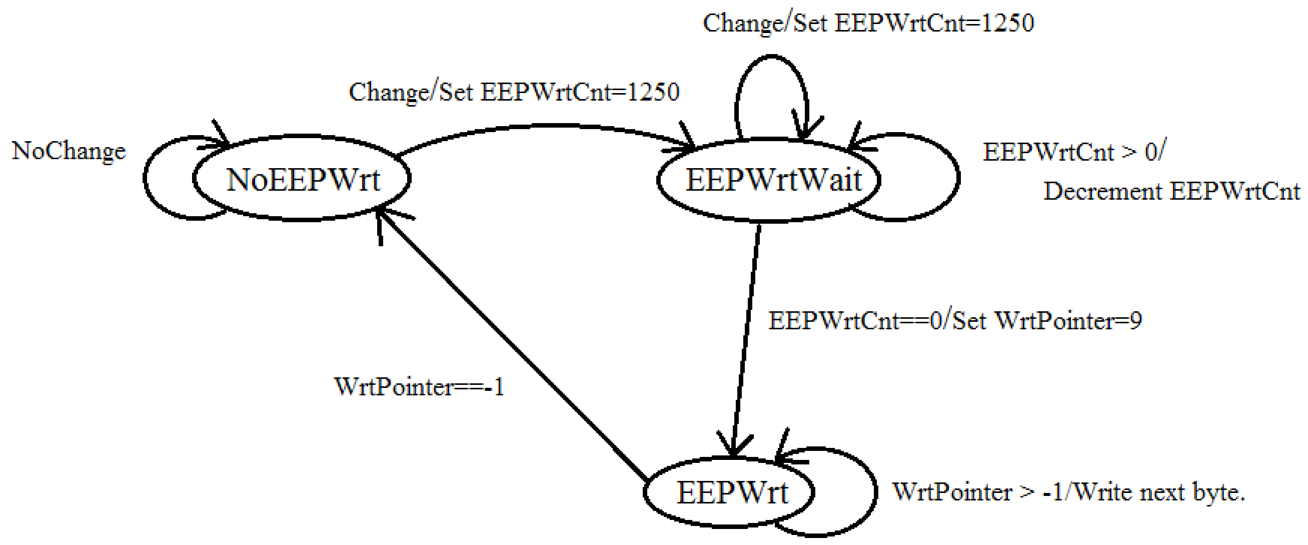
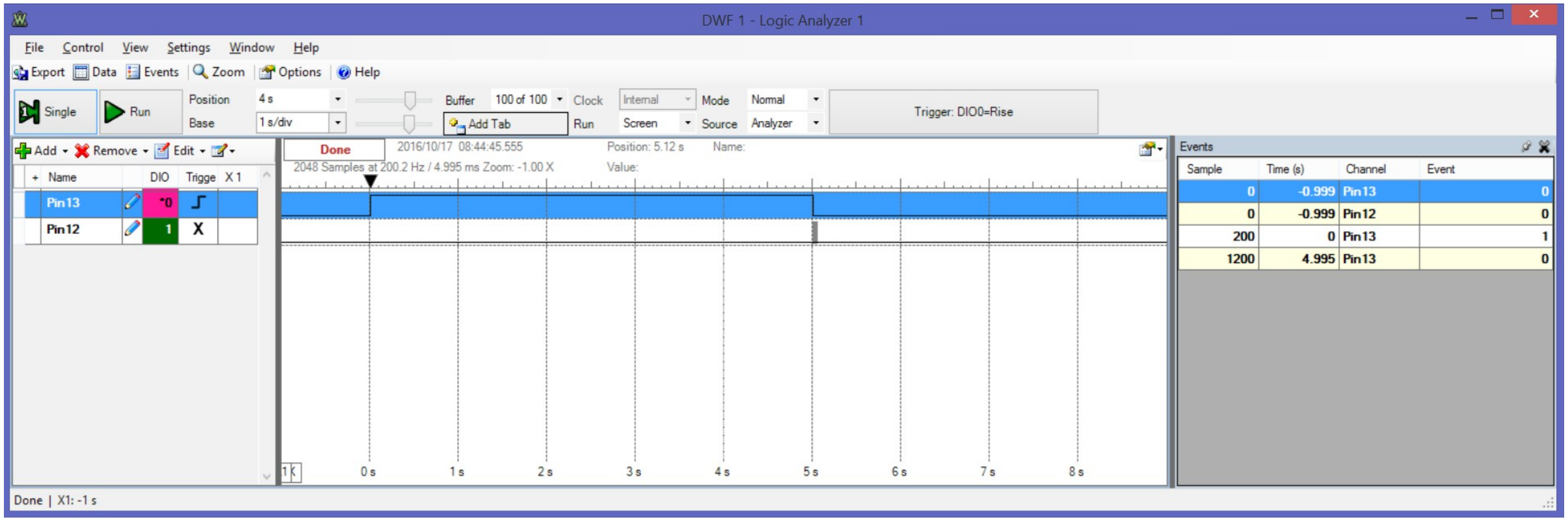
This system will maintain two floating point numbers, which represent the high and low temperature thresholds for a thermostat. These two numbers will be saved to the EEPROM, on the Arduino. The system will wait for the two parameters to change, and then be stable for 5 seconds. Then it will begin to write the state of the system and the two temperatures to EEPROM.

The system will use a hardware time that fires every 4 milliseconds, which is just greater than the 3.3 milliseconds needed for an EEPROM write. Then at each ISR of the timer, the following state machine will be used. It should be noted that the 5 second interval that we want to wait, is counted off as 1250 interrupts (4 milliseconds * 1250 = 5000 milliseconds = 5 seconds).


Change/Set EEPWrtCnt=1250
Change/Set EEPWrtCnt=1250
NoChange
NoEEPWrt
EEPWrtWait
EEPWrtCnt > 0/
Decrement EEPWrtCnt
EEPWrtCnt==0/Set WrtPointer=9
WrtPointer==-1
EEPWrt
WrtPointer > -1/Write next byte.

The timing results are shown below, and code is attached.



Position 4 s
Base 1 s/div
Buffer 100 of 100
Clock Internal
Mode Normal
Run Screen
Source Analyzer
Trigger: DIO0=Rise

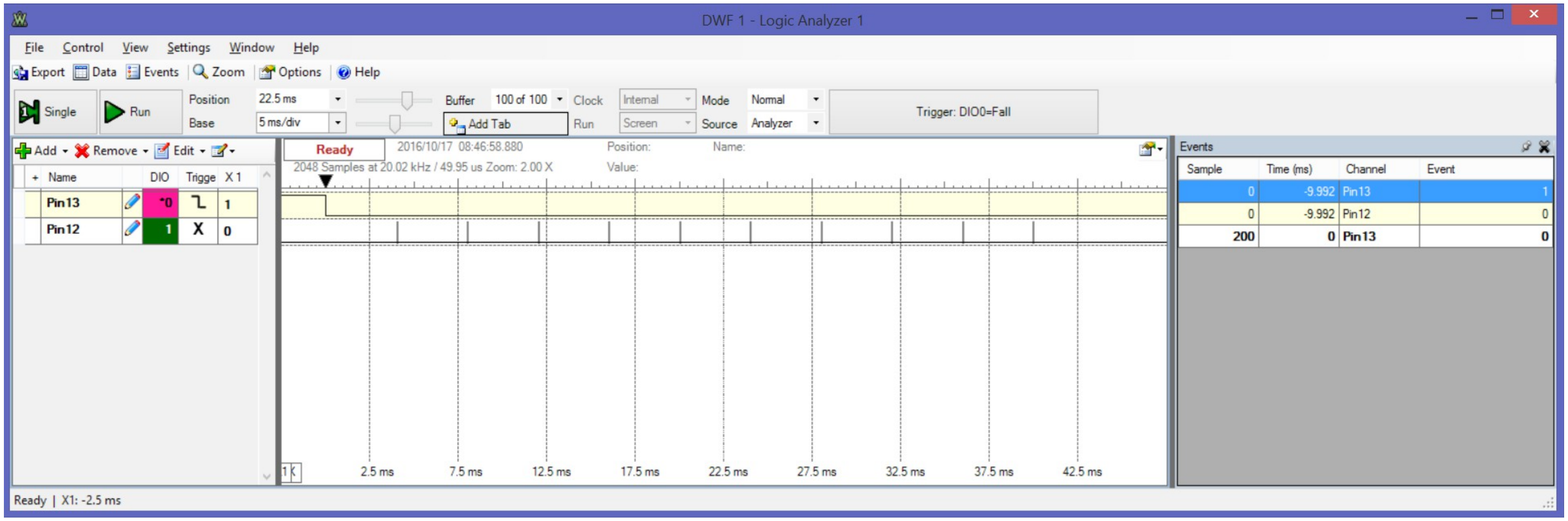
Done 2016/10/17 08:44:45.555
2048 Samples at 200.2 Hz / 4.995 ms Zoom: -1.00 X
Position: 5.12 s Name:
Value:

+	Name	DIO	Trigge	X 1
	Pin13	0	⎍	
	Pin12	1	X	

Events

Sample	Time (s)	Channel	Event
0	-0.999	Pin13	0
0	-0.999	Pin12	0
200	0	Pin13	1
1200	4.995	Pin13	0

Done | X1: -1 s



Position 22.5 ms
Base 5 ms/div
Buffer 100 of 100
Clock Internal
Run Screen
Mode Normal
Source Analyzer
Trigger: DIO0=Fall

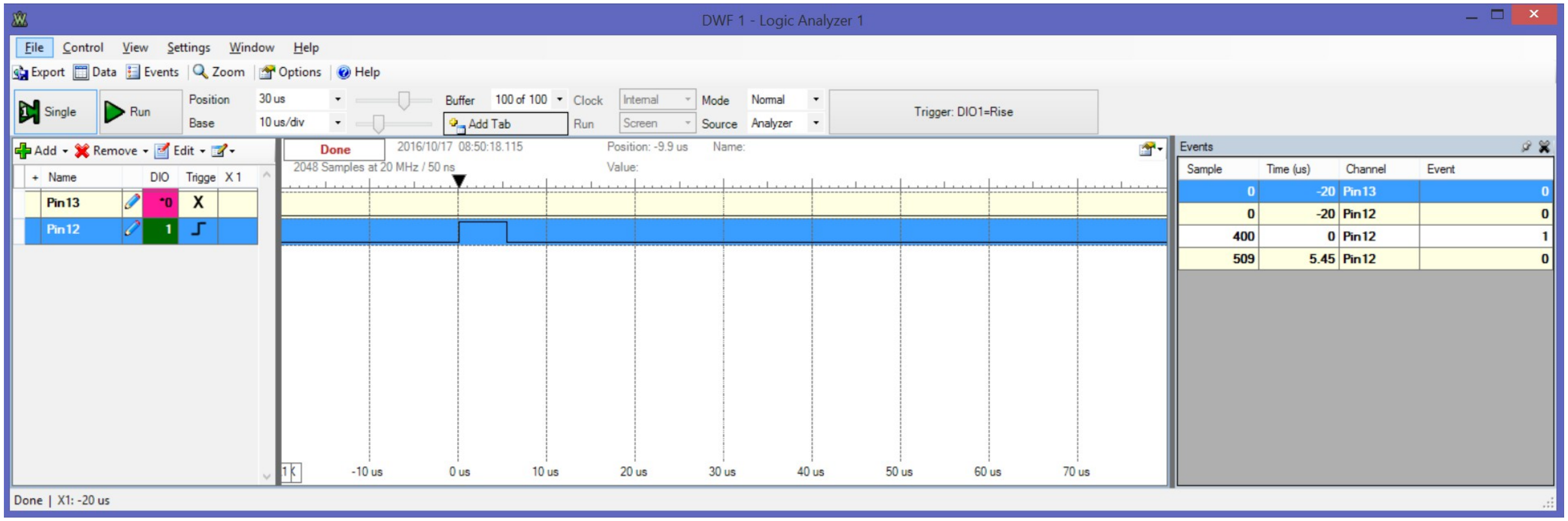
Ready 2016/10/17 08:46:58.880
2048 Samples at 20.02 kHz / 49.95 us Zoom: 2.00 X
Position:
Value:
Name:

Name	DIO	Trigge	X 1
Pin13	⁺0	⎿	1
Pin12	1	X	0

Events

Sample	Time (ms)	Channel	Event
0	-9.992	Pin13	1
0	-9.992	Pin12	0
200	0	Pin13	0

Ready | X1: -2.5 ms



DWF 1 - Logic Analyzer 1

Position: 30 us
Base: 10 us/div
Buffer: 100 of 100
Clock: Internal
Run: Screen
Mode: Normal
Source: Analyzer
Trigger: DIO1=Rise

Done 2016/10/17 08:50:18.115
2048 Samples at 20 MHz / 50 ns
Position: -9.9 us
Name:
Value:

Name	DIO	Trigge
Pin13	˜0	X
Pin12	1	⌐

Events

Sample	Time (us)	Channel	Event
0	-20	Pin13	0
0	-20	Pin12	0
400	0	Pin12	1
509	5.45	Pin12	0

Done | X1: -20 us

Appendix A: EEPROM Write code

```
#include <MsTimer2.h>

// includes for systems on Arduino
#include <EEPROM.h>
#include <LiquidCrystal.h>
LiquidCrystal Lcd(11, 9, 5, 6, 7, 8);

// Variable that are to be updated.
// Declared as globals.
union flt_byte { float F; unsigned char B[4]; } LowTemp, HighTemp;
int WrtPointer = -1; // indicates portion to be written, -1 being no writes.

// State Variable and its setup.
enum EEPWrtStates { NoEEPWrt, EEPWrtWait, EEPWrt };
volatile EEPWrtStates WrtState = NoEEPWrt;
#define EEP_WRITE_INTERVAL 1250
volatile int EEPWrtCount = EEP_WRITE_INTERVAL;

// Called if a change was made in the paramters.
void ResetWrt()
{
	WrtState = EEPWrtWait; // Move to wait state.
	EEPWrtCount = EEP_WRITE_INTERVAL;
	PORTB |= 0x20; // set pin 13 high indicating
	// a wait to write to EEPROM
} // end of ResetWrt

// Four millisecond ISR and State machine controller
void WrtNextStep()
{
	switch (WrtState)
	{
	case NoEEPWrt: // Nothing happening.
		break;
	case EEPWrtWait: // Waiting for 5 seconds to write.
		if (EEPWrtCount) // check if time out has occurred.
		{
			EEPWrtCount--;   // Count down to time out
		}
		else // Time out has occurred
		{
			// So move to write state.
			WrtState = EEPWrt; // Go back to Write.
			WrtPointer = 9;
			PORTB &= ~0x20; // Set 13 low, end of wait.
		} // end of write timer.
		break;
	case EEPWrt: // writing.
		WriteOutState(); // Write next data to EEPROM.
		if (WrtPointer == -1) // If we are done
			WrtState = NoEEPWrt; // Writing.
		break;
	}// end of switch
} // end of WrtNextStep
```

```
#define lowTempUpdate 0
#define highTempUpdate 1
unsigned char DisplayState = lowTempUpdate;

// Support functions for Display state and LCD
void ReadInState()
{
      // Read in DisplayState
      DisplayState = EEPROM[8];
      if (DisplayState >= 2) // This indicates that the
      {                        // EEPROM has NOT been written to.
            DisplayState = lowTempUpdate; // So load parameters with defaults.
            LowTemp.F = 250.0;
            HighTemp.F = 260.0;
      }
      else // if EEPROM has been written
      {           // Read in other parameters.
            LowTemp.B[0] = EEPROM[0];
            LowTemp.B[1] = EEPROM[1];
            LowTemp.B[2] = EEPROM[2];
            LowTemp.B[3] = EEPROM[3];
            HighTemp.B[0] = EEPROM[4];
            HighTemp.B[1] = EEPROM[5];
            HighTemp.B[2] = EEPROM[6];
            HighTemp.B[3] = EEPROM[7];
      }
} // end of ReadInState

void WriteOutState()
{
      if (-1 < WrtPointer) // if there is data to write,
            WrtPointer--; // move to next sample
      PORTB |= 0x10; // Pin 12 to flag write to EEPROM
      // Switch between the different values to be written out.
      switch (WrtPointer)
      {
      case 0:
            EEPROM.update(0, LowTemp.B[0]);
            break;
      case 1:
            EEPROM.update(1, LowTemp.B[1]);
            break;
      case 2:
            EEPROM.update(2, LowTemp.B[2]);
            break;
      case 3:
            EEPROM.update(3, LowTemp.B[3]);
            break;
      case 4:
            EEPROM.update(4, HighTemp.B[0]);
            break;
      case 5:
            EEPROM.update(5, HighTemp.B[1]);
            break;
      case 6:
            EEPROM.update(6, HighTemp.B[2]);
            break;
      case 7:
            EEPROM.update(7, HighTemp.B[3]);
            break;
      case 8:
            EEPROM.update(8, DisplayState);
      } // end of switch to write multiple values
      PORTB &= ~0x10; // Clear flag of write.
}// end of WriteOutState
```

```
// Display variables on lcd
void DisplayLcd(void)
{
	Lcd.clear(); // Start out with a clean display
	Lcd.home();
	Lcd.print(' '); // First a blank for the cursor to land on.
	Lcd.print(HighTemp.F); // Then write each word in its place.
	Lcd.setCursor(0, 1);
	Lcd.print(' ');
	Lcd.print(LowTemp.F);
	// Switch to set cursor to the correct place based on state.
	switch (DisplayState)
	{
	case lowTempUpdate:
		Lcd.setCursor(0, 1);
		break;
	case highTempUpdate:
		Lcd.setCursor(0, 0);
		break;
	} // Turn on cursor.
	Lcd.cursor();
	Lcd.blink();
} // End of DisplayOnLcd

// Updates display state and variables based on incoming characters.
void DisplayUpdate(char ch)
{
	switch (DisplayState)
	{
	case lowTempUpdate:
		if (ch == 'N')
			DisplayState = highTempUpdate;
		else if (ch == 'U')
			LowTemp.F++;
		else if (ch == 'D')
			LowTemp.F--;
		break;
	case highTempUpdate:
		if (ch == 'N')
			DisplayState = lowTempUpdate;
		else if (ch == 'U')
			HighTemp.F++;
		else if (ch == 'D')
			HighTemp.F--;
		break;
	} // end of DisplayState switch

	// Check to see if we need to update display.
	if (ch == 'N' ||
		ch == 'U' ||
		ch == 'D')
	{
		DisplayLcd(); // Update display, since a change occurred.
		ResetWrt(); // Reset wait for EEPROM write.
	} // End of
} // end of DisplayUpdate()

// put your setup code here, to run once:
void setup()
{
	// Setup the display
	Lcd.begin(16, 2);
	Lcd.clear();
	Lcd.home();
	ReadInState();
	DisplayLcd();
	// Indicators of write process.
	pinMode(13, OUTPUT);
	pinMode(12, OUTPUT);
	digitalWrite(13, LOW);
	digitalWrite(12, LOW);

	// Have timer run write code every 4 milliseconds.
	MsTimer2::set(4, WrtNextStep);
	MsTimer2::start();

	// Setup Serial port
	Serial.begin(9600);
} // End of setup


// put your main code here, to run repeatedly:
void loop()
{
	// Check incoming serial data.
	if (Serial.available())
	{
		char ch = Serial.read(); // Read in ch.
		// convert to upper and process.
		DisplayUpdate(ch & 0xDF);

	}// end of Serial input.

} // end of loop
```