Memo to: Randy Larimer
From: Erik Andersen
Date: April 2, 2015
Regarding: EELE 465, Lab3 Project – Analog Temperature Sensor

Summary:

The purpose of this lab is to measure the analog voltage output from an external temperature sensor and an internal temperature sensor on the MC9S08QG8 microprocessor and display the ambient air temperatures from both devices using the MC9S08QG8 microprocessor and a breadboard with a custom built circuit. An LED on the LED array will simply act as a heartbeat LED blinking on once every second as long as there is power to the microprocessor.

Preliminary Solutions:

First, the sixteen position keypad will be used to enter a value 1 through 9. That number will be displayed on the top line of the LCD. N readings will be taken of the voltage output of the external temperature sensor and the onboard temperature sensor on the microcontroller where N is the entered value. Upon finishing each conversion the program will calculate an average of the converted values. After averaging n results, the ambient air temperature in degrees Kelvin and Centigrade on the top line of the LCD for the LM19 (external temperature sensor) and on the second line of the LCD for the microcontroller (internal temperature sensor) will be displayed.

Second, according to the user manual of the LCD on power up of the microprocessor the LCD has to always be properly initialized. The LCD was initialized the same way as it was in the last project.

Third, developing a flowchart gave us a place to start from when it came to building the keypad read write code. The solution involved creating a new file called keypad read write within the sources folder of the project that contained all the code for reading from the keypad. The startup code jumps to the keypad read write subroutine which then after a long process of determining which keypad button was pressed that eight bit code is stored into memory. Based on what that eight bit code is the main code will branch to the correct subroutine. Each subroutine includes the correct code to write the correct value to the LED's and display the correct character on the display.

Setup:

The setup involved interfacing the MC9S08QG8 microprocessor with the desktop computer by USB cable to be used for downloading the firmware to the processor for debugging and wiring the keypad, LED array, LCD display, LM19 (external temperature sensor) and DEMO9S08QG8 onto the breadboard. The breadboard circuit contains three d-flip-flop packages, one bus transceiver package, one LM19(external temperature sensor) and one de-multiplexer package as well as an LCD display, potentiometer and some capacitors in order to make the LCD display visible. The LCD display is attached to the breadboard by soldering a 16-pin header to the pins so the LCD can be attached to the breadboard. The d-flip-flops and bus transceiver are used to read the keypad press store it and then write the correct value to the LED's and LCD display based on the key pressed. All instructions from the MC9S08QG8 microprocessor go through the de-multiplexer before they go to the three d-flip-flops and the bus transceiver. Further, the instructions for the LCD display come directly from the MC9S08QG8 microprocessor the four bit bus line and an inverted signal from the de-multiplexer. Additionally, the LM19 (external temperature sensor) is read directly from the Port A2 of the MC9S08QG8 microprocessor. The reset button on the MC9S08QG8 microprocessor was also enabled.

Solution – Keypad Read Write:

First, set the PTBDD register to an output and store binary number 11110111 into the correct memory address in order to read the first row of the keypad. Then, logical shift left that binary number in order to test the next row of the keypad if the first row of the keypad had no press. Next, check to make sure all rows have been tested if not reinitialize memory address sixty until all rows have been checked. Once all rows have been checked branch to the roughly one second delay loop which was determined by using a stop watch and guessing. After the delay loop branch to the re read subroutine to recheck the button press to verify that it is still pressed after one second. Lastly, store the pattern to the correct memory address and based on the pattern stored to memory which is how many readings of the temperature sensor we want to take, branch to average readings and read the temperature sensors that many times.

Solution – LCD Initialization

First, after power is on load values into the correct addresses for the delay loop subroutine to delay for 15 milliseconds. Then, clear bits 0 and 1 of PTAD and move binary number 00111100 into PTBD and then bring the clock back up by setting bit 3 of PTBD (function set command). Now, store the correct values for the delay loop subroutine to delay for 4.1 milliseconds. Next, write the function set command code as written on the second line of this paragraph. Then, store the correct values for the delay loop subroutine to delay for 100 microseconds. Next, write the function set command code same as before. Now, check the busy flag by jumping to the busy flag check subroutine. Refer to the busy flag check flowchart at the end of the report to see how the busy flag is checked. Then, clear bits 0 and 1 of PTAD and move binary number 00101100 into PTBD and then bring the clock back up by setting bit 3 of PTBD (function set). Again, check the busy flag by jumping to the busy flag check subroutine. Next, write the function set code same as before. Then, clear bits 0 and 1 of PTAD and move binary number 10101100 into PTBD and then bring the clock back up by setting bit 3 of PTBD (set N and F). Now, check the busy flag by jumping to the busy flag check subroutine. Then, clear bits 0 and 1 of PTAD and move binary number 00001100 into PTBD and then bring the clock back up by setting bit 3 of PTBD; again, clear bits 0 and 1 of PTAD and move binary number 10001100 into PTBD and then bring the clock back up by setting bit 3 of PTBD (Display off command). Now, check the busy flag by jumping to the busy flag check subroutine. Then, clear bits 0 and 1 of PTAD and move binary number 00001100 into PTBD and then bring the clock back up by setting bit 3 of PTBD; again, clear bits 0 and 1 of PTAD and move binary number 00011100 into PTBD and then bring the clock back up by setting bit 3 of PTBD (Clear display command). Now, check the busy flag by jumping to the busy flag check subroutine. Then, clear bits 0 and 1 of PTAD and move binary number 00001100 into PTBD and then bring the clock back up by setting bit 3 of PTBD; again, clear bits 0 and 1 of PTAD and move binary number 01101100 into PTBD and then bring the clock back up by setting bit 3 of PTBD (Entry Mode Set). Now, check the busy flag by jumping to the busy flag check subroutine. Then, clear bits 0 and 1 of PTAD and move binary number 00001100 into PTBD and then bring the clock back up by setting bit 3 of PTBD; again, clear bits 0 and 1 of PTAD and move binary number 11111100 into PTBD and then bring the clock back up by setting bit 3 of PTBD (Display On). Lastly, jump to subroutine done which simply jumps to keypad read write subroutine with some initial setup. The LCD is finally properly setup.

Solution – Calculate Temperature of both Sensors in Celsius and Kelvin

First, enable Port A as an input (set bits one and two) so it can be used to read the external temperature sensor. Next, initialize the LCD display as described in the previous section. Then, move 00000000 into the register ADCCFG to enable 8-bit mode. Now, clear bits 4, 5, and 6 of the ADSC2 register. Next, clear bits 6, 5, 2 and 0 and set bits 4, 3 and 1 of the ADSC1 register. This initializes the analog to digital converter on the MC9S08QG8 microprocessor. Then, the characters “Enter n:” are written to the LCD display as described in the next section. Now, the heartbeat LED is toggled with an XOR mask at the rising edge of a clock. The first key is loaded into the correct address to be scanned by the keypad. Everything described in this first paragraph happens on power up of the MC9S08QG8 microprocessor.

Second, based on the value stored in memory from a key press of 1-9 we then branch to the correct subroutine that then stores that many readings in memory. Next, branch to the average readings subroutine. By checking the COCO flag we verify that a reading was taken and each time a reading is taken the number of readings to be taken is decremented by one. Each time a reading is taken it is stored as counts and divided by the number of readings taken in order to average the results. Also, the remainder of each division is kept track of and averaged the same way as a reading. After all readings have been taken the average remainder is combined with the average actual readings and stored in a single memory address. The average readings for the internal temperature sensor and the external temperature sensor are taken and calculated at the same time. The internal temperature sensor average readings at the end of the subroutine branch to convert_temp to convert the counts into a temperature. The external temperature sensor average readings are converted into a temperature after the above is completed by subtracting the counts from 144 base 10 and storing that value into memory as a Hex value. This is the transfer function of the LM19 for a temperature range of -10 to +65 degrees Celsius. 144 base 10 is 1.8641 volts converted to counts for our system. To convert to counts multiply your voltage by the max counts which is 255 in our case since we are in 8-bit mode and divide by V_{cc} which is 3.3 volts. Next, branch to the displayx subroutine in order to display the external temperature on the first line of the LCD display.

Third, the internal temperature sensor counts are converted to a temperature in the convert_temp subroutine with the transfer function Temp = 25 – ((V_{TEMP} – V_{TEMP25})/m). If the average readings counts ($V_{TEMP)}$ are greater than V_{TEMP25} in counts than we branch to cold slope; if the average reading counts are less than V_{TEMP25} in counts than we branch to hot slope. The difference between hot slope and cold slope is the value of m that we divide by. Both the subtraction is divided by m (a constant converted into counts) and the remainder is divided by m. Then the two values are combined and stored into memory and subtracted from 25. The result is the temperature recorded by the internal temperature sensor in Hex. Finally, branch to the display subroutine in order to display the internal temperature on the second line of the LCD display.

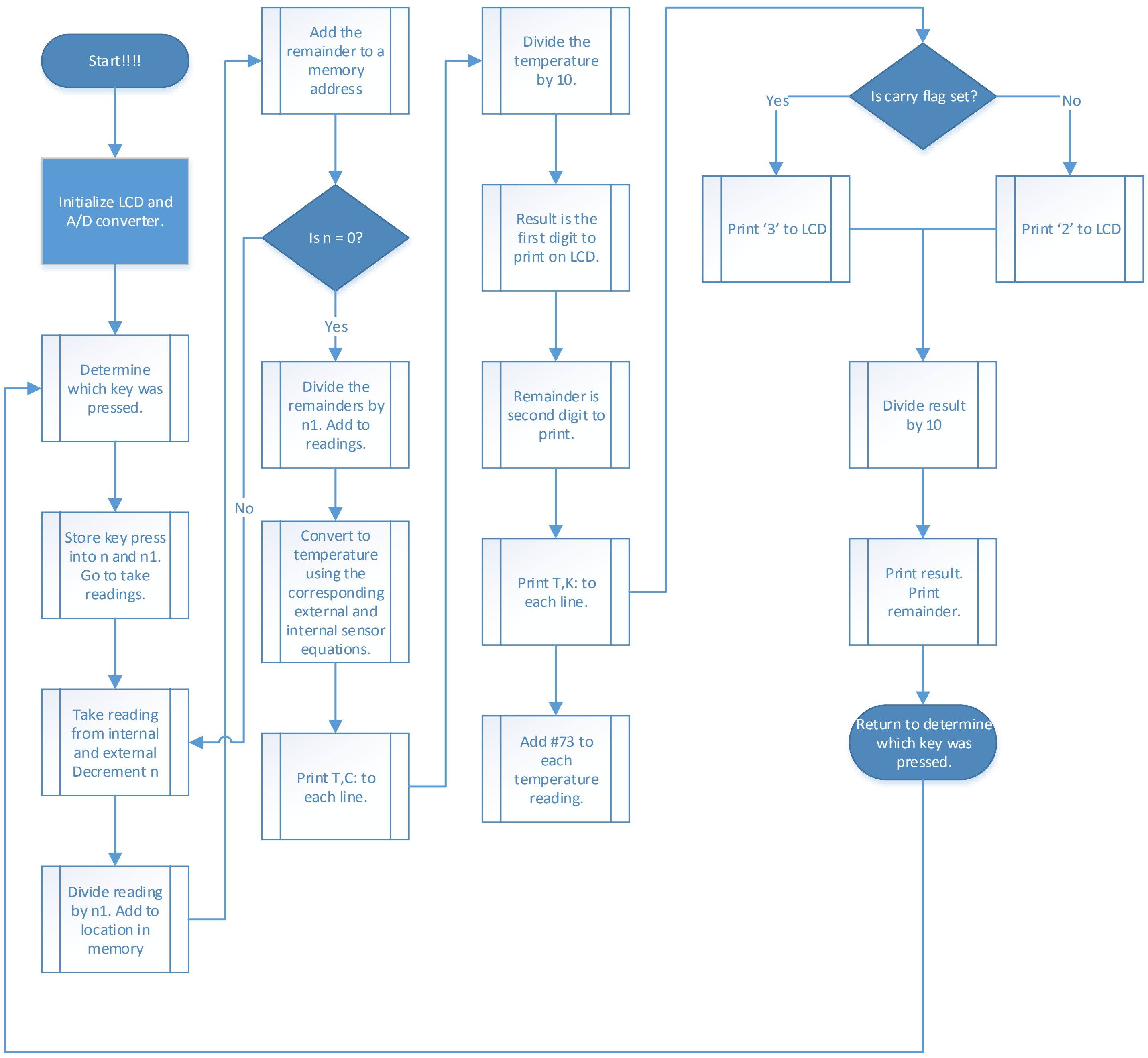
Fourth, the only difference between the display subroutine and the displayx subroutine is the display subroutine jumps to the new address subroutine in the BF_check file to start the cursor on the second line of the display as well as clears the display by jumping to the clear display subroutine in the BF_check file whereas the displayx subroutine just returns the cursor to the first line. Refer to the BF_check flowchart to see how the previously mentioned subroutines in BF_check work. Next, the display subroutines write the characters “T”, “C”, “:”, onto the LCD display as described in the next section. Then, the Hex value of the temperature in one memory address is converted into binary coded decimal (BCD). Next, we jump to subroutine DAA_decode which by comparing the binary number in memory branches to the appropriate subroutine that then writes the correct number to the LCD display as described in the next section. The, temperature sensors by default read the temperature in Celsius. Now, the Celsius temperature read by the sensors is finally written to the LCD display. Then, we just print out a space on the LCD for proper formatting. Additionally, the display subroutines write the characters “T”, “K”, “:”, onto the LCD display as described in the next section. Then, the temperature stored in memory is converted to kelvin by adding 73 to the Celsius temperature and checking to see if there is an overflow or not if there is an overflow branch to the write_3 subroutine where a 3 is written to the display before 100 is subtracted from the temperature and the Hex value stored in memory is written to the display the same way as described above for the Celsius temperature. If there is no overflow branch to the write_2 subroutine where a 2 is written to the display before the Hex value stored in memory is written to the display the same way as described above for the Celsius temperature. Finally, the LCD display now shows the temperature in both Celsius and Kelvin for both the internal and external temperature sensors. By pushing the star button on the keypad the display is cleared and “Enter n:’ is displayed on the LCD again.

Solution – Write to LCD and LED’s

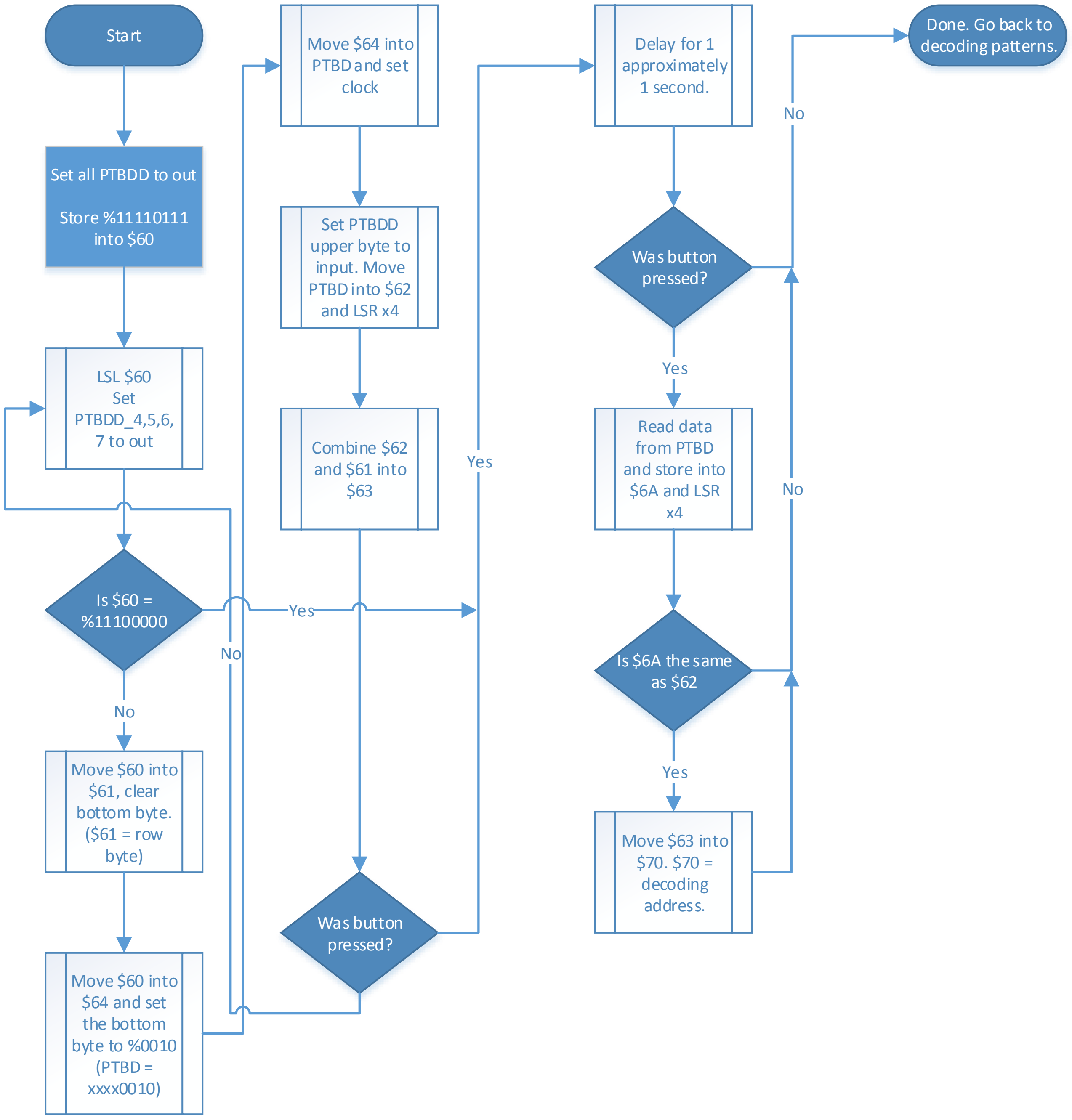
First, set bit 3 of PTBD for a rising edge of the clock. Then, reset the button press to zero by moving 0 into the correct address. Also, set PTADD to an output and set bit 1 of PTAD and clear bit 0 of PTAD. Then, move the correct value into PTBD (upper four bits) to select the column then bring the clock back up by setting bit 3 of PTBD. Next, move the correct value into PTBD (lower four bits) to select the row then bring the clock back up by setting bit 3 of PTBD. By selecting the correct row and column characters 0,1,2,3,4,5,6,7,8,9 are written to the LCD based on the two binary coded decimal values stored in memory which is the temperature of the sensors in Kelvin and Celsius. The characters “T”, “C”, “:”, “K”, “,” are also written to the LCD display with the same method as above, but not using BCD values stored in memory. Lastly, jump to the busy flag check subroutine and return to main.

Final Comments:

While performing this lab, many problems and errors in the code were encountered. Some took multiple hours to troubleshoot and debug. The most bothersome and frustrating was figuring out and understanding how to design the solution to convert the readings from the internal temperature sensor to a Celsius temperature. Converting that transfer function for the internal temperature sensor into assembly that would manipulate the average reading count into a Celsius temperature was very frustrating and challenging. Also since we are in 8-bit mode the internal temperature sensor is very inaccurate and only increases or decreases by 8 degrees Celsius. Whereas the external temperature sensor is very accurate and changes by 1 degree Celsius up or down

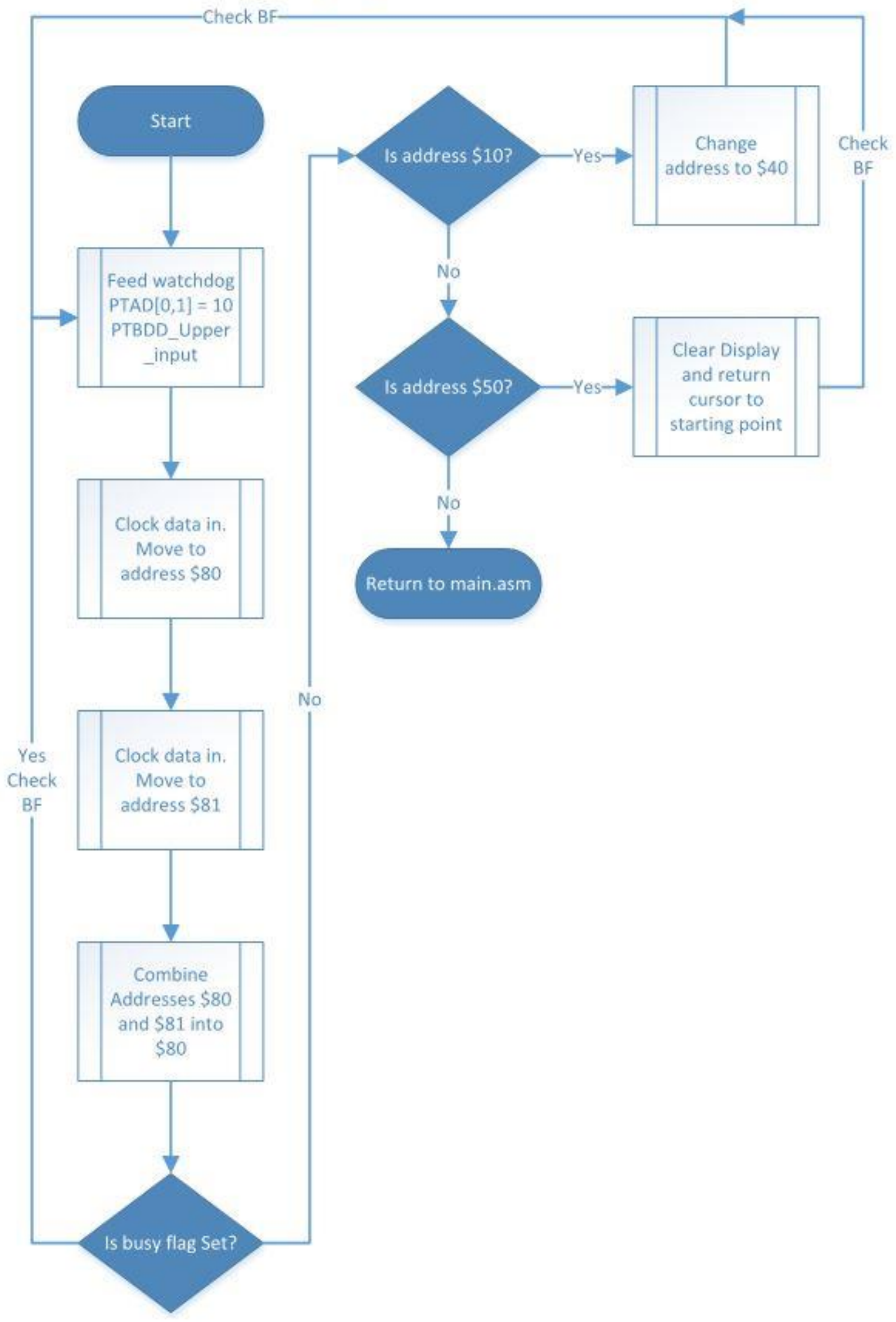

Start!!!!
Initialize LCD and A/D converter.
Determine which key was pressed.
Store key press into n and n1. Go to take readings.
Take reading from internal and external Decrement n
Divide reading by n1. Add to location in memory
Add the remainder to a memory address
Is n = 0?
No
Yes
Divide the remainders by n1. Add to readings.
Convert to temperature using the corresponding external and internal sensor equations.
Print T,C: to each line.
Divide the temperature by 10.
Result is the first digit to print on LCD.
Remainder is second digit to print.
Print T,K: to each line.
Add #73 to each temperature reading.
Is carry flag set?
Yes
No
Print '3' to LCD
Print '2' to LCD
Divide result by 10
Print result. Print remainder.
Return to determine which key was pressed.

Reading from Keypad Flowchart



Start
Set all PTBDD to out
Store %11110111 into $60
LSL $60 Set PTBDD_4,5,6, 7 to out
Is $60 = %11100000
No
Yes
No
Move $60 into $61, clear bottom byte. ($61 = row byte)
Move $60 into $64 and set the bottom byte to %0010 (PTBD = xxxx0010)
Move $64 into PTBD and set clock
Set PTBDD upper byte to input. Move PTBD into $62 and LSR x4
Combine $62 and $61 into $63
Was button pressed?
Yes
Delay for 1 approximately 1 second.
Was button pressed?
Yes
Read data from PTBD and store into $6A and LSR x4
Is $6A the same as $62
Yes
Move $63 into $70. $70 = decoding address.
No
No
Done. Go back to decoding patterns.

Busy Flag Check Flowchart



Check BF
Start
Feed watchdog PTAD[0,1] = 10 PTBDD_Upper _input
Clock data in. Move to address $80
Clock data in. Move to address $81
Combine Addresses $80 and $81 into $80
Is busy flag Set?
Yes Check BF
No
Is address $10?
Yes
Change address to $40
Check BF
No
Is address $50?
Yes
Clear Display and return cursor to starting point
No
Return to main.asm