Inputs, Outputs, and Time

Electrical Engineering 474 Lab 2
University of Washington

Name	Student ID
Denis Jivaikin	1432129
Novin Changizi	1434956
Ting-Yu (Jacky) Wang	1333301

INTRODUCTION

This lab focuses on the implementation of an LCD driver utilizing named pipes. By writing data to the named pipe through the client end, the server end will be able to read the data from the named pipe and push the data onto the LCD. In addition, a Hangman game was developed based on the aforementioned named pipe technique. Finally, this lab also allowed the observation of scheduler jitter of the operating system within the BeagleBone using an oscilloscope / logic analyzer.

HARDWARE OVERVIEW

For this lab, the BeagleBone is connected to the Newhaven Display LCD as shown in Figure 1. In order to adjust the contrast of the LCD, a 10 kOhm turn pot was used to adjust the voltage on the Vcontrast pin. For specific information on pin assignment see the implementation files.

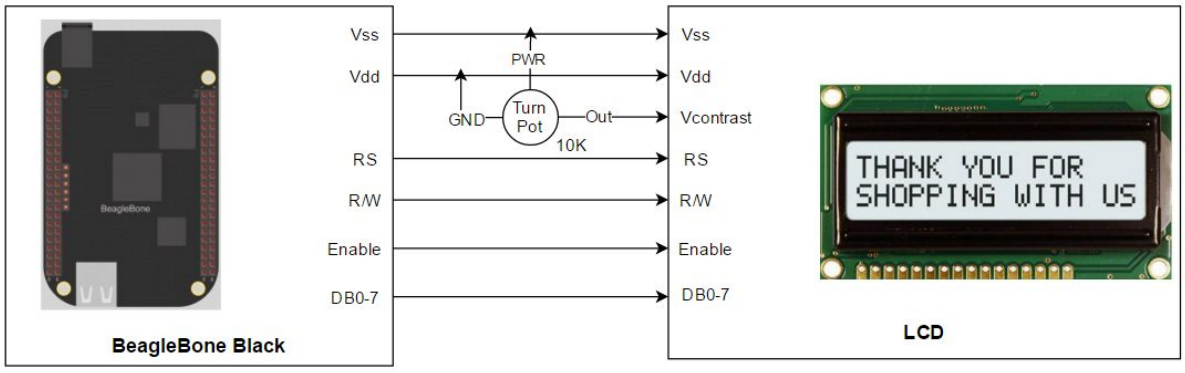

Figure 1: Hardware interconnection block diagram

SOFTWARE OVERVIEW

To communicate with the LCD, a pipe is established with *read_fifo* and *write_fifo* drivers, which were implemented in C programming language on the BeagleBone through the Linux kernel.

read_fifo driver in Figure 2 initializes all of the GPIO pins on the BeagleBone by exporting and changing the directions of the pins. After which a fifo pipe file is created in the /tmp directory, which is then constantly read from until the data is available, at what point the lcd display is initialized and data is sent to the display.

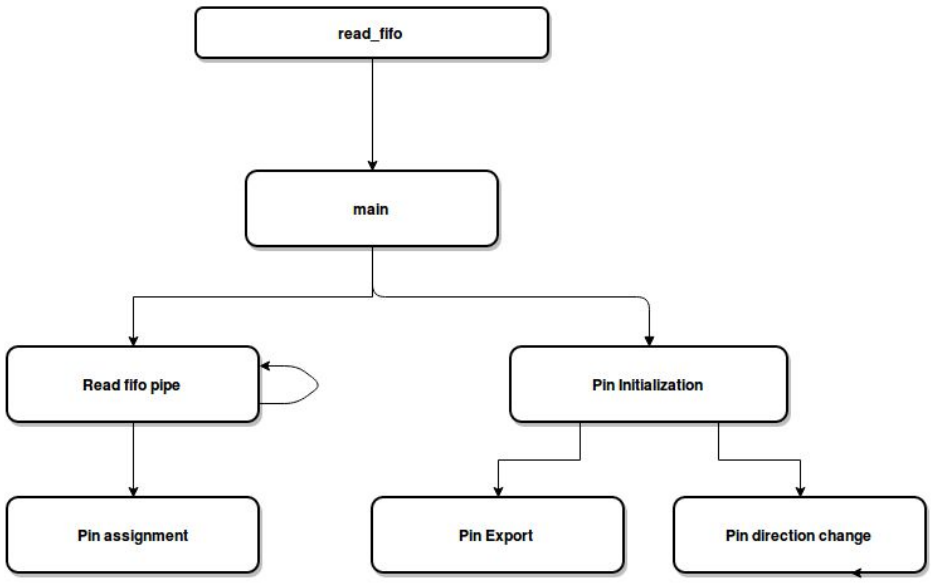

Figure 2. Functional diagram of *read_fifo*

write_fifo driver in Figure 3 is a program designed to continuously scan for user input which is then processed and sent to the pipe. Since the screen can only show 32 characters using the two-line option, a limitation was set on the user to maximum use of 32 characters.

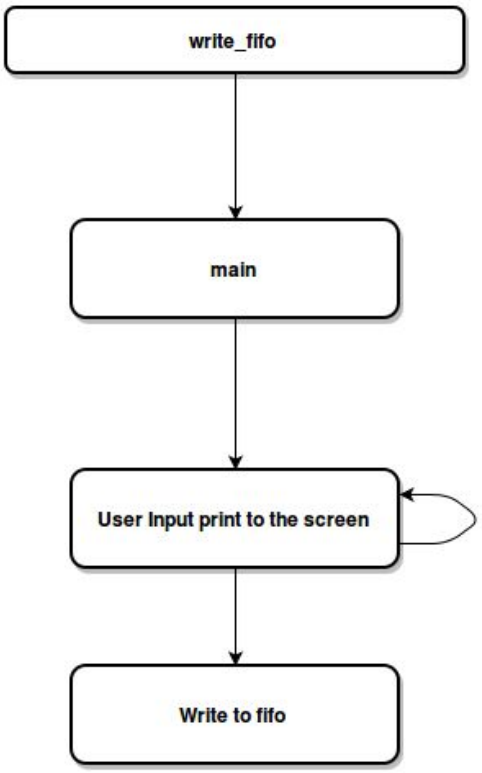

Figure 3. Functional diagram of *write_fifo*

read_fifo and *write_fifo* both have to be run as separate processes at the same time, since *write_fifo* would wait for the *read_fifo* to start reading and vice versa.

For the second part of the assignment a hangman game was implemented. *Hangman* is a library file written to play a hangman game. It stores the game state in a struct that contains information like the word, the state of the game, and previously guessed letters. You can initialize a game by calling init_game with the word and number of guesses allowed. Then you can use functions like get_state and guesses_left to extract information from the game struct. To guess a letter call guess with the game struct and the letter you are guessing. This returns an int from 0 to 3 each representing the outcome of that guess, like whether it was a correct guess and whether the game is over.

To implement the *Hangman* on the LCD another driver similar to *write_fifo* was implemented - *play_game* as seen in Figure 4. *play_game* follows the same format as *write_fifo* where it takes user input and sends game updates to the screen. *play_game* initiates the *Hangman* by calling the

init_game function with a random word that is generated from a local dictionary of 1000 most popular English words. After which *Hangman* is updated on the LCD screen by using get_state and guesses_left functions.

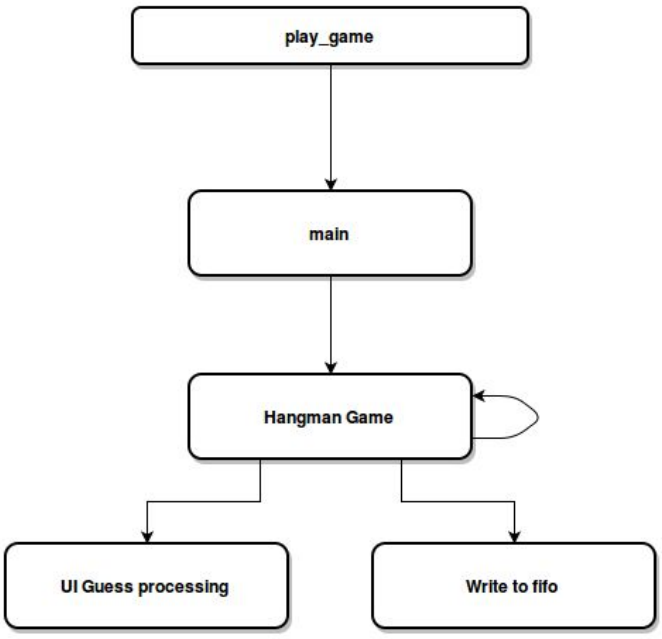

Figure 4. Functional diagram of *play_game*

All of the functions contain an exit option where the user can quit running the program which will safely unlink the fifo file, closing the communication pipe between *read_fifo* and others.

ABOVE AND BEYOND

Instead of using the oscilloscope to observe the scheduler jitter, a logic analyzer is used. The screen capture on the scheduler jitter can be found in the measurements section.

MEASUREMENTS

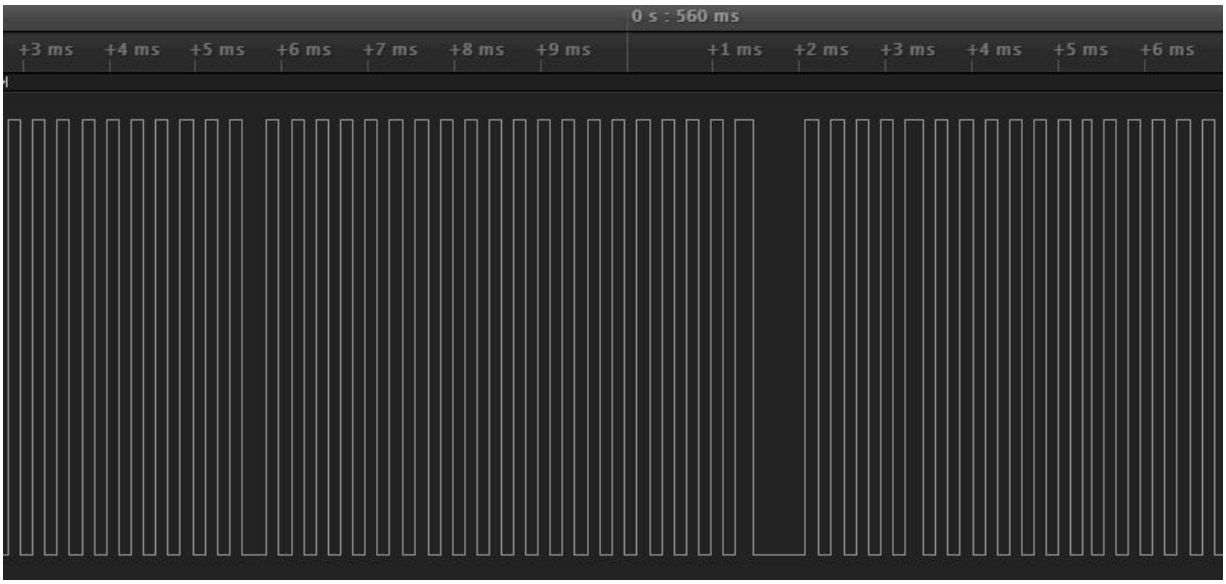

Figure 4: Scheduler jitter of Beaglebone

Figure 4 shows the scheduler jitter of the BeagleBone. In order to observe the jitter, a GPIO was setup to toggle (low-high-low) at the speed of 10MHz. The output of the GPIO is then measured using a logic analyzer to produce the resulting square waves.