LCD



* Table taken from A Generic Keypad and LCD Tutorial.pdf
* Did not use L+, L- pins
* Pin No. 3 Vee is grounded for maximum contrast

lcd.h

* Pin and clock initialization
* Used ports GPIOB and GPIOE
* Commands were defined as hex constants based on the table below

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Command | Binary | | | | | | | | Hex |
|  | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |  |
| DISPLAY\_ON | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0C |
| CLEAR\_DISPLAY | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 |
| EIGHT\_BIT\_MODE | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 30 |
| LOCATION\_COMMAND | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 80 |

* two variables RS\_COMMAND = 0 for sending commands and RS\_CHAR = 1 for sending characters
* void lcd\_init(void)
  + Sets up GPIO pins using static method gpio \_init()
  + Set up 8-bit enable mode
  + Switch on the display
* void send\_data(uint8\_t data, uint8\_t rs\_line)
  + Set the data pins (D0-D7) individually to either 0 to 1 by examining ‘data’ via logical shift right
  + data: value to write to display register
  + rs\_line: either RS\_COMMAND or RS\_CHAR
  + Also enables to write to LCD (character/command)
  + Depending on rs\_line, the RS GPIO will be set to '1' (character) or '0’ (command)
  + Sets the enable line to high, puts a delay and sets it to low again



* void clear\_display(void)
  + Clears the LCD screen by sending 0x01 to the data lines
* void write\_char(char \*text, uint8\_t length)
  + Writes the characters given by ‘text’ to the LCD screen
  + Uses send\_data function with RS\_CHAR = 1
* void move\_cursor(uint8\_t location)
  + Moves the cursor to a specific location
* BitAction convert\_to\_bitaction(uint8\_t data)
  + Takes the value of data and if it is 0, it resets (remains at 0) if not it sets (remains at 1)
  + Used for setting the value of the bit action in the GPIO\_WriteBit function within the send\_data function

**Notes for testing:**

**When did the board not function?**

* No connection to Pin No. 5
* Solution:

- Pin No. 5 is grounded

* Connecting Pin No. 2 to Vdd
* Solution: Connected it to 5V

Keypad

* Used port GPIOD
* Define GPIO pins 1-4 to be rows and pins 5-7 to be columns
* void keypad\_configuration(void)
  + General keypad configuration
  + Screenshot for theory



* + Sets up the clock
  + Initializes the rows to be outputs with No PULL and sets their initial state to low (0)
  + Initializes the columns to be inputs with PULL UP and sets their initial state to high (1)
* void row\_check(void)
  + See the screenshots for theory
  + Switch rows to input mode and columns to output mode
  + Sets column pins (PD5, PD6, PD7) in output mode and no pull
  + Sets row pins (PD1, PD2, PD3, PD4) in input mode and pull up





* void column\_check(void)
  + Switch rows to output mode and columns to input mode (just like initial configuration)
  + Sets column pins (PD5, PD6, PD7) in input mode and pull up
  + Sets row pins (PD1, PD2, PD3, PD4) in output mode and no pull
* void keypad\_press\_check(void)
  + Determine the corresponding column and row of the key pressed
  + Pull the column inputs to see if a 0 is detected
  + When a column is 0, set isPressed variable to 1
  + Call row\_check() function to switch the column and row configuration (rows as inputs)
  + Pull the row input to see if a 0 is detected
  + Call column\_check() function to revert column and row configuration (columns as inputs)
* int get\_key\_pressed(int row, int column)
  + Determine the key number pressed based on the row and column parameters
  + row- the row number of the corresponding key
  + column- the column number of the corresponding key
  + Returns the key pressed on the keypad by examining the 2D array to map the row and column parameters to the key pressed
* **Notes for Testing**
  + 8 pins in all, but only 7 were used
  + Initially GPIO\_Pin\_4 was not used (trial and error) but row 4 was being detected incorrectly (showing as always pressed)
  + Replaced GPIO\_Pin\_4 with GPIO\_Pin\_0 and experienced the same issue
  + Used a ‘hack’ with a long if-statement checking whether all the other rows were set to 1 (not pressed) when row 4 was set to 0 (pressed)

**Motor**

Refer to these links for theory

* 1100us = 0 degrees
* Based on the data sheet
* 400us = 45 degrees
* <http://www.servocity.com/html/how_do_servos_work_.html>
* <http://www.kaltpost.de/?page_id=412>

pwm.c/.h

* void PWM\_GPIO(void)
  + Enable TIM4 clock and configure GPIO for PWM
  + Initialize pins GPIO\_Pin\_6 and GPIO\_Pin\_7 on Port GPIOB
  + Connect these pins to TIM4
* void PWM\_TIM(void)
  + Configure the TIM4 parameters
  + Set the values of prescalar = 72-1 and period = 20000-1
  + Set the initial pulse to 1500us
* void PWM\_configure(void)
  + Collection of the above functions

Servo.c/.h

* void alpha\_motor(float alpha)
  + Turn the motor based on the alpha angle
* void beta\_motor(float beta)
  + Turn the motor based on the beta angle