**CTEC1436/2024 PROJECT REPORT DEBBIE SANTANA 4507468**

**Abstract**

the presented project consists of a graphical animation in an DMD screen display using the Hitachi HD44780 controller in the Explorer 16/32 Development board. The device will show custom characters created in C language under the use of MPLABX 6.0 with its compiler XC16 and custom code from instructors Mike Boldin and Marc Cools under the course program.

The board has buttons that will be programmed to modify the animation; this animation consists of a jellyfish, using four buttons the speed as will change as well as stopping it and resuming its loop.

**Concept and Theory**

The Hitachi HD44780 is a widely used LCD controller that has been in existence for several decades. It is commonly found in a variety of electronic devices such as consumer electronics, industrial equipment, and embedded systems. One of the interesting applications of the HD44780 controller is in creating simple animations on LCD screens. In this report, we'll delve into the concept and theory behind LCD animation using the HD44780 controller.

Before diving into animation, it's essential to understand the basics of how LCDs work. Liquid Crystal Displays (LCDs) consist of an array of pixels arranged in rows and columns. Each pixel can be individually controlled to display either an on or off state, resulting in the formation of characters, symbols, or graphics on the screen. The HD44780 controller provides an interface between the microcontroller and the LCD, handling tasks such as generating character patterns, controlling the display, and managing the cursor position.

**Animation Fundamentals**

LCD animation involves dynamically changing the display content to create the illusion of motion or transformation. Unlike modern display technologies like LED or OLED, LCDs have inherent limitations due to their slow response times and lack of grayscale capabilities. However, by intelligently manipulating the display content, simple animations can still be achieved.

**Animation Techniques**

Several techniques can be employed to create animations on an LCD with the HD44780 controller:

1. **Character Animation:** By defining custom characters, it's possible to animate objects or symbols by sequentially displaying different character patterns. For example, to animate a moving object across the screen, a series of custom characters representing different positions of the object can be displayed sequentially.
2. **Segment Animation:** LCDs with segmented displays, such as those used in digital watches, can utilize segment animation. By selectively turning on or off individual segments, animations like scrolling text or rotating symbols can be implemented.
3. **Frame Animation:** Frame animation involves pre-defining a series of frames representing different states of the animation. These frames are then displayed sequentially at regular intervals to create the illusion of motion. This technique is commonly used for more complex animations.

**Challenges and Limitations**

While LCD animation with the HD44780 controller offers simplicity and low-cost implementation, it comes with its set of challenges and limitations:

1. **Limited Memory:** The HD44780 controller has limited CGRAM (Character Generator RAM) for storing custom characters, restricting the complexity and number of animations that can be created.
2. **Speed:** LCDs have inherent response time limitations, making real-time animations challenging to achieve. Careful optimization and timing considerations are required to ensure smooth animation playback.
3. **Resolution:** LCDs typically have low resolution and lack grayscale capabilities, limiting the detail and realism of animations that can be displayed.

**Hardware Design**

**External communications**

LCDRW connected to E, R/W and RS ports to the LCD screen via J19, a GND (ground point), VC and VCC with voltage input and to the right side the PIM connections.

A computer screen shot of a computer

Description automatically generated

*Picture from CTEC1432 – Explorer 16/32 Schematic R6/3*

There are also connections like DB0 to DB3 that don’t have a connection to the PIC.

**Table 1. LCD screen Overview**

|  |  |
| --- | --- |
| **Part number** | Truly TSB1G7000 |
| **Operating voltage** | +3.3V |
| **Input current** | 0.6A |
| **Interface type** | Parallel |
| **Response time** | 508.2ms (Min.) |
| 762.3ms (Max.) |

As for the connections with the PIM, there are six main ones as presented in table 2 with a short description.

**Table 2. LCD PIM Mapping**

|  |  |  |
| --- | --- | --- |
| **Device Pin # (80-Pin TQFP)** | **I/Os (pic connection)** | **Function** |
| 62 | RE12 | LCD D4 |
| 64 | RE13 | LCD D5 |
| 77 | R14 | LCD D6 |
| 79 | R15 | LCD D7 |
| 59 | RE11 | LCD RS ( Register Select Signal) |
| 57 | RE10 | LCD E (Read/Write Signal) |
| No connection | No connection | LCD R/nW (pulled low by R4) |

*According to PIM Information Sheet*

The project also makes use of the buttons embedded in the board, as so, the following would be their schematics:

A screenshot of a computer

Description automatically generated

*Picture from CTEC1432 – Explorer 16/32 Schematic R6/3*

This buttons, that function as inputs, are routed to the PIM (Plug in Module) that can be supported with different PICs. A datasheet is always provided with details on how the pins are routed to the components of the Explorer board for its correct use. Here are the basic connections and functions:

**Table 3. Button PIM Mapping**

|  |  |  |
| --- | --- | --- |
| **Device Pin # (80-Pin TQFP)** | **I/Os (pic connection)** | **Function** |
| 44 | RE9 | Button S4 |
| 1 | RB14 | Button S3 |
| 42 | RE8 | Button S6 |
| 39 | RE7\* | Button S5 (also LED D10) |

*According to PIM Information Sheet*

There are also GPIOs (General Purpose Input/Output) that the dsPIC33CH128MP508 uses for the programming of the inputs and outputs: Port A, Port B, Port C, Port D and Port E.

**Software Design**

Character generation and custom character made in C code are defined by the use of arrays of bytes that represent pixel patterns. An example of this can be Jelly\_1 or Jelly\_2 inside the code; each of those arrays contain 8 bytes and are later sent to the LCD controller using specific commands (LCD\_Cmd) to define them and store them in the CGRAM (Character Generator RAM) of the LCD.

The initialization process is handled in the “LCD\_Init” function where settings are configured to start the display, it includes setting up the communication protocol (SPI or I2C) and configuring display settings such as cursor behavior and display mode.

Button debouncing is also implemented in the “Button\_Check” functions to ensure reliable button input. This debouncing algorithm is used to prevent multiple buttons presses from being registered as separate events by introducing a delay (BTN\_DEBOUNCE\_MS) between each button press detection. For the interrupt handling, it is used for the implementation of the button debouncing. The timer interrupt is configured to trigger at regular intervals and is used to debounce inputs by incrementing a debounce counter and clearing the flag when the debounce period elapses.

This code defines sequences of custom characters to create a looping animation, for that it was necessary to implement the use of clearing the memory in the CGRAM before updating the display with new content, that way there is enough space for the custom characters to show in the LCD.

It is important to notice that the code is structured into modular functions to organize the software design and architecture; initialization routines, main loop and other functions collaborate in this code.

For this project a custom character of a Jellyfish was made, it was determined during the implementation process that a small 8bit per pixel character was the best for optimization of the code; this way it would also allow for faster reading. A snip of the custom character:

A screenshot of a computer

Description automatically generated

*Picture from “main.c” base project code.*

The main code for it is defined as follows:

**#include "main.h", etc:** inclusion of the internal header files for the functions to work properly.

**uint16\_t Jelly\_1[8], etc:** variables to define the custom characters pixels.

**InitCustomChar1, etc:** void function to assign the addresses to each custom character and show them in the display according to the position they need to be.

**LCD\_Char(0),etc:** calling of character according to code sequence that hey are written.

**LCD\_ToLineCol(2,2), etc:** moves the custom character to designed position.

**void main(){}:** void function that calls other functions to start the buttons, LCD and button check. Also runs the loop.

**void Start\_LCD(){}:** Contains the loop of the animation.

**void Button\_Check(){}:** check for the debounce in the buttons and calls for stop, resume, fast or slow animation speed according to the button pressed.

**void Start\_Buttons(){}:** Initiates each button.

**void StopAnimation(){}, etc:** Functions that allow the animation to be stopped, resumed, increases speed or decreases it.

**void \_\_attribute\_\_((\_\_interrupt\_\_)) \_T1Interrupt(){}:** clears interrupt flags and performs the debouncing of the buttons.

**Software Flowchart**

A screenshot of a computer

Description automatically generated

**Figure 2. Software Flowchart**

**Performance Tests**

The test made during the development of the project were checked to rectify any misspelling or missing information in the code. Mostly, the buttons generated the most common problem when setting them as well as the limitations in the CGRAM memory when creating the custom characters.

A summary of that is provided as follows:

**Start\_LCD:** at the begging of the project this was used to test the custom characters were appearing in the correct address.

**InitCustomCharX:** There was an initial function to test each binary to hex pixel demonstration where a table in C code was made outside of the main code to corroborate the correct placement of the pixels in each line and row.

**Button\_Check:** Worked as a function that review that the button was pressed correctly and allowed the btnDebounce to be called to prevent errors.

After that, a complete implementation of all the functions and service routine interrupts were developed together. First test of the code was unsuccessful.

The CGRAM address seemed to be overlapping during the animation sequences, at first it seemed to be something else like the looping being structured incorrectly but upon changing the addresses in correct order the custom characters were displaying but not entirely. Before the correction the display was showing each pixel as black and in the correct positions, after the changes the loop worked but the overlapping continued, meaning the CGRAM was not being cleared after the other custom character was being placed in the same addresses.

Then the buttons stopped to function after the code was compiled and run, there were missing functions that needed to be implemented to correct the errors they presented once the program was running.

**Conclusions**

The project was not successful in its entirety. The system, as of now, still presents errors in the display of custom characters and overlapping between each other.

What did I learn:

* Hardware not always performs as the data sheet show. Between theory and practice implementation of the documentation of a PIC microcontroller, this type of coding can be very sensible and prone to fail or have firmware that, like RE7, are connected to multiple functions that can make the programmers to take more time in coding. A lot of debugging happens because of that.
* Just as mentioned before, hardware can stop working. In the process of doing the project sometimes the cables would move a little and the code would give errors of building which caused the need to disconnect and connect many times the cables that connect the PIC to the computer.
* Because you disconnect and reconnect the cables, MPLAB needs to load and search again for the connection to the board, leading to more time to modify the code.
* Just because you change the addresses and the custom characters are overlapping, doesn’t mean the loop animation would not run. It just displays the incorrect patterns in the LCD.
* Interrupt service routines make the code more easy to manage when considering undesired interactions; in this case we had the buttons problem that they needed for a debounce to prevent the multiple readings of them that would cause noise and prevent them from working accordingly.

**Limitations of the System (CGRAM)**

* Not enough memory for the storage of various custom characters without the necessity of clearing the addresses each time.
* The memory only storages 8 bit per address.
* Limited custom character customization.
* Overlapping of Custom Characters displaying in the LCD prone to happen if not knowing how to locate addresses correctly.

**Limitations of the System (Buttons)**

* Buttons are prone to bounce when pressed (noise in signal) from lack of internal capacitor for debouncing.
* If more actions were required (like changing manually the animation) there is not enough buttons and external pins (device) would be required to be attached to the board.

**Suggestions**

* Limitations listed above need to consider when implementing this type of code.
* One of the buttons (RE7) is connected to both D10 and SW5 where the first is an active-high LED and the second an active-low button; enabling the internal pull up resistor inside the PIC would allow the button to correctly be sensed by the I/O port.
* Side effect of previous suggestion is the LED D10 being lit until SW5 is press.
* The animation can be upgraded by adding more “uint16\_t” variables and assigning the correct address.
* Cables can make the connection to the MPLABX to fail and not build, so make sure you have your cables in good shape, if necessary, buy new ones.