector(to_unsigned(128, writedata'length));
        wait for 20 ns;

        -- start outputing signals
        address <= "0000";
        writedata <= std_logic_vector(to_unsigned(1, writedata'length));
        wait for 20 ns;

        write <= '0';
        wait;
end process simulation;
```

The simulation started by resetting the interface, then programmed each register of **RegData** to some initial values and set **RegStart** to 1. The resulting waveform is shown in Fig. 5 where **dout** was the output of the interface. As the timescale of ModelSim was in picosecond, we zoomed in the waveform where the interface was transmitting data of a single register. Since the first data was 1, the output consisted of 23 patterns of bit 0 and 1 pattern of bit 1.

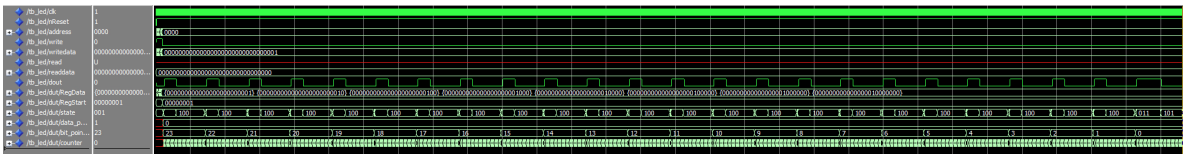

Figure 5: Simulation waveform of the interface transmitting the first 24 bits

3.2 On-board Testing

After successful simulation, we programmed the FPGA with the whole system. To test the entire system, we wrote a simple C program to verify the design. The source code is listed below.

```
#include <inttypes.h>
#include <stdio.h>
#include "system.h"
#include "io.h"

int main()
{
  int i;
  unsigned offset = 0;
  // define number of colors, number of WS2812 and the individual color code
  int num_colors = 14, num_regs = 8;
  int colors[14] = {0xff0000, 0xff0087, 0xff00e7, 0xa800ff,
          0x4500ff, 0x0036ff, 0x008eff, 0x00f9ff,
          0x00ff93, 0x00ff01, 0x9fff00, 0xffff00,
          0xffa700, 0xff6200};

  IOWR_32DIRECT(WS2812_INTERFACE_0_BASE, 0, 1)
  while(1){
    // program the data registers with the color code
    for (i=0;i<num_regs;i++){
        IOWR_32DIRECT(WS2812_INTERFACE_0_BASE, 4 * (i + 1), colors[offset++]);
        offset = (offset + 1) % num_colors;
    }
    // delay for a bit
    for(i=0;i<200000;i++);
  }

  return 0;
}
```

We flashed the program into on-chip memory and the system worked as expected. The colors successfully cycled through each pixel. The video of demonstration can be found in the zip file.

4 Conclusion

In this lab, we designed, implemented and tested Avalon custom programmable interface for WS2812. The resulting system allows programmers to easily program a chain of 8 WS2812 and the interface design can be easily modified to support chains of any length by utilizing generic in the VHDL source file.