Lab Report 2: LCD Calculator

Kyle Truong

Summary

Lab 2 builds on top of the first lab, by improving the calculator to print double digit numbers. However, I decided to take it a step further, and programmed it to calculate any digit up to 10000. This is because after 10000, the HSC12 in unable to output proper decimal numbers, falling back to random four-digit integers, and displaying random sets of characters on the LCD. The calculator is also able to handle multidigit input, with an explanation coming in the code breakdown.

Code Breakdown

```
volatile char message_index_on_7segment_LEDs = 0;
volatile unsigned int counter_for_real_time_interrupt;
volatile unsigned int display_counter = 0;
volatile unsigned int counter_for_real_time_interrupt_limit;
//volatile int buf;
int cursor = 0;
```

The variable buf from Lab 1, responsible for indicating what's being displayed on the seven-segment display, has been replaced by a new variable called cursor**.** This variable is used to keep track of the position on the LCD and where the next character should be drawn. By starting at 0, the cursor will begin on the far left of the display.

```
int printres(int x) { //tail recursive function to print each character of the result
	if (x / 10 == 0) {
		DispChar(0, cursor++, x % 10 + '0');
		return (x % 10);
	} else {
		printres(x / 10);
		DispChar(0, cursor++, x % 10 + '0');
	}
}
```

This new printres function was designed to help print out results with more than one digit. It works by dividing the result by 10 until the result is less than 1, then printing out that character. It then recursively prints out the next digit by printing the remainder of the result divided by 10. For example, if the result was 12345, the function would be divided by 10 4, times, then print out the remaining integer as follows:

1.2345 % 10 = 1.2345 → Prints 1

12.345 % 10 = 2.345 → Prints 2

123.45 % 10 = 3.45 → Prints 3

1234.5 % 10 = 4.5 → Prints 4

12345 % 10 = 5 → Prints 5

DispChar is a predefined function in the lcd.h, which requires three parameters: row, column, and the character to print. For this program, the row will always be 0 (the same row), since not enough

characters will be generated to overfill the first row. Cursor refers to the global variable mentioned above, attached to it is '++', which increases cursor by one. The final requirement for the function is the character to be drawn at the given coordinates. To sum it up, every time a character is printed, the cursor counter increases by one, allowing the draw head to move to the right.

```
int main(void) {
	set_clock_24mhz(); //usually done by D-Bug12 anyway
	DDRB = 0xff;
	rti_init(0x11, 10);
	__asm("cli");
	//enable interrupts (maskable and I bit in CCR)

	DDRH = 0x00; //for push buttons
	KeypadInitPort();

	DispInit(2, 16);
	DispClrScr();
	//DispStr(1, 1, "Calculator")
	int a = 0;
	int b = 0;
	char op = '!';
	int result = 0;
	while (1) {
		char x = ~PTH;
		if (x)
			printf("DPKey: %x\r\n", x); //print on serial console
		unsigned char c = KeypadReadPort();
		if (c != KEYPAD_KEY_NONE) {
			switch (op) {
			case '!': //first input: will equal 0 if A or B is pressed
				if (c >= '0' && c <= '9') {
					a = a * 10 + (c - '0');
					DispChar(0, cursor++, c);
				}

				if (c == 'A') {
					op = c;
					DispChar(0, cursor++, '+');
				}
				if (c == 'B') {
					op = c;
					DispChar(0, cursor++, '*');
				}
				break;
```

This first part of the main code defines the variables that will be used within this piece of code:

a – represents the first integer in the calculation, additional place values are added by multiplying the current value by 10, then adding the new value

b – represents the second integer in the calculation, additional place values are added in the same method as 'a'

op – represents the position of the calculator (1st or 2nd integer input) and whether the calculator will be adding or multiplying

result – the result of either adding or multiplying the two integers, stores the variable to be printed after the equal sign.

Nothing happens until one of the buttons on the keypad is pressed, then that variable is stored in unsigned character variable *c*.

We then see the input of the first integer. Since 'op' initially defaults to '!' and after clearing, the switch case jumps to the '!' case. Where variable c is checked if it is an integer, or an operation. If it is an integer, it's added to variable a, by multiplying the current value by 10, then adding c. For the first integer, since a = 0, multiplied by 10 remains 0, then variable c is added. This process is continued until a different button is pressed. If the program detects that either A or B on the keypad was pressed, it stores the chosen operator as the op variable, then displays the corresponding character operator on the LCD. The program would then jump to the corresponding switch case, either A or B, which is important for tabulating the final result.

```
case 'A'://second input for addition
      if (c >= '0' && c <= '9') {
            b = b * 10 + (c - '0');
            DispChar(0, cursor++, c);
      }
      if (c == 'D') {
            DispChar(0, cursor++, '=');
            result = a + b;
            printres(result);
      }
      break;

case 'B': //second input for multiplication
      if (c >= '0' && c <= '9') {
            b = b * 10 + (c - '0'); //increase place value of
            current input by multiplying by 10 then adding new
            digit
            DispChar(0, cursor++, c);
      }
      if (c == 'D') {
            DispChar(0, cursor++, '=');
            result = a * b;
            printres(result);
      }
      break;
}
```

The two cases for A and B function exactly the same, the only difference is the mathematical operator they conduct at the end.

As mentioned before, the method for calculating 'b' is the exact same as 'a'. The calculator will continue inputting digits until the D is pressed on the keypad, when it will print '=' then either

adds or multiplies depending on the user's request. It then runs the result through the printres function mentioned above.

```
                    if (c == 'C') { //always checking for this character
                           cursor = 0; //reset count to 0, starts equation from
                           beginning
                           a = 0; // a and b are reset to 0 as a placeholder
                           b = 0;
                           op = '!'; //op is reset to ASCII ! as a placeholder
                           DispClrScr();
                           //DispStr(1, 1, "Calculator");
                    }
                    printf("Debug: %d\r\nKeypad: %c\r\n", result, c);
             }
      }

}
```

Like Lab 1, the program is constantly checking for the input 'C' on the keypad, and if it's detected, then the LCD is cleared, and all the variables are reset back to their original values. The printf statement was used for debugging purposes, where 'Keypad' indicated what key was pressed, and 'Debug' was interchanged for various testing functions, i.e., printing individual characters, printing results, and calculating input totals.

Challenges

Overall, this lab wasn't extremely challenging, but the few bumps along the way slowed down our progress. On the first day, the group discovered the 'DispChar' function inside the lcd.h program and used it to program one-digit calculations by modifying the Lab1 code to change the display to the LCD rather than the seven-segment display. However, this was not the requirements of the lab, and demonstrates limited knowledge of programming within C.

The first challenge was increasing the value of the inputs by modifying the place values. For instance, if you press 1, then press 1, the first one should move from the one's column to the ten's column. I originally overthought this, thinking that a place values variable would be required, and multiplying the value by ten to the power of the place values, which in hindsight was overcomplicated an not a good idea, since the HCS12 can only handle up to 10000.

Another challenge that I had to overcome was printing the result. I knew that each character needs to be printed individually, since converting the integer into a string is difficult without the basic C libraries, it's easiest to print each character individually. The thing that made this difficult was the use of head recursion vs tail recursion. For the longest time I used head recursion, to no success; the function would print the result backwards. I spent a lot of time experimenting within Python to help me better understand the logic of it. Eventually, I managed to figure out that by printing the previous result after running the function again, I could get it to print in the right order.