# LKMs and Shift Registers

Electrical Engineering 474 Lab 3
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 kernel mode driver while using a shift register for data transfer in order cut down the number of GPIO usage. Advanced cursor shifting functionality (up, down, left and right) was also implemented. Furthermore, to make this lab even more interesting, we have incorporated the Leap Motion Controller to send hand gesture signals to the BeagleBone using bluetooth, thereby controlling the LCD and the motors through an H-Bridge.

## HARDWARE OVERVIEW

This lab is similar to lab 2. However, instead of connecting the data pins (DB0-7) directly between the BeagleBone to the LCD, a shift register is used in between them as shown in Figure 1.

In addition to the core project of this lab, we have implemented Bluetooth module BlueSMiRF RN42 module that is connected to the BeagleBone. Implementation of H-Bridge TB6612FNG was implemented to control two motors. Power to H-Bridge was supplied by a battery pack of 6V.

LeapMotion camera and Windows PC were used to transfer user inputs over Bluetooth.

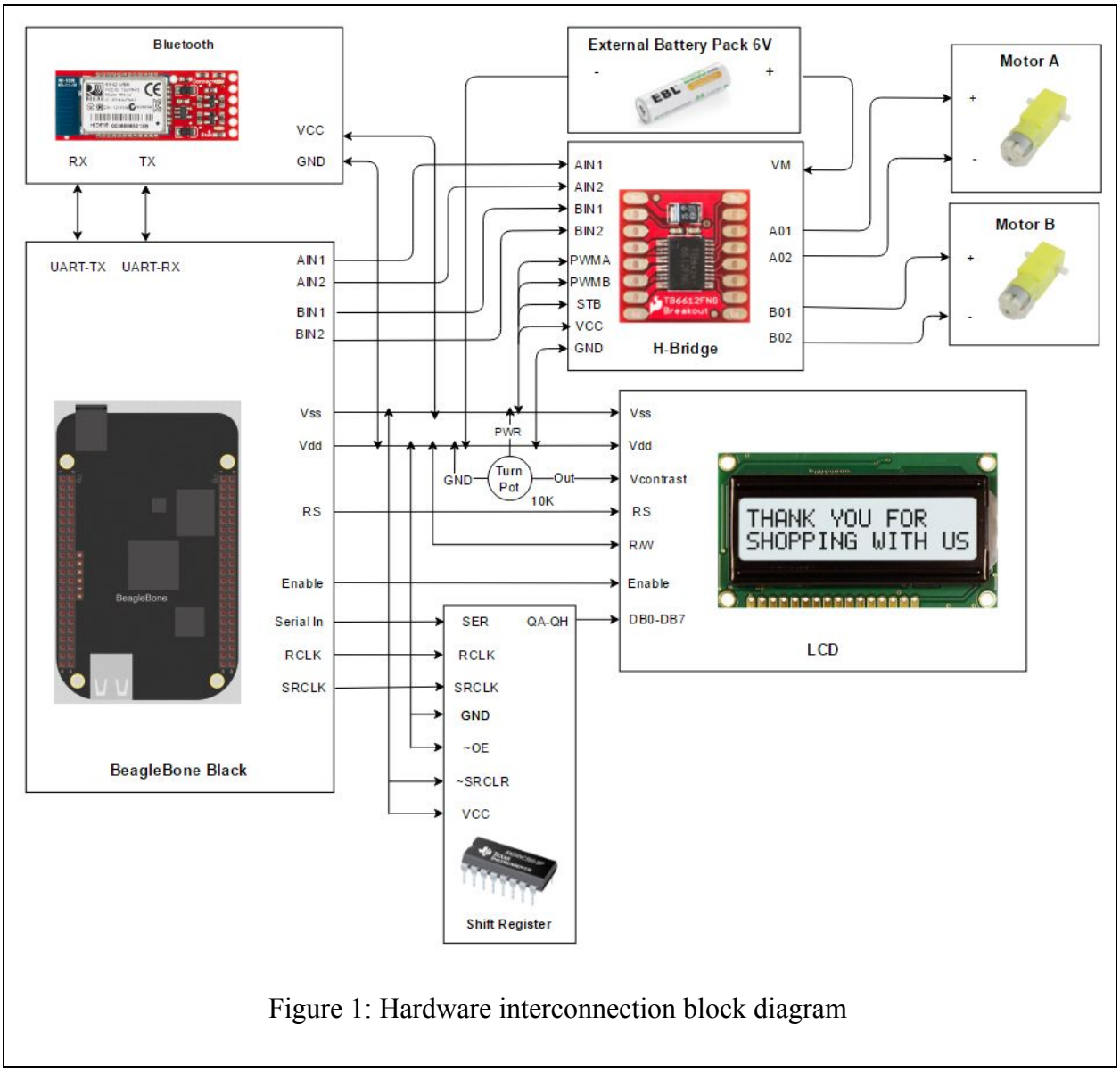


Figure 1: Hardware interconnection block diagram

## SOFTWARE OVERVIEW

To communicate with the LCD, a kernel module, *lcd_sr.ko*, was implemented which allows the creation of file */dev/lcd_sr_device* that can be written to using any userspace program. The kernel uses read, write, open, release, and owner file operations to communicate with the LCD. Some of the function the kernel program can handle:

- *lcd_start* - initializes the display for kernel use and sets parameters on it.
- *lcd_exit* - exits the kernel module freeing any malloced structures.
- *device_open* - called when the user uses open(...) call in userspace. Lifts the restriction on the semaphore allowing to write to the device.

- *device_close* - closes the device and returns access to semaphore using the userspace close(...) function.
- *device_write* - when in userspace write(...) is used, the information is copied over to the kernel where it is processed. Unless any of the special characters ('@' - clear, '>' shift cursor right, '<' - shift cursor left, '^' - shift cursor to opposite row) the driver writes to the screen using write_lcd helper function while keeping track of the cursor position. The program makes sure that only 32 characters can be represented on the two line display screen, shifts the cursor to the beginning when a string is being written to the end of the screen, overwrites previous characters if the cursor point to it, and keeps the cursor inside the lcd display. Some of the helper functions:
    - *write_lcd* - incorporates the shift register where the data is collected (8 bits) by shifting the bit mask until the st_clock is toggled, at which point the data is transferred to the LCD screen using GPIOs.
        - *sr_clock_pulse* - toggles shift register to shift the bits to storage register
        - *st_clock_pulse* - toggles storage register to shift all stored data bits out
        - *write_enable* - toggles the enable pin on the lcd to allow the LCD to pick up data from the storage register.

To test the LCD display *lcd_sr_test* executable was implemented to test kernel function and open(...), close(...), write(...) and read(...). The program prompts the user for continuous input where the user can enter a limited string of 32 characters.

To communicate with the motors two executables were developed *motor* and *uart_drive*.

*motor*:

- Exports gpio and pwm pins and changes the directions with previously used functions: export_gpio_pins, export_pwm_pins, and change_dir.
- Creates a fifo that both *motor* and *uart_drive* use.
- In a while(1) loop constantly reads the fifo for user input. If the user specifies an action, prints it to the screen and calls a corresponding function: forward(...), backward(...), left(...), right(...), stop(...). In the action functions input pins are toggled in different combinations to get the motor to turn in desired directions.

*uart_drive*:

- Calls a python script UART_SETUP.py to setup UART4 for use on BeagleBone.
- Opens the device (UART4) and modifies the settings of serial communication such as: baud rate, number of bits transferred, delays, control flow, and parity bit choice.
- Writes the initialization string to Bluetooth (UART4) to enter the configuration mode and change the name of the device.

- Uses while(1) loop to constantly read data from UART and write it to the pipe created by *motor*.

To test the Bluetooth (UART4) communication an Arduino terminal app on an android phone was used to send commands through.

We then used a LeapMotion device to capture hand gestures as controls to the motors. We defined 5 different hand gestures: forward, left, right, stop, and reverse. The Windows PC first connected to the bluetooth module on the BeagleBone which opened up the COM3 port. The the python code is run, which does the following:

- Opens the COM3 serial port that is connected to the fifo buffer set up by the bluetooth server
- Initializes the LeapMotion controller to grab images
- Wait until connection to LeapMotion
- Main loop:
    - Grab a frame from the LeapMotion
    - Find the closest hand in the frame
    - Use fist strength and palm positioning of the hand to determine the direction
    - Output that to the buffer
    - Wait 0.2 seconds before looping again