Dual-Mode LCD Character Entry System

Remington Davids & Bryan Dogariu
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
RemingtonDavids@Oakland.edu, BryanDogariu@Oakland.edu

Abstract -- This project was undertaken not only to create a character entry system for the LCD, but also to fully grasp how the LCD operates. The 4x4 keypad was implemented along the way and gave birth to the “Dual-Mode” title for the project. The LCD 1602 module proved to be front-loaded in terms of its complexity. Once the datasheet was fully grasped and the awkward timings were pinned down, the LCD was surprisingly simple to manipulate. With the proper controller for one’s needs, the LCD is a reliable middle ground for information display; more flexible and informative than the seven-segment display and far simpler and resource-friendly than a VGA output to a monitor.

- Introduction

The LCD module is a widely-used tool, from credit card readers to temperature sensors, its diversity cannot be understated. This report will break down the LCD 1602 module and its functionality into digestible pieces without delving too far into the esoteric workings of the microcontrollers inside the device.

This project allows a user to print characters two different ways onto an LCD. In Switch mode, the user can use switches on the FPGA to select any ascii value, then press a button to print that character. In Keypad mode, the user can hold a button on the keypad down, then press the print button to print that character to the LCD.

Our group unfortunately had a member drop the class, so our project was given permission to be less complex than originally intended. That being said, untangling how the LCD worked was still a fun and interesting challenge. Deciphering datasheets and testing out different controllers was a large portion of our research time. State machine and block diagram analysis were the two biggest skills we used in preparing for this project. Executing the project involved keypad interfacing, state machine modification, and a clear understanding of the LCD module’s instruction set.

- Methodology

The project can be encapsulated in three parts: The Setup, The Initialization, and The Output Process.

- *The Setup*

Before we could get started, we had to wire up the LCD and the keypad to the FPGA board. Bryan’s initial setup also included a power module, but more on that later. Figure 1 shows the layout of the LCD module, a deceptively simple one, especially considering the vast functionality of the device.

The VCC/GND are self-explanatory, though the FPGA’s 3.3V output powered the device perfectly fine. The Vo input was attached to a potentiometer between 3.3V and ground so that adjustments to the contrast could be made easily. The RS, RW, and E ports will be explained in further detail in the Initialization and Output Process sections, for now we just need to know that these three are all input ports. RS is connected to JA1, RW is connected to ground, and E is connected to JA2 of the FPGA board. The D0-D7 pins are all data input and they are connected to JB 1-4, as well as JB 7-10. The A and K pins are the power for the back light of the display. A is connected to positive 3.3V and K is connected to ground.

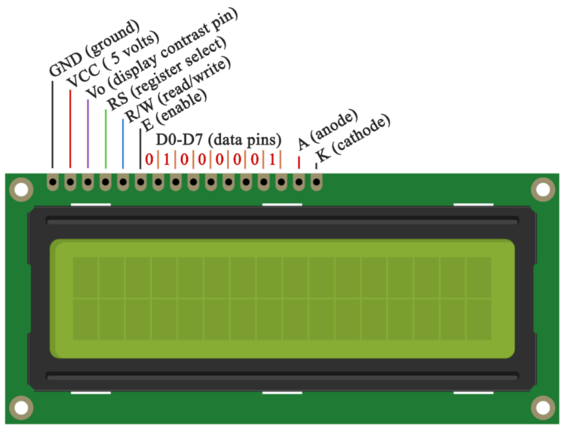


Figure 1: LCD Ports

Originally, a power module was used in Bryan’s setup. His LCD lit up and seemed perfectly normal, but it would never display any characters. After over a week of troubleshooting, the power module was discovered to be the problem. For unknown reasons, the power module was making the characters on the screen invisible, likely too much power (despite using the 5V and 3.3V outputs, respectively). After the power module was removed and the FPGA was used as the power source, the LCD worked perfectly.

The keypad (seen in figure 2) was a straightforward device to physically integrate (more on the complex logical integration later). The 8 pins were simply plugged into JC 1-4 and JC 7-10.

Figure 2: 4x4 Keypad

The keypad/pmod interface uses 8 pins to detect which of 16 buttons are pressed by the user. 4 pins are dedicated to the rows and 4 to the columns. When no buttons are pressed, the row pins will all pulled up to a logic high through 10kΩ resistors. Nothing happens immediately after a button is pressed. The column pins are all brought to a logic low one at a time, and when the column of the pressed button is brought low, the row pin will be brought low as well. Knowing the row and column that are low indicates which button is pressed.

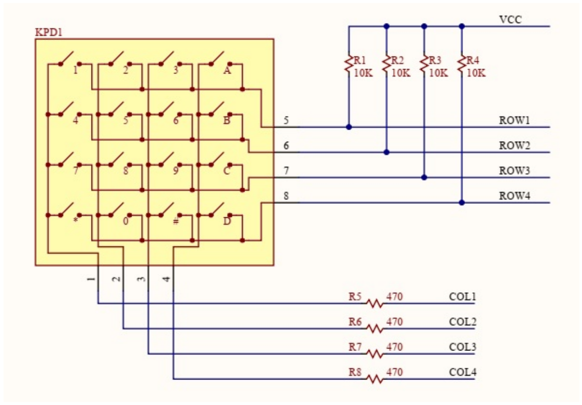


Figure 3: Keypad Circuit

The block diagram, shown below in figure 4, shows what our project encapsulates. The outputs on the right side (RS, E, and DB) go to the LCD module itself. The KP bus is the keypad input while the KP_buffer bus acts as both an input and an output. SW are the FPGA's switches. reset_n is the active-low cpu reset button on the FPGA board, and clk is the FGPA's 100 MHz clock. Forward, back, and clear are user input buttons for moving the LCD cursor forward and backward as well as for clearing the display and resetting the cursor to the leftmost position. pb is the user input button for printing the selected character to the LCD.

- *The Initialization*

The LCD's biggest hurdle, as mentioned before, was just how it operated. Making sure the LCD was receiving the correct data at every stage during its initialization was crucial. The timing of the instructions was key. The FSM diagram, shown in figure 6, goes over the details of the entire process. States 1 through 6 are considered the initialization phase. These states require specific timing so that the

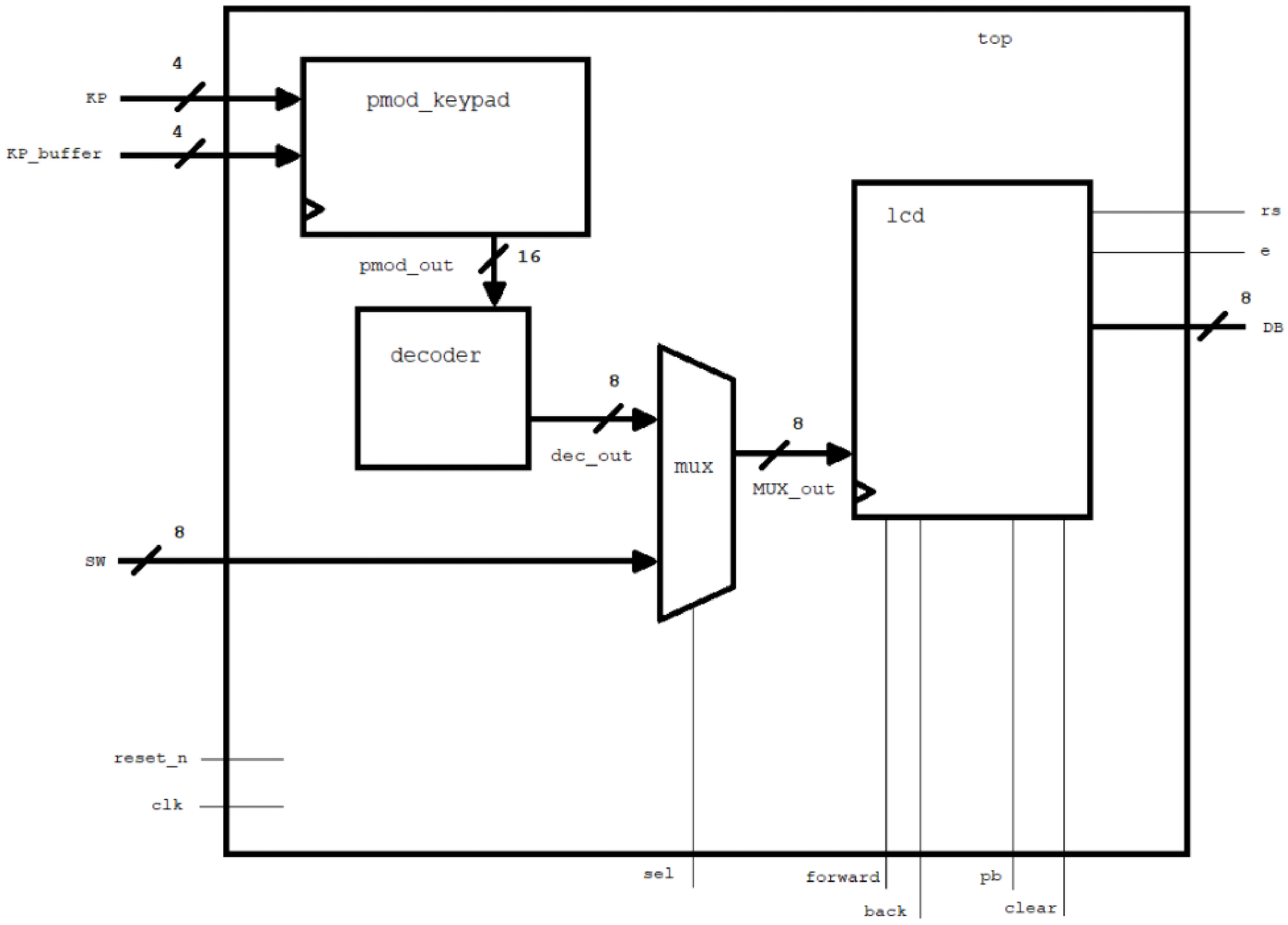


Figure 4: Block Diagram

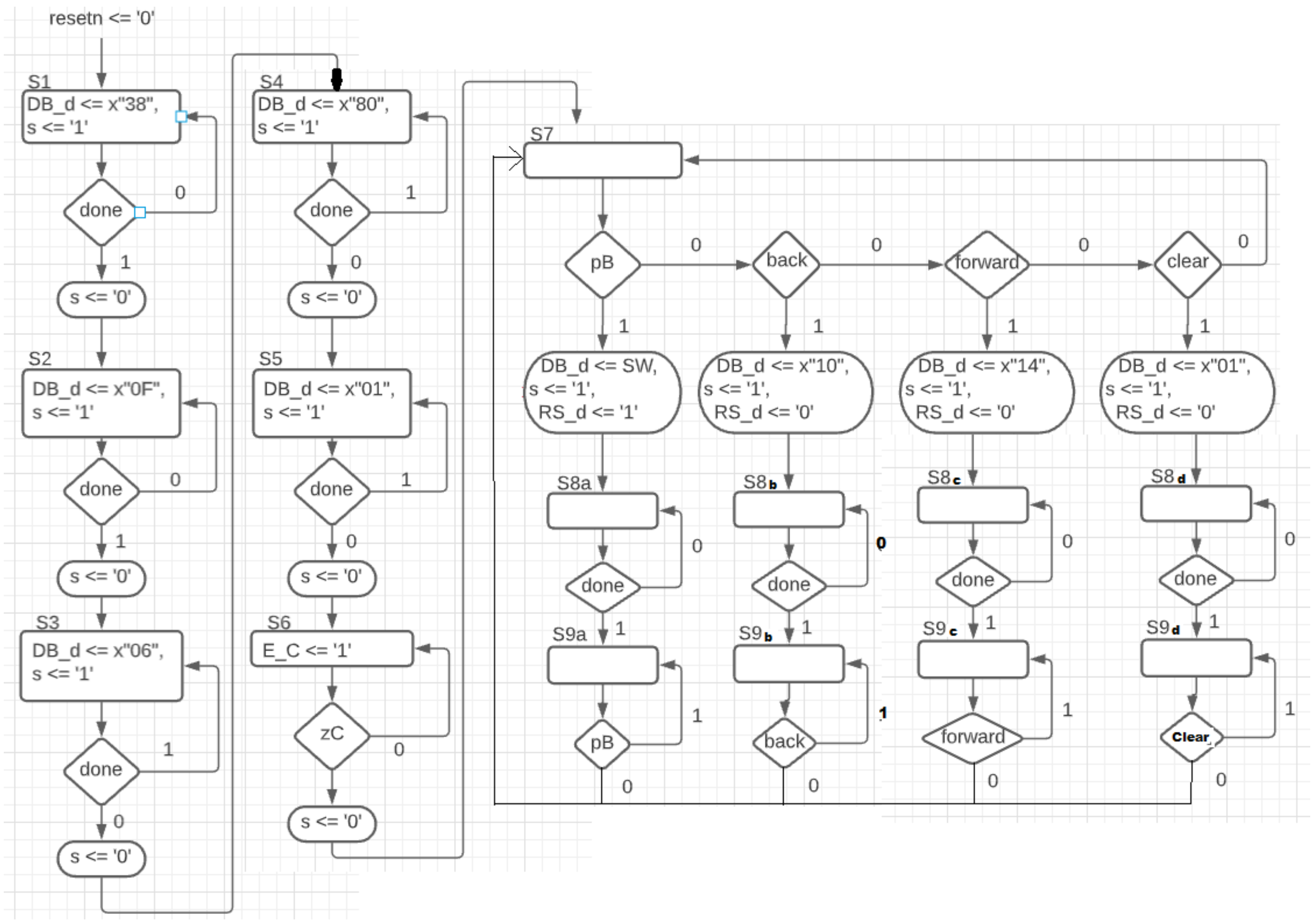


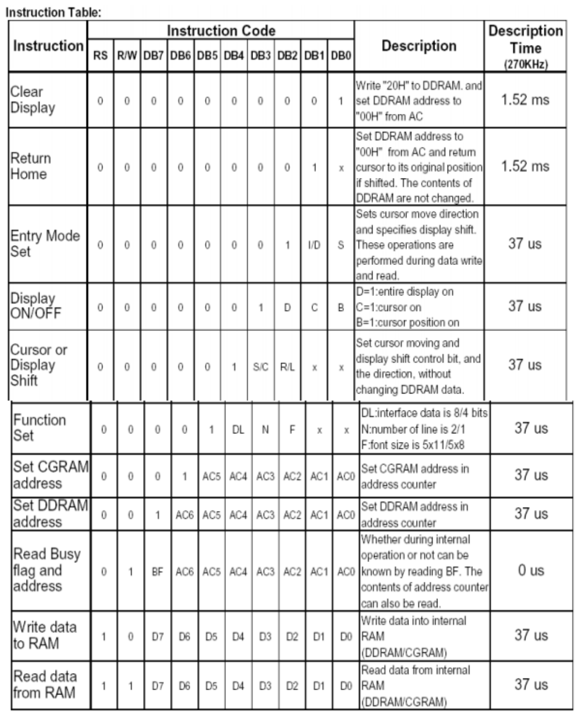
Figure 5: FSM Diagram of LCD.vhd

LCD can handle them. The lcd.vhd file provided by Professor Llamocca handles the timing for the initialization phase, using various counters to ensure that each step gives the LCD enough time to process each instruction before moving on to the next.

An important thing to note here is the RS input to the LCD has a default value of '0'. When RS is low, the LCD is in instruction mode. Any data sent to it will be interpreted as one of the instructions listed in figure 6. When RS goes high, the LCD writes the data it receives onto the display using an ascii lookup table in one of its internal microcontrollers. RS must be 0 for the entirety of the initialization phase. RS will be used more dynamically in states 7 and beyond (in the Output Process section).

One final detail to point out is that the initialization phase (states 1-6) only occur once after the system is reset. Once the system reaches state 7, the initialization is done and the device goes into operating mode.

In state 1, the LCD's D inputs are receiving "0011 1000" which, according to the instruction set shown below in figure 6, corresponds to the Function Set instruction. This particular set of bits sets the LCD to receive 8-bit data and to display data on both rows of the LCD. The LCD can receive data in either bytes or nibbles. Since I/O ports were not in short supply with the boards we were using, the 8-bit data mode made more sense and allowed for a wider range of characters. Since both lines were used for text, the font selection bit was not taken into account (there was only one font option; 5x8 dots).



Instruction Table:

Instruction	Instruction Code										Description	Description Time (270KHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM. and set DDRAM address to "00H" from AC	1.52 ms
Return Home	0	0	0	0	0	0	0	0	1	x	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.52 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 us
Display ON/OFF	0	0	0	0	0	0	1	D	C	B	D=1:entire display on C=1:cursor on B=1:cursor position on	37 us
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	x	x	Set cursor moving and display shift control bit, and the direction, without changing DDRAM data.	37 us
Function Set	0	0	0	0	1	DL	N	F	x	x	DL:interface data is 8/4 bits N:number of line is 2/1 F:font size is 5x11/5x8	37 us
Set CGRAM address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter	37 us
Set DDRAM address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter	37 us
Read Busy flag and address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0 us
Write data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM)	37 us
Read data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM)	37 us

Figure 6: LCD Instruction Set

In state 2, the LCD receives “0000 1111” which is the Display ON/OFF instruction. This sets the display to be on (very important), the cursor to be displayed, and enables the blinking of the cursor.

In state 3, the LCD receives “0000 0110” which is the Entry Mode Set instruction. This dictates the direction the characters are input into the LCD and, optionally, if the display shifts rather than the cursor. For our purposes, the cursor moves in the positive direction and the display does not shift.

In state 4, the LCD receives “1000 0000” which is the Set DDRAM Address instruction. Functionally, this simply sets the DDRAM address to 0. Since our Entry Mode Set instruction is set up to increment the DDRAM with each instruction, starting at 0 makes sense. Other applications could decrement the DDRAM address and start it at 1111111b or 127dec.

In state 5, the LCD receives “0000 0001” which is the Clear Display instruction. Without this, the display would contain the characters stored from the last use prior to reset.

State 6 is simply a waiting state for the counters to clear themselves. This synchronizes them for their use in the output process.

- *The Output Process*

This is where the magic happens. The default values for all variables in this section are ‘0’. States 7, 8, and 9 (and their variants) make up the output process. This process is a modification of the original FSM in the lcd.vhd file. The process essentially boils down to checking for an input button press, then initiates the instructions to carry out the corresponding command.

State 7 is where the device waits for an input. If pb is set to high, RS_d is set to ‘1’ (which feeds to the RS input of the LCD) and the DB_out (which feeds to the data input of the LCD) is set to SW. RS being 1 sets the LCD to write data to the display according to the ASCII value of the data being sent. SW will either be the switch input on the board or the keypad input, depending on the device’s mode.

If back, forward, or clear are set to high,. Firstly, RS_d goes to 0, indicating to the LCD that the data in DB_out is going to be instruction data, not ASCII data. x“10” shifts the cursor one space to the left, x”14” shifts the cursor one space to the right, and

"01", just like in the initialization phase, clears the display. It's worth noting that shifting the cursor position is actually the result of the address counter being shifted in the LCD's microcontrollers.

In state 8, the same timers from the initialization phase are making sure the instructions have enough time to occur before moving on to state 9. In state 9, the device is simply making sure that the button is released before going to back state 7. This helps prevent the same instruction from being input multiple times from one button press.

- Experimental Setup

The primary testing method of this device was trial and error and copious use of LEDs. Having LEDs turn on when a signal went high was a great way to approach testing since making a testbench for this device would have been very time-consuming and relatively unhelpful. If we needed to see a signal being sent to the LCD, a set of LEDs did the job much better than a testbench because all we needed to know was if the signals were reaching it or not. If we didn't see the output LEDs light up, we assigned more LEDs to more signals in the system until we could deduce what the problem signals were.

- Results

https://www.youtube.com/watch?v=__IQKvxvu_A&feature=youtu.be

Conclusions

The Dual-Mode LCD Character Entry System is far from perfect, but we accomplished what we set out to do. We learned about two very commonly-used devices and got them to interface with an FPGA board. We also learned that sometimes the most troublesome problems have the simplest solutions (like when Bryan's LCD started working by simply changing the power supply).

Even though this project heavily relied on two external files (Professor Llamocca's lcd file and the pmod controller from https://forum.digikey.com/t/keypad-pmod-controller-vhdl/13134) we still had to delve deep to learn how to make them do what we needed them to do. We heavily modified the lcd file to handle four different instructions instead of just one. We decoded the outputs of the pmod keypad block to map them to the proper ASCII values so that the LCD module could accept and interpret them.

There is a major fault in this device; sometimes the system locks up and doesn't respond to input and must be reset. We've spent a good deal of time trying to figure out exactly what causes it, but don't have anything conclusive. Our working theory is that it has no debouncer in the system as is. Because of this, the LCD is receiving multiple instructions at once. We don't actually check the busy flag of the LCD (instead we opt to just wait and give the LCD enough time to do each instruction one at a time). If we were to improve upon this device, checking the busy flag before sending more instructions and/or adding a debouncer would certainly help with the reliability of the system.

In the end, though, we just scratched the surface of what we could have done with this device. If we had more time and a full group, we could have made a sensor and output the data to the LCD. Or perhaps a calculator, anything with a dynamic display, really. The LCD is, for the most part, a tool meant to be utilized in something greater. Now that we have the knowledge of how it functions, implementing it alongside another device is what a future project has in store.

References

1) ST7066U Datasheet:
https://www.newhavendisplay.com/app_notes/ST7066U.pdf

2) ST7065C Datasheet:
http://www.hantronix.com/files/down/st7065c.pdf

3) LCD 1602 Module Datasheet:
https://ieee.ee.ucr.edu/sites/g/files/rcwecm1621/files/2018-10/eone-1602a1.pdf

4) ASCII Table: http://www.asciitable.com/

5) LCD wiki article:
http://wiki.sunfounder.cc/index.php?title=LCD1602_Module

6) Elegoo Arduino Kit Components:
https://images-na.ssl-images-amazon.com/images/I/D1oC-c3G5TS.pdf

7) 4x4 Keypad module info:
https://components101.com/misc/4x4-keypad-module-pinout-configuration-features-datasheet

8) Nexys A7 Datasheet:
https://reference.digilentinc.com/_media/reference/programmable-logic/nexys-a7/nexys-a7-sch.pdf

9) LCD instruction Set:
https://mil.ufl.edu/3744/docs/lcdmanual/commands.html

10) Alternative LCD controller we tried:
https://forum.digikey.com/t/character-lcd-module-controller-vhdl/12571

11) Professor Llamocca's ECE 4710 Moodle Page:
https://moodle.oakland.edu/course/view.php?id=249986