Lego LED Christmas Tree

Embedded Final Project

Matthew Wolf - wolfm6@students.rowan.edu

Nolan Foy - foyn2@students.rowan.edu

Joseph DiBenedetto - dibenedej5@students.rowan.edu

Intro to Embedded Systems
Rowan University
Dr. Ying (Gina) Tang
Due Date: December 19th, 2018

Embedded Final Project Report

N Foy, M Wolf, J DiBenedetto
Rowan University

December 19, 2018

1 Introduction

Throughout the entirety of this Embedded systems course, we have attained considerable knowledge on how the various topics from class are applied to technology, software, and customer needs. For our final project, we wanted to make sure to encapsulate the importance of applying that knowledge to real-world situations. In order to do that, we went into the project thinking of the consumer, and what they might request that involves the application of embedded system design. After much thinking, we were able to come up with a figurative scenario that a customer may ask. Our entire final project is based on this possible scenario that could arise during the holiday season. The scenario is as follows:

The holiday season has arrived. With the town littered by several creative Christmas decorations, a customer approaches us with the desire to go above and beyond this year. His desire is to build his Christmas tree completely in Lego's, but explains to us that he is having some trouble figuring out how to implement the lights onto his makeshift Christmas tree. He wants us to show him the best way to finish off his Lego creation.

As a response to our customer's request, our plan is to build a small prototype of his imagined Lego creation. Using a breadboard, LED circuitry, and a GPIO expander, we will show our customer how our prototype can control/drive multiple LEDs. By doing so, we will show off how we can apply embedded systems to solve our customer's problem, as well as showing how this prototype is the best way to illustrate what our customer is trying to convey.

2 Design Overview

Our design, while more intricate to the core, will consist of three basic design features:

- Serial communication over Inter-integrated circuit protocol (I^2C)
- Full control of up to 16 LEDs
- Control blink rate and brightness of each LED

2.1 Featured Applications

This system has the ability to to control the on/off state, blink rate, and brightness levels of up to 16 different LEDs. The final product will be constructed into the form of a Christmas tree, so using the previously defined applications, settings can be found to make the LED assembly look appealing on display.

2.2 Design Resources

The software running this project can be found at the following link on Github:

Enable One LED

Certain lines of this code for the I^2C protocol were inspired by a code found at the TI resource center:

USCI_B0 I2C Master TX single bytes to MSP430 Slave

2.3 Block Diagram

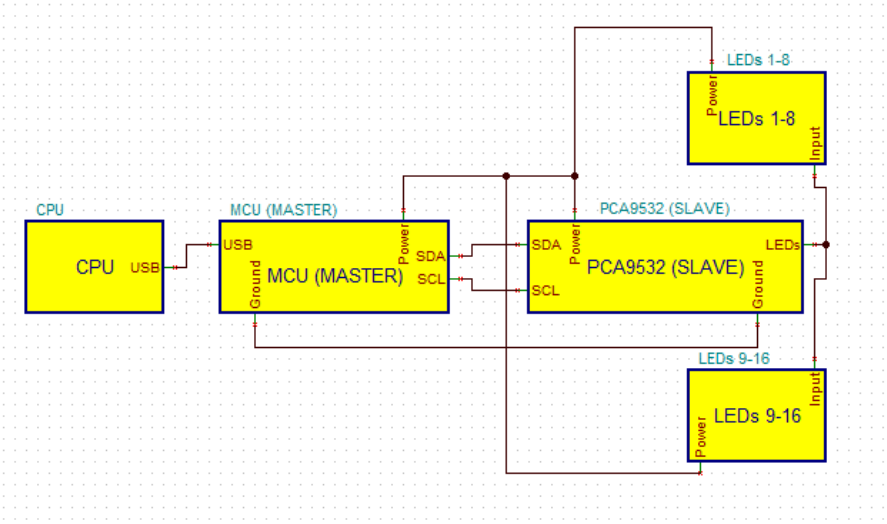

Figure 1: Basic block diagram of the system

Pictured above is the basic block diagram of our entire system.

2.4 Board Image

The circuit built on the breadboard can be seen in the following image:

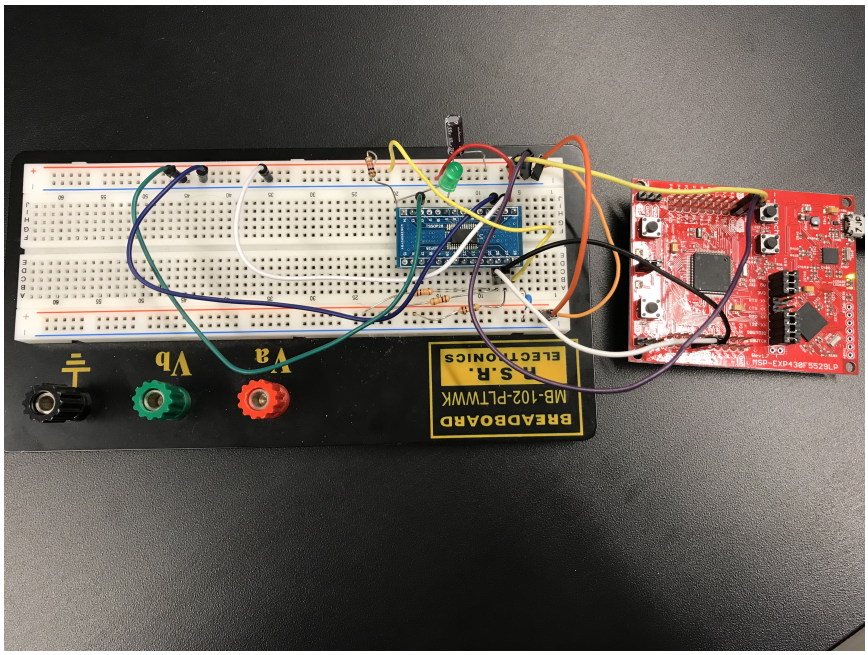

Figure 2: Image of Circuit

The MSP430F5529 acts as the Master, and the PCA9532 acts as the slave. A very important note to include is that the PCA9532 had to be soldered in order for a stable connection with the breadboard to be made.

3 Key System Specifications

PARAMETER	SPECIFICATIONS	DETAILS
Microprocessor	MSP430F5529	Microprocessor needed for corresponding code parameters
GPIO Expander	PCA9532	Chip used for I2C Communication
1 watt 1k, 10k	Resistors	Resistors used in circuit
Code Composer	I2C	Serial Communication requirement for system

4 Design Approach

At the heart of this system is the I^2C communication protocol. This means that their must be at least one master, slave, transmitter, and receiver. In our case, the MSP430F5529 micro-controller is both the master and primary transmitter. We used an I/O expander to control the LEDs. This expander was the PCA9532 and the reason for using I^2C is because that is the only way to communicate with the PCA9532. This device is both the slave and primary receiver. In I^2C the slave transmits an acknowledge bit back to the master after correctly being addressed. This is the only case in this system that the slave will act as the transmitter.

I^2C works with only two transmission lines: SDA for data and SCL for clock. The micro-controller has pins specifically for these two lines and on the MSP430F5529 these were pins P3.0 and P3.1. These were connected directly to the SDA and SCL pins on the PCA9532 and both of these lines and the reset line were pulled up to 5 volts with 10k ohm resistors. The 5 volt source and ground source used were that from the micro-controller, meaning all of this system's power is generated from a laptop battery. The remaining pins used on the PCA9532 were obviously both VDD and VSS, and then addressing bits A0, A1, and A2 were all tied to ground due to only one slave being used. The one LED that was experimented with on this circuit was on the LED3 pin and pulled up to 5 volts with a 120 ohm resistor. This resistor was selected to keep the amperage out of the pin at 25 mA while accounting for a 2 volt drop across an LED.

4.1 Detailed Block Diagram

The following image of the circuit modeled in the TINA-TI software shows all of the specific connections discussed directly above:

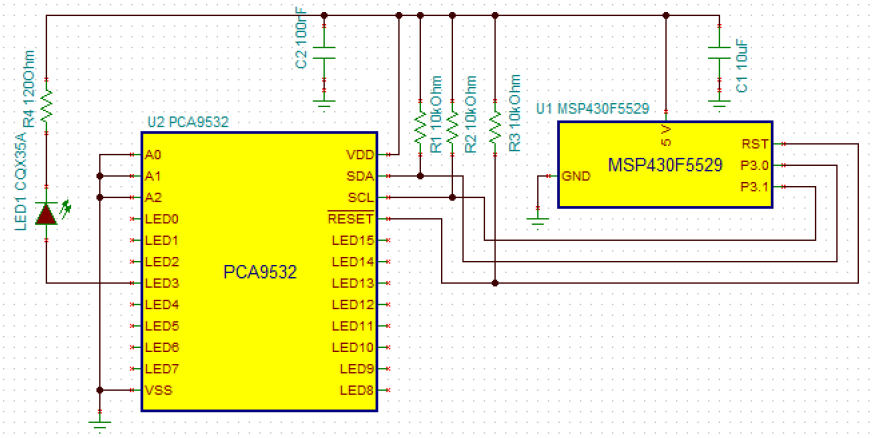

Figure 3: Block Diagram with All Connections

4.2 Highlighted Devices

The two devices that were used in the system is what allowed it to operate. These two devices are listed below:

- MSP430F5529 - I^2C Master, transmits information for which LEDs are on at which PWM
- PCA9532 - I^2C Slave, receives address, command byte, then data byte

4.3 SYSTEM DESIGN THEORY

This section explains in detail how both of the two devices mentioned above contribute to the system and communicate with each other over I^2C.

4.3.1 Device 1: MSP430F5529

The MSP430F5529 is the brain of this system. The program is written into Code Composer Studio and then loaded onto the processor on this board. The board has many capabilities including timers, UART, SPI, and I^2C communication, I/O, etc. Even with the micro-controller being so crucial to this system, the only feature of the board we needed to use was the I^2C communication. The F5529 acts as the master within the system. Communication occurs between master and slave on the SDA line, the clock is controlled from inside the master and sent on the SCL line, and a reset line is also sent from master to slave. The address byte is transmitted first by just loading registers within the micro-controller with certain information, and it will then send out this first byte on its own. The following command and then data bytes must be loaded into the UCB0TXBUF buffer in order to be transmitted.

Within our system the micro-controller also acts as the power source. Both the 5 volt and ground pins are ran from the board to the positive and negative rails on the breadboard that can be seen in Figure 2.

4.3.2 Device 2: PCA9532

The contents of this relatively simple I/O expander can be seen in the following figure:

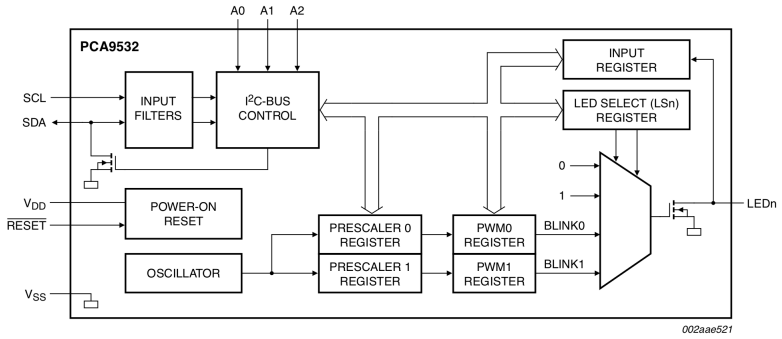

Figure 4: Contents of the PCA9532

The slave within an I^2C system works by receiving an address byte, followed by a command byte, followed by a data byte. If more data bytes wanted to be sent, another command byte for each additional data byte must be sent first unless the auto-increment feature is used. Following the reception of each of these bytes the slave (PCA9532) transmits and acknowledge bit. Depending on the contents of the command and data bytes certain LEDs will be either turned on, left off, or set to one of two programmable PWM rates. The PCA9532 possesses 16 LED drivers. In this system the slave is powered and grounded from the micro-controller.

4.4 Getting Started/How to use the device

Figure 4 shows the output of the pins from the PCA9532. The circuit schematic shown in figure 5 is how we connected the master (MSP430F5529) to the slave (PCA9532).Ports 3.0 which is SDA (Serial Data) and 3.1 (Serial Clock) from the F5529 were then connected to the PCA9532's corresponding SDA(pin 23) and SCL(pin 22) allowed us to communicate from the master to slave. The SDA and SCL from PCA9532 was then tied to power with 10kΩ resistors. Using the oscilloscope and TI Explorer example code we then confirmed that the PCA9532 was acknowledging the F5529. We then connected the 5v source to pin 24 and the ground to pin 12 (Vdd) from F5529 to the PCA9532. Pin 1(A0), pin 2 (A1), and pin 3 (A2) were all grounded to the board. The LED is then placed in pin 7 of PCA and then connected to a 1kΩ pull-up resistor. If we wanted to light up more LEDs, LEDs could be placed in pins 4-11 and 13-20 as long they are tied to a 1kΩ pull-up resistor.

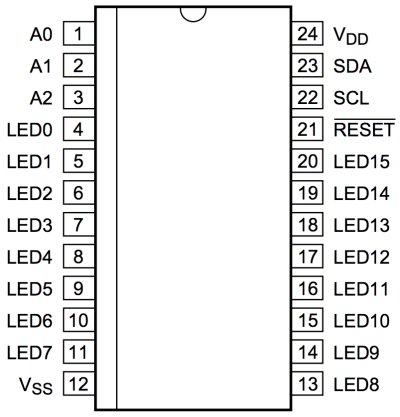

Figure 5: PCA9532 Pin Outputs

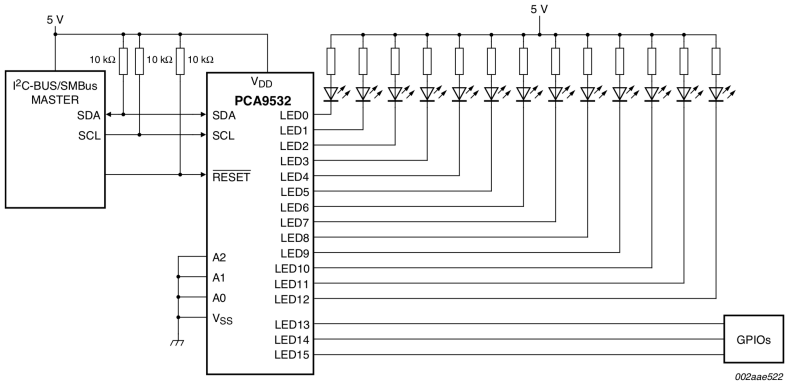

Figure 6: Circuit Schematic Master to Slave

4.5 Getting Started Software/Firmware

Code Composer was used in order to write the C code that entails the I^2C serial communication. Prior to the int main to unsigned char were as signed in order to force the data to be 8 bits. 4 chars were then assigned to the 16 LEDs where each char contained 4 LEDs in order to call to. The int main void contained all the parameters in order to communicate from master to slave where I^2C was assigned to USCI_B0. A while loop then followed after in order to pass the command line and data in order to turn the LED on.

4.6 Communicating with the Device

In order to communicate with the device I^2C was used, since we are using serial communication by using a master(MSP430F5529) and slave(PCA9532). Therefore ports 3.0 (SDA) and 3.1(SCL) from the F5529 which contain the necessary parameters to utilize I^2C. This was then connected to the PCA9532's pins 22 (SCL) and 23 (SCL). We then determined if the devices were communicating with each other by hooking up the oscilloscope to display acknowledgment. Once done this allowed us to send the command package and data package to turn on the LEDs connected to the PCA9532.

5 Results and Conclusion

The original objective of this project was to construct an LED Christmas tree made of legos for a theoretical customer. This product would have the ability to blink the various LEDs in a pattern and at brightness levels of the customers preferences. A micro-controller was required for this project, so the MSP430F5529 was chosen, and in order to accommodate for the many LEDs that had to be used, an I/O expander, the PCA9532, was used. This device is a 16 bit I2C chip with 16 LED drivers, with the capability of storing and using two user programmable PWM rates. The major difficulty and time consuming portion of this project was programming I2C transmissions into Code Composer Studio. A master within an I2C system must send the slave an address, command, and data byte. We eventually learned that the micro-controller takes care of sending the address byte by setting some registers when writing the program. The following bytes must all be loaded and then transmitted using the UCB0TXBUF buffer register.

All of the allotted time for this project went to constructing a functional circuit on a breadboard, and producing software that could at the very least light an LED. This was accomplished at the conclusion of the semester and at the time of project demos. No Christmas tree has been constructed at this point in time, however by getting the LED to turn on by using I2C we are optimistic about what we could have achieved with this knowledge. Using the command bytes, we would have sent the PCA9532 two different PWM rates and set half of the LEDs on the tree to one PWM and half to the other. This would have been the initial light assembly and appealing to the customer. The existing .1 klocs of code would have expanded to about .15 klocs of code if accounting for all the additional LED and PWM programming over I2C. This project in general was troublesome but rewarding because I2C was learned and was an area that was too complex to be covered during the semester.

5.1 Bill of Materials

MSP430F5529: $12
PCA9532: $1
Resistors and wires: $3

6 Appendix

include ¡msp430.h¿

unsigned char *PTxData; // Pointer to TX data //unsigned char TxData[] = 0x00, 0x00; unsigned char TXByteCtr; //Defines TxByte Counter

unsigned char ConfigReg = 0x00; // Forces ConfigReg to be 8 bits unsigned char Data = 0x00; // Forces Data to be 8 bits

//0x03 (PWM0) is PWM register 0 //0x05 (PWM1) is PWM register 1 char LS0 = 0x06; // LED0 to LED3 selector char LS1 = 0x07; // LED4 to LED7 selector char LS2 = 0x08; // LED8 to LED11 selector char LS3 = 0x09; // LED12 to LED15 selector

unsigned char TxData[] = // Table of data to transmit 0x06, 0x55

;

int main(void)

unsigned int i;

USCI$_W DTCTL = WDTPW+WDTHOLD; //StopWDTP3SEL| = 0x03; //AssignI2CpinstoUS$ $UCSWRST; //EnableSWresetUCB0CTL0 = UCMST+UCMODE_3+UCSYNC; //I2CMaster, sy$ $UCSSEL_2+UCSWRST; //UseSMCLK, keepSWresetUCB0BR0 = 12; //fSCL =$ $SMCLK/12 = 100kHzUCB0BR1 = 0; UCB0I2CSA = 0x60; //SlaveAddressis048hUCB0CTL1 =$ $UCTXIE; //EnableTXinterruptTxData[0] = ConfigReg; TxData[1] = Data; while(1) for(i = 0; i <$

//TxData = 0x80; ConfigReg = LS0; Data = 0x7F; TxData[0] = ConfigReg; TxData[1] = Data; PTxData = (unsigned char *)TxData; // TX array start address // Place breakpoint here to see each // transmit operation. TXByteCtr = 2; // Load TX byte counter

UCB0CTL1 —= UCTR + UCTXSTT; // I2C TX, start condition

$_b is_S R_r egister(LPM0_b its+GIE); //EnterLPM0, enableinterrupts_{n o_o peration(); //RemaininLPM0untilalldata//isTX'dwhile}$

//———————————————————— // The USCIAB0TX$_I SRisstructuredsu$ $loadingTXByteCtrwiththebytecount. Also, TXData//pointstothenexttbytetotransmit. //-$ $- -$ $- -$ $-ifdefined(_{TI_C OMPILER_V ERSION})^{||defined(}{}_{IAR_S YSTEMS_I CC})^{pragmavector=USCI_B 0_V ECTOR}{}_{i nterruptvoidUSCI_B 0_I S}$