CmpE Internship Summer 2018
Design Doc

Dial Controlled Flashlight

Authors & Editors:

Zackery K. Plovanic

Revision History:

Revision	Date	Description
A.3	**8/14/2018**	Documented changes to my system architecture, updated memory map, added clarity to the pseudocode.
A.2	**7/13/2018**	Clarified Design, added pseudocode for high level functionality.
	6/30/2018	Approved by Vincent.
	6/26/2018	Vincent set as reviewer. Approval pending.
A.1	**6/16/2018**	Initial Commit

Table of Contents

Objective

In this project I aim to create a dial-controlled flashlight. The system is composed of a series of LEDs that are controlled/activated depending on the current position of a rotary encoder. As this rotary encoder is turned, the LEDs fluctuate as the encoded signal changes.

Background

Input Device: Rotary Encoder - https://reference.digilentinc.com/reference/pmod/pmodenc/reference-manual

Output Device: Eight High Brightness LEDs - https://reference.digilentinc.com/reference/pmod/pmod8ld/reference-manual

Overview

For this project, the architecture for the system as a whole needs to be designed. This includes integrating a MIPS processor with its dedicated ROM, as well as the I/O devices that I have chosen, a rotary encoder and high-brightness LEDs. These devices need to be designed to prevent any possibility of bus contention. In addition to this all, a clock generator module is needed to slow the native clock of the Nexys 4 to allow the MIPS processor sufficient time to execute its instructions.

To allow the system architecture to interact with my rotary encoder, a quadrature decoder is needed to track the position of the PMOD device. In addition, eight PWM Drivers will be used to adjust the brightness of the either LEDs. Aside from those, no additional architecture is needed to interface with the PMOD devices.

In addition to the hardware design, I will also need to write the assembly code to constantly read the quadrature decoder and write the corresponding value to the LEDs. This needs to be done using the MIPS instruction set. Once that program is written, I will take that hex code and place it into the ROM to be executed.

Detailed Design

My system uses memory mapped IO with partial address decoding to interface with the PMOD devices. The system's addressing is as follows:

Device	31-16	15	14	13-11	10-7	9-0	Equation
Frequency Register	0	X	0	X	0x9	X	NOR(31-16)*14*10*~9*~8*7
Quadrature Decoder	0	X	1	X	0x8	X	NOR(31-16)*14*10*~9*~8*~7
PWM Driver 7	0	X	1	X	0x7	X	NOR(31-16)*14*~10*9*8*7
PWM Driver 6	0	X	1	X	0x6	X	NOR(31-16)*14~10*9*8*~7
PWM Driver 5	0	X	1	X	0x5	X	NOR(31-16)*14~10*9*~8*7
PWM Driver 4	0	X	1	X	0x4	X	NOR(31-16)*14*~10*9*~8*~7
PWM Driver 3	0	X	1	X	0x3	X	NOR(31-16)*14*~10*~9*8*7
PWM Driver 2	0	X	1	X	0x2	X	NOR(31-16)*14*~10*~9*8*~7
PWM Driver 1	0	X	1	X	0x1	X	NOR(31-16)*14*~10*~9*~8*7
PWM Driver	0	X	1	X	0x0	X	NOR(31-16)*14*~10*~9*~8*~7

0							

Table 1:

The system is designed in such a way that ensures mutual exclusivity. Address lines 31-16 need to be low in order to interface with the system, as well as line 14 needing to be a logic high. If an IO device is in fact being accessed, lines 10-7 determine which specific IO device is in use. The entirety of the system can be seen here: https://drive.google.com/file/d/1uP8uKzEfXDxVvkTnBYWsJRX8j2orhl_X/view?usp=sharing.

The pseudocode of the program to be stored in side the ROM would be have the following logic:

```
Write the desired frequency value to the Frequency Register.
While(1)
{
Read current value in Quadrature Decoder;
Send bits 2:0 extended to 8 bits as the duty cycle to a PWM
     Driver determined by bits 5:3.
}
```

Caveats

- Limited duty cycle, since its dependant on bits 2:0 from the Decoder

Testing Plan

Unit testing scheme

- Simulate the processor accessing memory locations to ensure that each device is mutually exclusive.
- Step through the program code via simulation to ensure that data values are transferred as intended.
- Assure that the PWM Drivers are sending functioning correctly, checking the frequency and duty cycle of the driver.

Integration Testing

- Once the simulation has proven to function correctly, the bitstream shall be synthesized and generated to load onto the Nexys 4. Once generated it can be loaded onto the FPGA to be physically assessed.

Demonstration Project

- A demonstration/discussion will be had with SJSU faculty during the Internship’s expo on 8/16/2018