Department of Electrical Engineering
University of Washington

E E 474 Lab 2

Inputs, Outputs, and Time

The report and its contents are solely our team's work and cites outside references for work that is not our own.

Signature:
Xiaoyi Ling, 1428502
Tiffany Luu, 1435225
Pezhman Khorasani, 1640743

TABLE OF CONTENTS

Introduction

In this lab, we used the Beaglebone's GPIO pins to talk to an LCD display. We initialized the LCD display and used commands to send strings from the keyboard to the display. This is done in two separate terminals, where a sender sends characters through a pipe to the receiver, who displays it on the LCD display. We also utilized the LCD's display shifting to view longer lines, to view all 40 characters on each line of the display.
Additionally, we changed the LCD display to display music notes and played notes by sending a pwm output to a buzzer. Instead of reading from user input, the input was read from a file and the output was converted to display music notes from A-G.
We also flipped GPIO pin between low and high every 1 us and examined the waveform on the oscilloscope display to examine the function of the scheduler.

Design Specification

Talking to an LCD Display

Basic Implementation

Input: Keyboard
Output: LCD display and terminal window
Error check: The program is able to check if a file is opened successfully and if the pipe is made successfully. If not, the system will print error messages to the terminal and exit.

This program will send what the user inputs from keyboard to the LCD display.

- The receiver program that interfaces to the LCD should be run first. It creates a pipe and prints "Waiting for input…" to the terminal and waits for input from sender.
- Run the sender program that sends input to the LCD. The terminal will display "Enter input to send to the LCD display. Enter \n to go to the next line, or enter \c to clear the display" to instruct users." The user can enter input when "Enter input:" appears on the window.
- When user enters the input, the input will be sent to the pipe. The program will request the next input by printing to the terminal "Enter input:" again. On the other side, receiver program will receive input from the pipe and send the proper command to display the words to the LCD.
- The LCD screen displays 16 characters per line at a time. When a line passes the first 16 character, the display will shift to left one by one when a new character enters. When it reaches 40 characters, the cursor will move to the second line to print next character and the entire display will go back to the first 16 characters. When the second line has reached 16 characters, the entire display will again begin to shift until the second line reaches 40 characters. Then the entire display will be cleared and this process will start over again until the end of this input.
- The LCD can identify new line character and clear display character. When the user types "\n", it will goes to the next line. When the cursor is already in the second line, it will

clear the display the start over from the first line. When the user types "\c", the entire display will be cleared.

- The terminal window will display the input contents starting with "Received input:". After all the contents have been received, the program will wait for input again by printing to the terminal "Waiting for input…"
- The program will terminate when the user input an EOF character, or Ctrl-D.

Playing Music with the LCD Display

Input: File
Output: LCD display and terminal window
Error check: The program is able to check if a file is opened successfully and if the pipe is made successfully. If not, the system will print error message to the terminal and exit

This program will read inputs from text files and send it to the LCD display. Each letter from A-Z will represent a different pitch. It will also play sounds when it identifies 26 upper-case letters, and prints A-G depending on which note it played.

- The LCD/music program should be run first. It creates a pipe and prints "reading for file" to the terminal and then waits for the file input from sender.
- The file input program should be run next. It takes in files we want to read as arguments and sends them one line at a time through the pipe to the LCD/music program.
- Any non A-Z or "-" character received from the file input by the LCD program will print to the terminal normally. Newline characters are also read automatically.
- The LCD screen displays 16 characters per line at a time. When a line passes the first 16 character, the display will shift to left one by one when a new character enters. When it reaches 40 characters, the cursor will move to the second line to print next character and the entire display will go back to the first 16 characters. When the second line has reached 16 characters, the entire display will again begin to shift until the second line reaches 40 characters. Then the entire display will be cleared and this process will start over again until the end of this input.
- For the 26 uppercase letters, the program will tell the piano buzzer to output sound. Letter A equals to note A2, B for B2 and it goes up until Z for E5. However, the program will output the proper A-G musical letter to the LCD. For example, when H and O are identified from the pipe by the program, the program will print A to the LCD display. For other characters, LCD will display normally without sound outputted.
- If the "-" character is received, the sound from the piano buzzer will terminate. This means that typing "A -" will allow the user to play a longer note.
- The terminal window will display the input contents starting with "Received input:". After all the contents have been received, the program will for input again by printing to the terminal "Waiting for input…"

Hardware

Talking to an LCD Display

The pins of the LCD Display are connected to the Beaglebone power and GPIO pins in the following locations in Table 1 below.

Table 1: LCD to Beaglebone Connections

LCD	Beaglebone	Function
1	GND	Power: V_{SS} = 0 V
2	SYS 5	Power: V_{CC} = 5 V
3	~0.05 V	Power: powers the LCD display. A potentiometer was used to set a voltage close to 0 V to make the display readable.
4	GPIO 49	Register Select: 0 to read/write to the instruction register, 1 to read/write data as characters
5	GPIO 20	Read/Write: 0 to write to the LCD, 1 to read from the LCD
6	GPIO 115	Enable: Tells the LCD display when to perform an instruction, reads the posedge of the enable pulse
7	GPIO 66	Data 0: Input/output data to/from the LCD
8	GPIO 69	Data 1: Input/output data to/from the LCD
9	GPIO 45	Data 2: Input/output data to/from the LCD
10	GPIO 47	Data 3: Input/output data to/from the LCD
11	GPIO 44	Data 4: Input/output data to/from the LCD
12	GPIO 26	Data 5: Input/output data to/from the LCD
13	GPIO 46	Data 6: Input/output data to/from the LCD
14	GPIO 65	Data 7: Input/output data to/from the LCD, busy flag that tells when the LCD is running an instruction

Input from either the keyboard or a text file is sent from the computer to the beaglebone, which then translates to outputs to the LCD using the GPIO pins described above.
For the additional features, the piano buzzer that produces the music is connected to pin EHRPWM2A, a pulse width modulation generator.

Talking to Oscilloscopes

GPIO 60 is used for this part. This pin is connected to GPIO 60 in series with a blue LED and a 10kΩ resistor. The oscilloscope probe was connected to the pin and the ground to read the voltage, LED is used to show if the GPIO pin was active, and the resistor is used for circuit protection for the LED.

Software

The following is a short description of the C files and programs used to implement the LCD display and oscilloscope pin.

Talking to an LCD Display

commands.c: Contains functions for initializing the values of the GPIO pins, including export and access to GPIO direction and value files. Also contains several functions for commands for the LCD display. Includes the initialization process to get a blinking cursor, writing a character, return home, clear display, shift left, etc.

commands.h: Header file for commands.c. Initializes the functions in commands.c

Makefile: Compiles the programs for the project. Contains the following commands:

- make: Compiles the program that interfaces to the LCD (./run)
- make send: Compiles the program that sends user input to the LCD (./send)
- make music: Compiles the program that writes and plays music notes to the LCD (./music)
- make finput: Compiles the program that reads input from files and sends it to the LCD (./finput [filenames])
- make osc: Compiles the program that flips the GPIO pin up and down for oscilloscope testing (./osc)
- make clean: deletes all the executable files, .o files, and text editor files

Basic Implementation

input.c: Receives input from user by typing from keyboard. It sends the input to the pipe provided from the lcd.c program. This should be run before the lcd.c program.

lcd.c: It creates a pipe and receives user input from it, then displays it on the terminal and sends the characters to the LCD. It also deals with the situation of shifting, inputting a newline, and clearing the display when the instructions are given by the user.

Playing Music with the LCD Display

fileinput.c: Reads input files given as arguments by the user. It sends the input to the pipe provided from the filemusic.c program. If several files are given, it will send all of them in order until the last one.

filemusic.c: It creates a pipe and receives user input from it, then displays it on the terminal and sends the characters to the LCD. It also deals with the situation of shifting and receiving a newline. In addition, it will also play corresponding music notes when 26 upper-case letters are given, and output the corresponding music letter (A-G) to the LCD display.

Talking to Oscilloscopes

osc.c: Changes the output of a GPIO between 0 and 1 with 1us intervals.

Testing

Talking to an LCD Display

Figure 1 shows an example of text successfully displayed on the LCD display. The user inputted "Hello, world!" from the keyboard, as shown below.

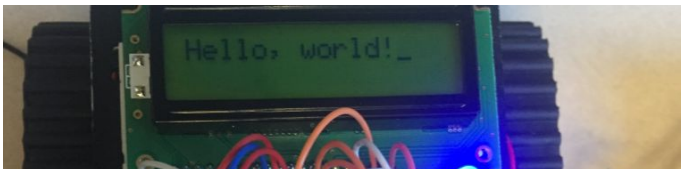

Figure 1. LCD displays "Hello, world!"

Talking to Oscilloscopes

An oscilloscope was used to view a pin that was being flipped continuously, to view the jitter. In the C program implementing this flip, we set the intervals between each change of the voltage to be 1 us. When we used an oscilloscope to display the change of a GPIO pin's voltage, we can see delays from the waveforms. In Figure 2, we can see the big gap appearing occasionally in the waveform. The reason for this huge delay is that when the Beaglebone flips the voltage of the pin, there are other background programs running at the same time. The Beaglebone's scheduler has to decide how much time to give to each process. Therefore, wherever there is a gap, the Beaglebone is likely allotting time to another process before returning to flip the GPIO pin again. In the two large breaks in Figure 2 below, we purposely ran our LCD display program in order to catch the delay caused when the scheduler decided to give time to the LCD program instead of the GPIO pin program. These delays happen occasionally since the Beaglebone is running background processes as well.

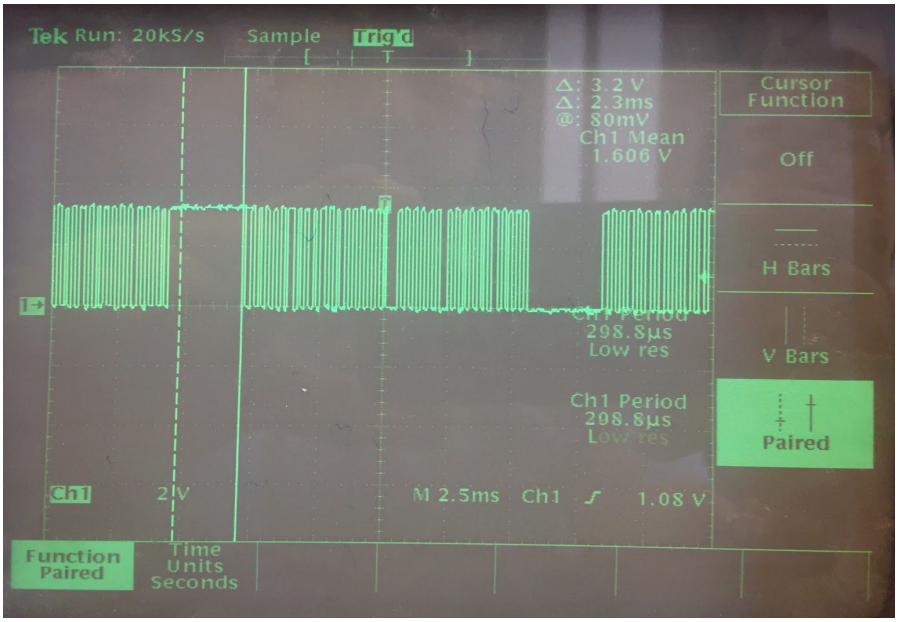

Figure 2. Oscilloscope waveform with 100us deviation

Figure 3 below shows a more zoomed in image of the jitter. Although the program we wrote was set to flip GPIO at 1 us, we can see via the cursor measurement that the GPIO is actually flipping every 100 us. We can also see that the positive/negative width of the pulses are also around 100 us. Although the negative width is shown to be larger than the positive width here, this changes depending on when you stop the oscilloscope image. This shows that there is a delay between the GPIO pin flipping up and down, of ~100 us.

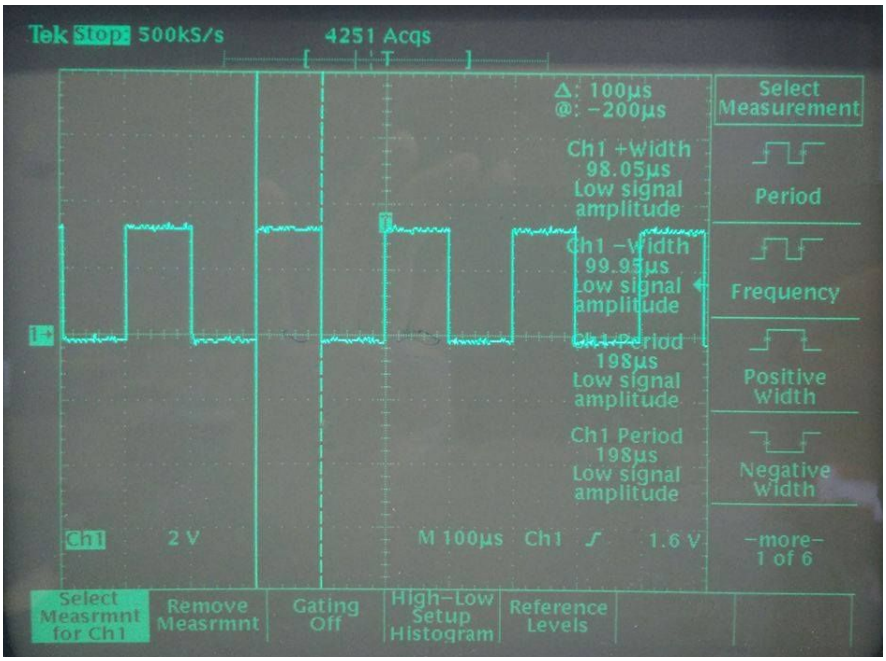

Figure 3. Oscilloscope waveform with 2.3us deviation

Conclusion

Throughout this lab we learned to initialize an LCD display and send different commands to write and shift characters to the display, by programming GPIO pins from the Beaglebone with C code. We also studied the scheduler jitter and examined how the scheduler allocates time for different processes by flipping a GPIO pin high and low and watching that output on an oscilloscope. Additionally, we practiced using the PWM pin by outputting pulses buzzer to play sounds, and combining that function with commands to the LCD display. One thing we may be able to improve is that currently we are reading from files and standard input keyboards. We could potentially expand this by connecting to external device such as a phone and read from it. There are also other LCD functions we did not utilize, such as read input from it, that we could potentially add to our current functionality.