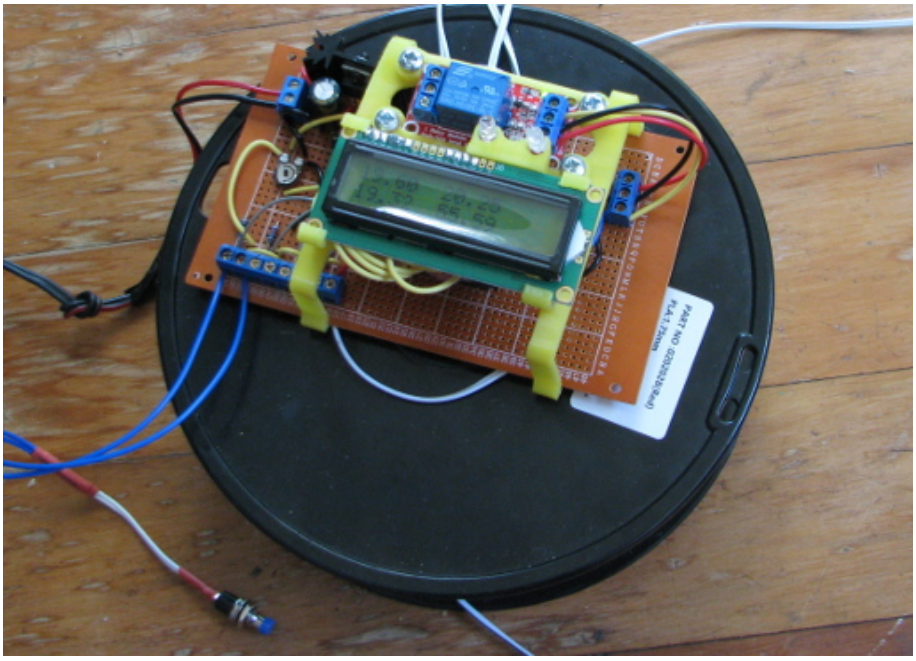
Arduino Solar Water Heater Controller

An Emergency Project

So we got back from our Christmas/New Year break to find our water cylinder temperature was somewhat colder than we would have expected. Normally when we go away for a period, the solar water heater gets the water up to a very high temperature even during times during which the weather is a bit variable. Although this summer has been pretty rubbishy the low temperature immediately had me suspicious that something was not right. This was soon confirmed when night fell and the solar water heater circulation pump continued to operate.

As it turned out a power-spike or something had gone through the house while we were away and scrambled the solar water heater controller's brains. It now thought that even 25°C days were too cold and needed the pump turning over for frost protection. So I quickly put a timer in between the controller and the wall socket and set it so that it would only run during daylight hours.

The existing controller is a Solastat controller (http://www.senztek.com). I can't say it was the best controller, and had some annoyingly inappropriate set points for the onset of frost protection. This had frost protection beginning when the ambient conditions were 8°C. Wellington often gets cold days around 8°C, but it very rarely gets below freezing. So this wasted a lot of hot water. I tried to get information from Senztek about adjusting the settings (which can be done), but they would not release the method for unlocking the device and basically were very unhelpful. So, for a few years my to-do list featured a project to replace the controller with something that could be set for the local conditions. This failure of the Solastat was the perfect opportunity to put it into action and design, build, program, install, and commission a much smarter Solar Water Heater Controller in which the set points could be optimised for the local conditions.



The New Controller Being Tested

The Project

Designing the electronics and logic was a relatively smooth and pain-free process. Installing it and dealing with real-world stuff was not. Below you will find links to articles where I describe the idealised system that would run very well if you have good reliable temperature sensors and you have the ability to put the sensors exactly where you want them. I also describe the actual system as it was installed and the adjustments and compromises made to get it to work nicely.

You will find the logic flow diagrams, arduino programs, circuit diagrams, and perf-board layouts for each system.

The Idealised System

In the perfect world of design and programming, sensors are shiny and accurate, noise in the system does not exist, and the system always has the perfect spot to install your sensors.

Here is the article on the controller for the perfect world. **Perfect World Solar Water Heater Controller.**

The Actual Installed System

In the real world, things are a bit more gritty. Your sensors do not agree perfectly, electric motors, relays, and other stuff throw messy electrical stuff everywhere, the plumbers have created a unholy mass of brass fittings on the solar water heater that make it hard place the sensors where they will be capturing the best data. The hot water cylinder does not have a nice convenient sensor pocket, and broken sensor wires happen too easily. Nuff said?

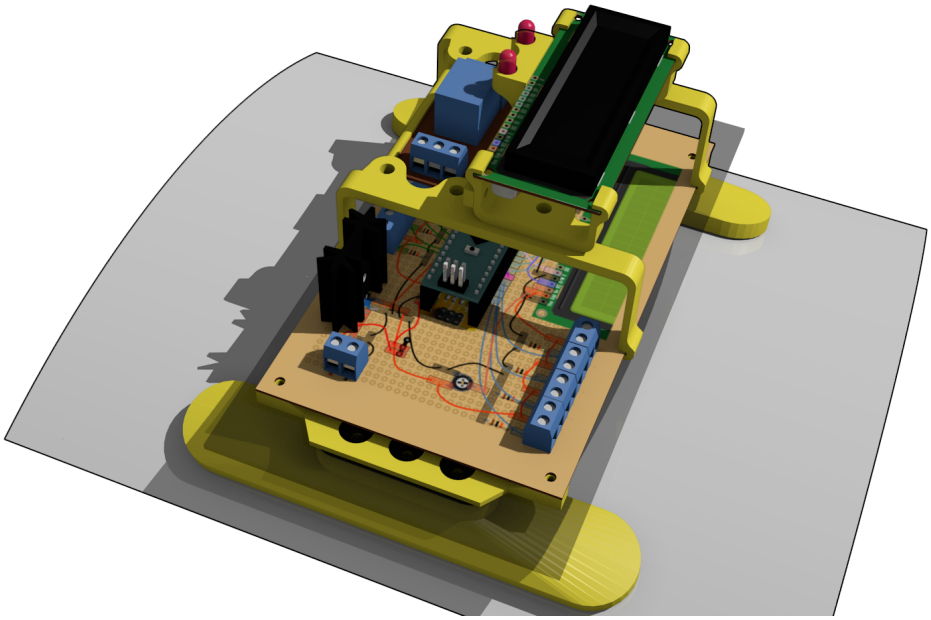
Here is the article on the controller for the real world. **Real World Solar Water Heater Controller.**

The Casing and Internal Supports

The circuit board was built up on a 9cm x 15cm perf-board (30x48 holes) and the LCD screen for displaying the data was connected to the main board through a number of wires which meant it could be positioned relatively freely. This meant the whole lot could be conveniently installed in a UB2 ABS Jiffy Box as supplied by Jaycar. These are a nice sturdy box with an internal volume of 172mm x 104mm x 57mm.

Because I wanted to be able to access the controller Arduino relatively easily to load new variations of the program while optimising the set points, the case needed to the lid to lift off without having the LCD screen or LEDs attached to it. A 3Dprinted frame was designed to carry the LCD screen, LEDs, and relay so they would remain in position when the cover was removed. Two feet were also designed to match the curve of the hot water cylinder surface and hold a series of magnets. In theory the magnets would have held the controller to the cylinder but in reality they weren't quite strong enough and duct tape was employed to help hold it in place.

Chances are though, if you are looking at this project, you are probably going to have a different size hot water cylinder, have made different choices about the relay module to use, different size box, and a whole host of other things that mean the structure you will require will be different from what I have used. For the sake of completion though, here are the relevant files for designing and 3D printing the support arrangement.



Blender File. The Blender file was designed in blender 2.58. Some of the components used have come from the collection of components I have on this website. www.techmonkeybusiness.com/component-model-library.html

- You can download the .blend files here. SWH_Controller_Frame-blend.zip
- The STL files for 3Dprinting can be found here: SWH_Controller_Frame-STLs.zip

Optimising the Device - What to Tweak

Control Set Points

Once the device is up and running there is always room for optimising the set points to the local conditions and the actual performance of the temperature sensors.
As mentioned already, Wellington gets cold but very rarely freezes. This means we can keep the frost protection related set points relatively close to 0°C without risking patches of the panel freezing.
The other factor that needs to be accounted for is the error between the actual temperature you are trying to read and the temperature the sensor sees. In the case of the system here, the solar water heater outlet temperature sensor is installed in a small pocket projecting into the flow near the outlet of the solar water heater. It gets good measurements when the water is flowing but when the pump is off, the heat has to travel through the metal fittings before it gets to the sensor. Because of this, the sensor can be slow to detect the water temperature in the panel reaching a suitable temperature to begin pumping. When the pump starts, this leads to a sudden temperature rise at the sensor when the heated water reaches it.

Observing the temperatures into and out of the solar water heater as well as the temperature difference between the solar water heater outlet and the cylinder intake during steady pumping and when the pump starts will allow the user to make an educated guess about what the appropriate set points might be. My current settings are: 2.5°C between the solar water heater outlet and the intake to the hot water cylinder, while to turn the pump off the temperature differential between these same two sensors must get below 1.5°C. This seems to give a good hot water yield and responds well to less sunny days and the low sun angles of early morning and late afternoon.

The set points are defined by the following variables; `FrostTemp`, `SWHFrostThresh`, `SWHOutHeatedTemp`, `SWHCylTdiff`, and `SWHTdiff`.

Look for the following lines in the control sketches.

```
double FrostTemp = 4.00; //Temperature below which Frost is a risk.
                         //Adjust for local conditions.
double SWHFrostThresh = 5.00; //Temperature at SWH at which pump
                         //will start if running in Frost protection mode.
double SWHOutHeatedTemp = 8.00; //Temperature which will indicate that
                         //hot water is in SWH for frost protection.
double SWHCylTdiff = 2.50; //Temperature difference between cylinder
                         //and solar water heater before pumping starts.
double SWHTdiff = 1.50;  //Min temperature difference between cylinder
                         //and solar water heater before pumping is stopped.
```

`SWHCylTdiff`, and `SWHTdiff` are the two set points that control the operation of the solar water heater pump during normal operations. Both refer to the temperature difference between the solar water heater panel outlet and the sensor location at the hot water cylinder intake.

On my system, I have observed that there is a difference in the readings between the solar water heater inlet and the solar water heater outlet in the early hours of the morning when they should be pretty much the same. Likewise there is a small but consistent difference between what the “Stagnation temperature” sensor is reading and what the nearby home weather station temperature sensor is reading. Unfortunately I was unable to have a time when I could calibrate the sensors together, but the presence of these differences makes me err on the side of caution when it comes to the set points for frost protection. With a bit more testing as autumn and winter approaches I am likely to be able to refine the frost protection set points to reflect actual conditions better.

Sensor Calibration

Before attempting to install the solar water heater controller, the idealised system had four thermistors that were supposedly 10kΩ NTC type with a *Beta Value* of 4100K. Generally it is not so important that the temperature they report is absolutely accurate, but agreement between the sensors makes life much easier. To test this it is worth putting all of your sensors in a glass of water (not touching the side) or on some surface that has an even temperature across it, and seeing what temperatures are reported. When I did this I found that one of the sensors had a permanent offset from the other three sensors. By changing the temperature of the water across a wide range, I was able to generate an expression to adjust the reading from this particular thermistor to a temperature that would match the others.

Carrying out a test like this is relatively simple when you have the circuit built, and reporting to the LCD screen.

Of course, the fact that I used the thermistors already installed on the system in the final version of this system meant that I was not able to achieve such a good match between the different sensors. This is reflected in a greater uncertainty in, and more fiddling of the set points.

Things to watch out for

There are several things I have learned from this system so far;
Check your sensors are reading pretty much the same. As alluded to above, having good agreement between your sensors makes life a whole lot simpler.
The LCD screen I used is very sensitive to electrical noise. Despite all the careful arrangements of capacitors, flyback transistors, opto-isolation, and pumps with their own noise suppression circuits, the LCD screen still spends a bit of its time displaying garbage when the relay opens. Just because the display is haywire does not mean the system has failed. The rest of the system is working perfectly fine. I have not bothered to try out some of the alternative screens such as the i2c version or just a simple four digit LED display. I just ignore the problem and sooner or later the display comes back. Damn thing!
This project deals with 240V. Watch where you're sticking your mitts. Even better make sure it is unplugged when you are fiddling with it.
The pump override switch requires the user to hold the button in for a period of before it will start. This is because of the “debounce” process used to guard against short cycling the pump. It is not a problem, just something for the user to be aware of.

Operation

The weather this “summer” has been really weird. This has been great for testing the solar water heater controller! It has had some really cold days to look at followed by some sinking hot ones, followed by dull days when the temperature in the hot water cylinder is still very high from the previous day's solar harvest, and also a few quite variable days where the cloud comes and goes. The solar water heater controller had responded as expected to all of these conditions and successfully given us good hot water without losing any due to inappropriate pumping when the incoming solar heat was not hot enough to provide any benefit to the water. That is what I call a success.

Perfect World Solar Water Heater Controller

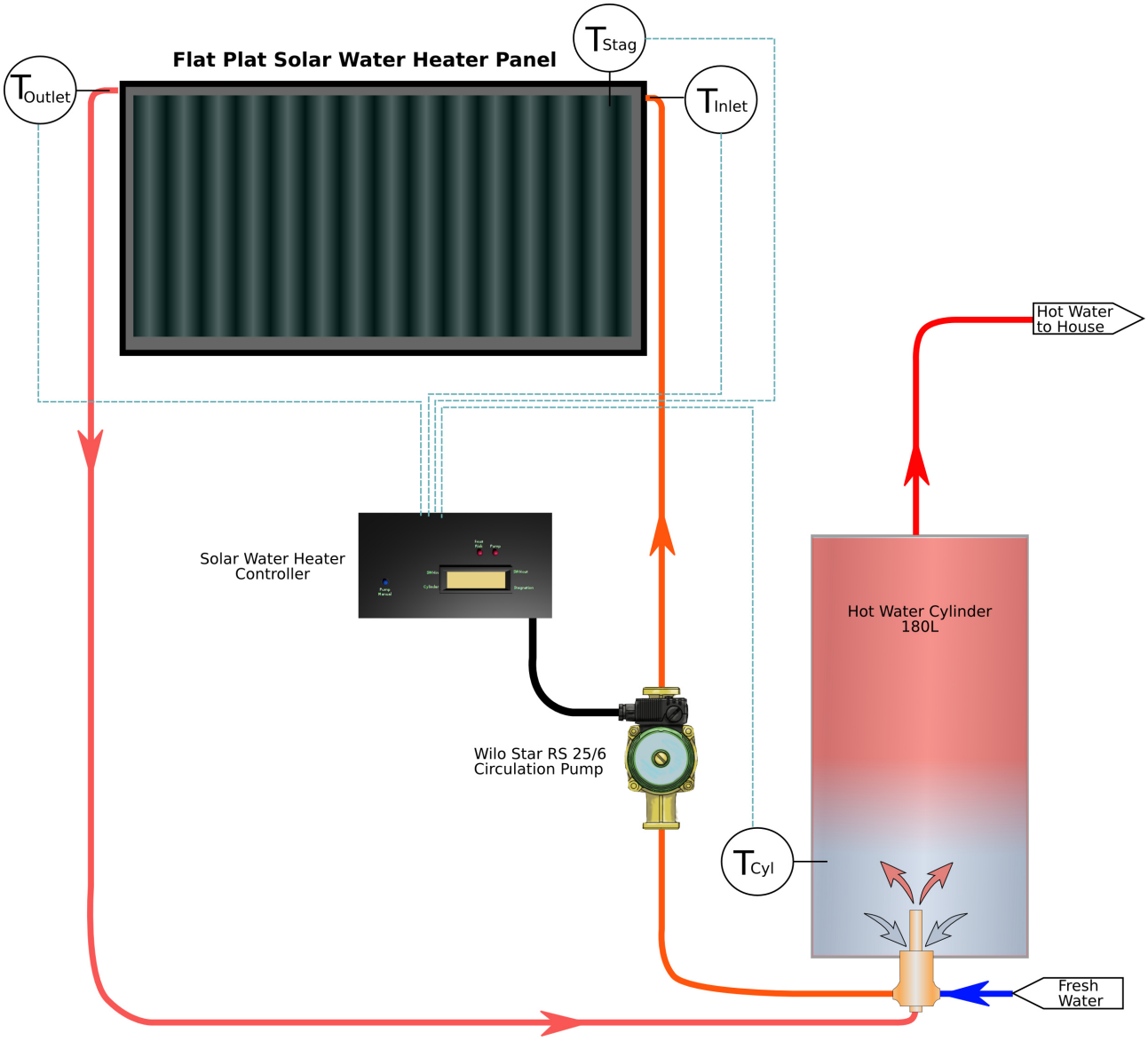
When programming, designing electronic circuits, and conceptualising fabulous machines, it is all done in this clean perfect world where everything is rosy and just so. Well at least it is when I do it. That's probably why I enjoy that part of the project so much. When reality and existing tech gets involved it all becomes somewhat grittier and harder. The real-world system is described here: **Real World Solar Water Heater Controller**

So, here is the perfect world arrangement for the Solar Water Heater Controller. I have included this here because it is the ideal starting point for any other Solar water heater controller system. From here the system can be tweaked and modified to suit the actual arrangements when it comes to dealing with reality and all its imperfections.

Arrangement of the System

The system we have installed on our house is a very simple open loop flat panel solar water heater. It is not the most efficient, but it is robust and well suited to our needs.

For the idealised system using the four temperature sensors the arrangement is as shown.



This arrangement has excellent positioning of the temperature sensors so they are able to detect the actual heat of the water in the panel. The method that would be required to achieve this would be probes that project into the water pipes as they exit and enter the solar water heater. It also makes use of an ideal location on the hot water storage tank where it can detect the actual water temperature close to where the solar water heater draws from and where cold water enters. Finally the biggest assumption is that the arrangement for the Stagnation Temperature sensor is a true approximation of the solar water heater panel performance.

What's Stagnation Temperature?

The *stagnation temperature* for a solar water heater is the temperature the solar water heater will heat to if no water is drawn off. On an overcast day it could be quite close to the ambient temperature. On a sunny day the temperature will be significantly over the ambient temperature because of the absorption of the solar radiation on the black surface of the panel, the insulation on the back side, and the prevention of convective heat loss to the air by having a glass cover.

In this ideal system, the Stagnation Temperature measurement was going to guide the whole system by effectively measuring the quality of the solar radiation. If the stagnation temperature was close to or below the temperature of the cylinder then there was not point in running the pump because there was nothing to be gained. In theory this would have made the system relatively fast to respond to changing cloud cover.

The stagnation temperature sensor also doubled as the frost risk detection temperature sensor. Because the stagnation temperature sensor arrangement has minimal thermal mass compared to the solar water heater panel it would be able to reliably detect when frost conditions were likely without the risk of parts of the panel freezing before the solar water heater inlet or outlet sensors cooled enough to signal the need to go into frost protection mode.

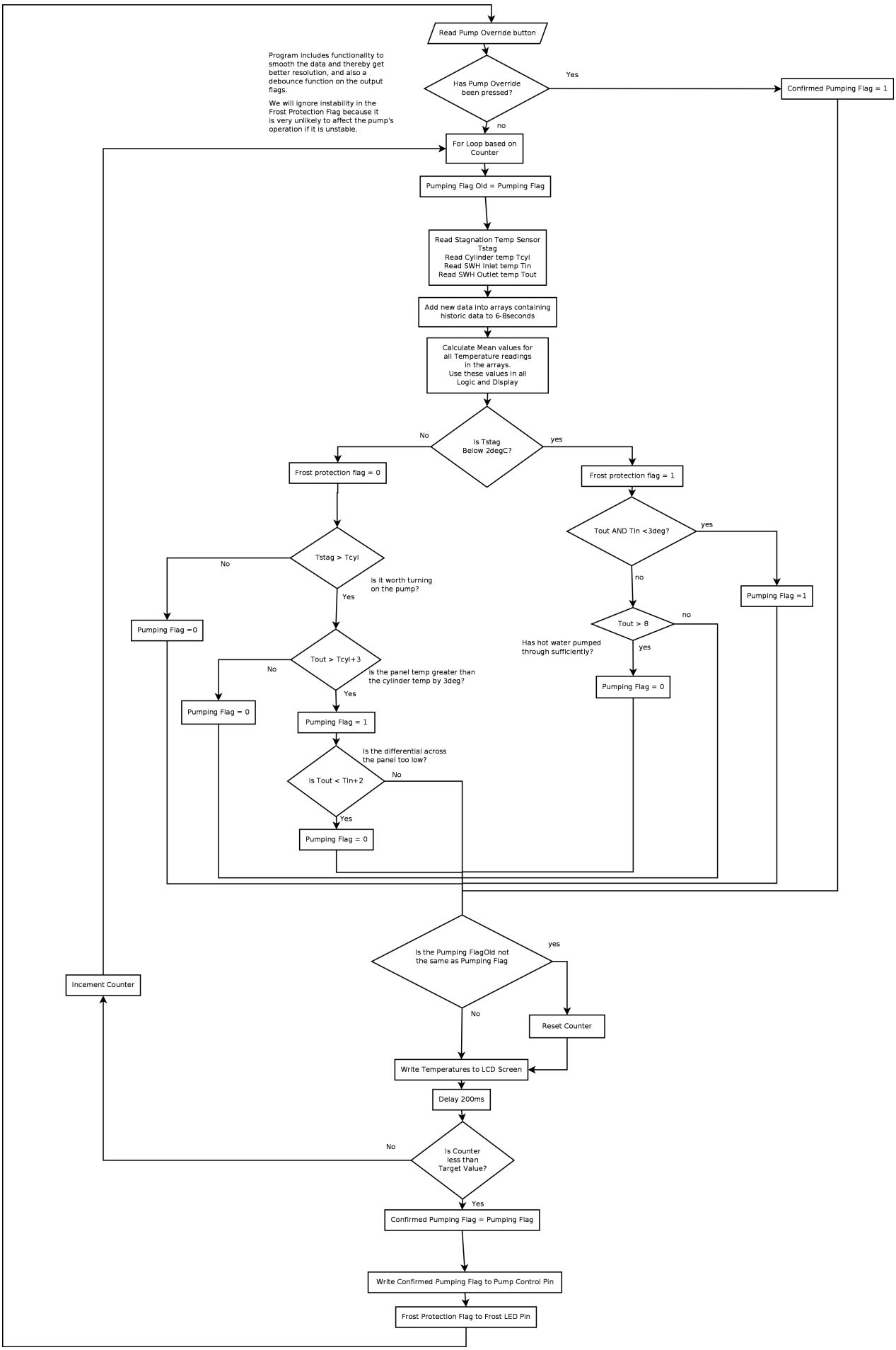
Logic for the Perfect World

The flow diagram presented below is the logic developed to make a responsive system that would react to changes in solar influx quality, temperatures in the hot water storage tank, and on the panels, as well as be able to protect itself from frost damage.

The logic diagram has some hand-waving descriptions of the method used for creating a rolling average to smooth the readings from the sensors.

A debounce process was included to prevent the pump from short cycling when the temperatures were getting close to set points. The period is approximately 8 seconds once a stable signal to start or stop the pump is reached. This is more or less the time it takes for the water to pump through the circuit once.

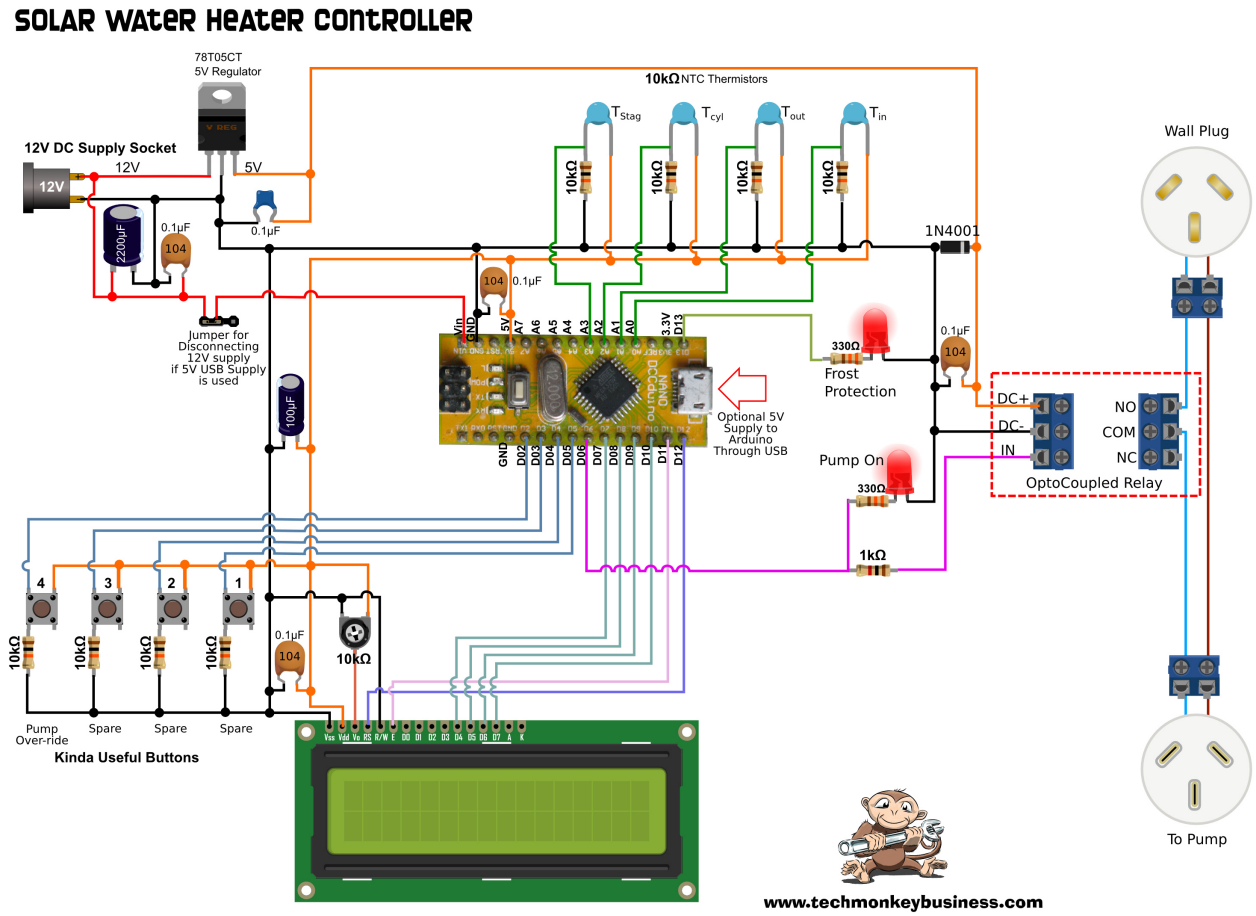
Reduced wear and tear on both the pump and relay should result from using this approach.


Read Pump Override button
Program includes functionality to smooth the data and thereby get better resolution, and also a debounce function on the output flags.
We will ignore instability in the Frost Protection Flag because it is very unlikely to affect the pump's operation if it is unstable.
Has Pump Override been pressed?
Yes
Confirmed Pumping Flag = 1
no
For Loop based on Counter
Pumping Flag Old = Pumping Flag
Read Stagnation Temp Sensor Tstag
Read Cylinder temp Tcyl
Read SWH Inlet temp Tin
Read SWH Outlet temp Tout
Add new data into arrays containing historic data to 6-8seconds
Calculate Mean values for all Temperature readings in the arrays. Use these values in all Logic and Display
Is Tstag Below 2degC?
No
yes
Frost protection flag = 0
Frost protection flag = 1
Tstag > Tcyl
No
Is it worth turning on the pump?
Yes
Pumping Flag =0
Tout AND Tin <3deg?
yes
no
Pumping Flag =1
Tout > 8
no
yes
Has hot water pumped through sufficiently?
Pumping Flag = 0
Tout > Tcyl+3
No
Is the panel temp greater than the cylinder temp by 3deg?
Yes
Pumping Flag = 0
Pumping Flag = 1
Is the differential across the panel too low?
Is Tout < Tin+2
No
Yes
Pumping Flag = 0
Is the Pumping FlagOld not the same as Pumping Flag
yes
No
Reset Counter
Incement Counter
Write Temperatures to LCD Screen
Delay 200ms
Is Counter less than Target Value?
No
Yes
Confirmed Pumping Flag = Pumping Flag
Write Confirmed Pumping Flag to Pump Control Pin
Frost Protection Flag to Frost LED Pin

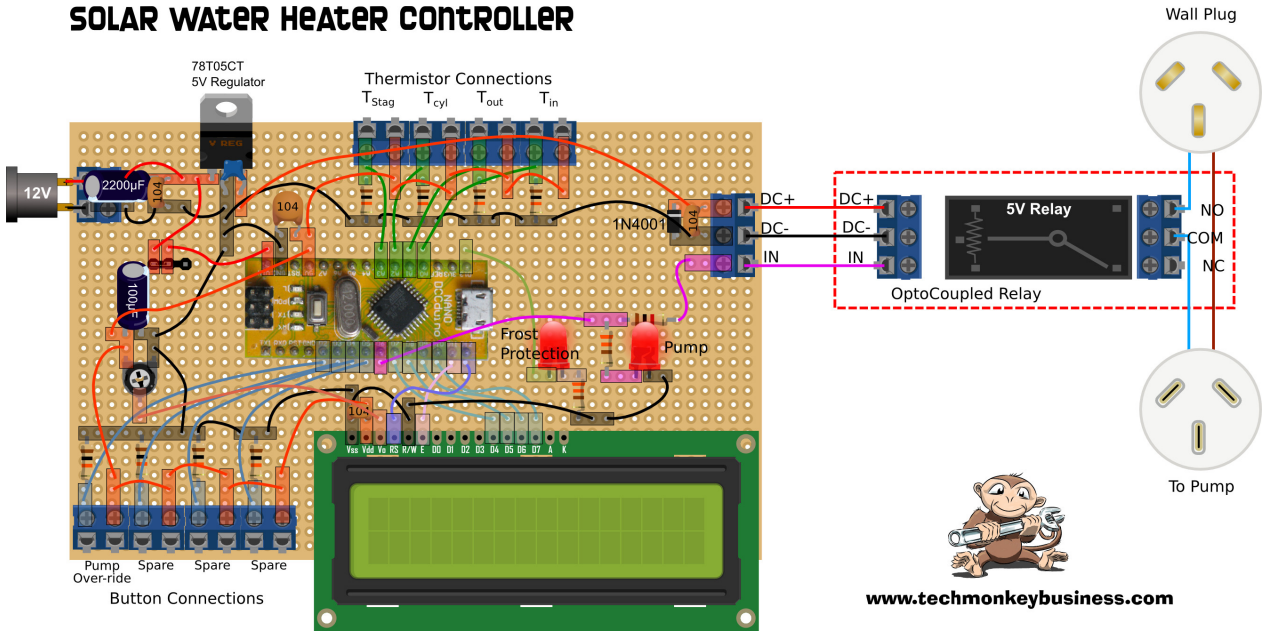
Circuit for the Perfect World System

The circuit is relatively simple and was designed to include power from a separate 5V USB power supply if problems with electrical noise interfering with the operation of the Arduino were encountered. It includes a number of extra connections for other buttons should there be a desire to add more functions.

The circuit shown below is the same for both the ideal and real world systems.

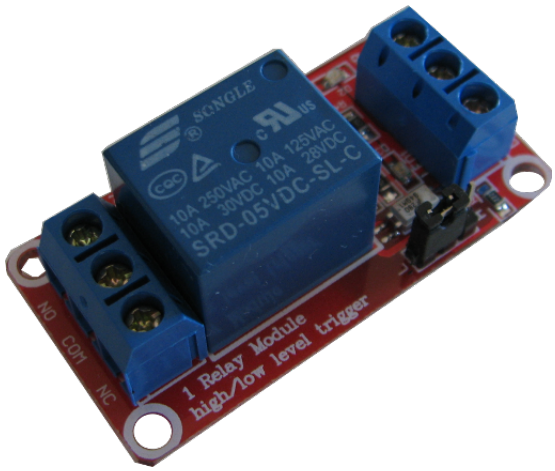


Circuit Diagram



Circuit Board Layout – perf' board.

At the heart of the system is an Arduino Nano. The demands on the Arduino are not large and so any Arduino could be used.



An opto-isolated relay module from Banggood was selected to control the pump. Flyback diodes and various LEDs are included in the package. All it requires is a ground connection, 5v supply, and a signal. Using the jumper on the module the relay can be set to trigger with a high or a low signal. The manufacturer claims the relay can handle 240V 10A AC. I have seen a few comments in various forums that suggest the terminal connections themselves are not capable of carrying that sort of current, but the pump used in our solar water heater system only draws 0.5A anyway, so the module is quite suitable for the task.

The LCD screen is a typical 16x2 type. I have not connected the back light because, why would I want to illuminate the interior of my hot water cupboard. I don't think the dust mites are into reading at night.

The other components of significance are the thermistors and the 5V regulator.

Here is a list of the components used.

- Arduino Nano
- LCD display – 16x2. This is not the i2c version – although I would suggest the 12c version might be better.
- Four NTC 10kΩ thermistors with Beta Values of 4100K.
- A button
- Two LEDs
- MC78T05CT regulator capable of outputting 3A at 5V not that we're using anywhere near that much.
- A 2200µF electrolytic capacitor.
- A 100µF capacitor.
- some 10kΩ resistors.
- A couple of 330Ω resistors.
- A 1kΩ resistor.
- A 1N4001 diode acting as a flyback diode for the relay coil, but probably completely superfluous with the flyback diode built into the relay module already.
- A handful of 0.1µF capacitors liberally sprinkled throughout the circuit like “fairydust” as suggested by Adafruit.

Most of the capacitors and the 1N4001 diode were added late in the piece to try to combat the problem with the LCD screen displaying garbage sometimes when the relay opened. They are not necessary for the circuit to work if the power supplied to the Arduino is clean enough. The only capacitor in the pre-real world scheme was the 2200µF electrolytic capacitor on the regulator inlet side.

Coding for the Perfect World System

Here is the code for the perfect world system. You will notice the key role of the stagnation temperature (Tstag) in this. In the real-world version this was set in such a way that it would enable to the pump to run in most conditions even those that would be really dull and miserable. It's major role was relegated to frost

protection.

Hopefully the comments through the sketch will explain how it works.

You can download the sketch from here: SWHControllerv4.ino

The sketch does not use any libraries that don't come with the standard Arduino IDE. I think it works on older versions of the Arduino IDE too. In short, there is nothing particularly fancy in this coding.

```
/*
 * SWHControllerv4.ino
 * Hamish Trolove 2017 - www.techmonkeybusiness.com
 *
 * This sketch is to run a solar water heater and optimise the solar heat
 * yield.  It is also designed to be easily modified to suit local conditions.
 * A process similar to the Debounce routine in the Arduino Cookbook (pg155)
 * has been used to protect against short cycling of the pump. In addition to
 * this the Array average process (https://www.arduino.cc/en/tutorial/smoothing)
 * has been used on all data to smooth the temperature readings and gain better
 * resolution.
 *
 * There are four thermistors which are located;
 * Solar Water Heater Inlet - Tin
 * Solar Water Heater Outlet - Tout
 * Tank Skin on the lower part - Tcyl
 * and Tstag on a black plate in a pyrex tube to measure stagnation
 * temperatures and frost conditions.
 *
 * The temperatures will control the operation of a pump through an optically isolated relay.
 *
 * Each thermistor reading is displayed on the 16x2 LCD screen.
 *
 * Four buttons are connected.  At this stage only one is used and that is an override to turn
 * on the pump.
 *
 * Circuit is described on www.techmonkeybusiness.com.
 *
 * Connections are:
 *
 * 10K B=4100 NTC Thermistors with 10K resistors in voltage dividers on A0, A1, A2, and A3
 * Push buttons with pull up resistors on D02, D03, D04, and D05
 * LED to indicate Frost Protection mode on D13
 * Pump relay control line on D06 (also connected to an LED)
 * LCD connections:
 * D12 to LCD RS pin
 * D11 to LCD E pin
 * D10 to LCD D7 pin
 * D09 to LCD D6 pin
 * D08 to LCD D5 pin
 * D07 to LCD D4 pin
 *
 *The LCD panel is not lit.
 */

#include <LiquidCrystal.h>
#include <math.h>  // We need this to calculate temperature values from thermistors.
const int ThermPinA = A0;  // The Tin thermistor on Analogue Pin 0
const int ThermPinB = A1;  // The Tout thermistor on Analogue Pin 1
const int ThermPinC = A2;  // The Tcyl thermistor on Analogue Pin 2
const int ThermPinD = A3;  // The Tstag thermistor on Analogue Pin 3
```

```
const int ButtonA = 02; // Some button pins defined (Pump Override)
const int ButtonB = 03;
const int ButtonC = 04;
const int ButtonD = 05;

const int LEDFrost = 13; // Front protection mode indicator LED
const int LEDPump = 6; // Pump running indicator LED.

boolean PumpFlag = false; //This is the flag to control the pump.  All
                          //control logic acts on this flag.
boolean FrostRisk = false; //Frost risk flag for driving LED.
boolean PumpFlagOld = false; //Flag to hold previous PumpFlag State
boolean ConfirmPumpRun = false; //Once all debouncing has occurred this is
                                //the flag to run the pump.

int CycleCount = 0;  //A counter for the number of times the loop has run for
                     //while awaiting stable readings.
int ThermValue = 0; // Somewhere to stick the raw Analogue Pin Value.
double Tin = 0.00; // Calculated Temperature Values after smoothing
double Tout = 0.00;
double Tcyl = 0.00;
double Tstag = 0.00;

//Arrays to hold historic data
const int numReadings = 40;

double TinArray[numReadings];      // the readings from the Temperature sensors
double ToutArray[numReadings];
double TcylArray[numReadings];
double TstagArray[numReadings];

double Tintotal = 800;   // the running totals for the temperature readings
double Touttotal = 800;  //Starting value is 20 x 40 ie starting value
double Tcyltotal = 800;  //in arrays.  If the arrays had been loaded with
double Tstagtotal = 800; //zeros then these figures would also be zero.

int readIndex = 0;              // the index of the Historic Data Array Pointer

double FrostTemp = 2.00; //Temperature below which Frost is a risk.
                         //Adjust for local conditions.
double SWHFrostThresh = 3.00; //Temperature at SWH at which pump
                              //will start if running in Frost protection mode.
double SWHOutHeatedTemp = 8.00; //Temperature which will indicate that
                                //hot water is in SWH for frost protection.
double SWHCylTdiff = 3.00; //Temperature difference between cylinder
                           //and solar water heater before pumping starts.
double SWHTdiff = 2.00;  //Min temperature difference across the solar
                         //heater before pumping is stopped.

LiquidCrystal lcd( 12, 11, 7, 8, 9, 10 );   //Pins for the LCD Screen




void setup()
{
  lcd.begin(16, 2);  //set up the LCD's number of columns and rows:
  lcd.setCursor(0, 0);
```

```
  lcd.print("Starting");  //Print a message to the LCD
  pinMode(ThermPinA, INPUT);
  pinMode(ThermPinB, INPUT);
  pinMode(ThermPinC, INPUT);
  pinMode(ThermPinD, INPUT);
  pinMode(ButtonA, INPUT);  //This is the pump override button.
  pinMode(ButtonB, INPUT);  //spare
  pinMode(ButtonC, INPUT);  //spare
  pinMode(ButtonD, INPUT);  //spare
  pinMode(LEDFrost, OUTPUT);
  pinMode(LEDPump, OUTPUT);

  //Load the arrays with normal(ish) values to begin with.
  for (int Indexer = 0; Indexer < numReadings; Indexer++)
  {
    TinArray[Indexer] = 20.00;
    ToutArray[Indexer] = 20.00;
    TcylArray[Indexer] = 20.00;
    TstagArray[Indexer] = 20.00;
  }
  delay(2000);
}

void loop()
{
  //Look to see if the pump override button pressed.
  if(digitalRead(ButtonA)==HIGH)
  {
    ConfirmPumpRun = true;
  }
  else  //if pump override button not pressed then all is running automatically.
  {
    //The debounce process looks to see if the PumpFlag has remained stable for
    //eight seconds or to a count of about 40 ... more or less.

    for(CycleCount = 0; CycleCount < 40; CycleCount++)
    {
      PumpFlagOld = PumpFlag;


      //The next few bits are the smoothing process.
      //Remove the reading being replaced from the running total.
      Tintotal =  Tintotal - TinArray[readIndex];    // the running totals for
      Touttotal = Touttotal - ToutArray[readIndex];   //the temperature readings
      Tcyltotal = Tcyltotal - TcylArray[readIndex];
      Tstagtotal = Tstagtotal - TstagArray[readIndex];


      //Read the Thermistor Data into the arrays
      ThermValue = analogRead(ThermPinA);  //What is the raw value from the Voltage Divider?
      TinArray[readIndex] = TempCalc(ThermValue);  //Just duck out to the calculation routine.
      ThermValue = analogRead(ThermPinB);
      ToutArray[readIndex] = TempCalc(ThermValue);
      ThermValue = analogRead(ThermPinC);
      TcylArray[readIndex] = TempCalc(ThermValue);
      ThermValue = analogRead(ThermPinD);
      TstagArray[readIndex] = TempCalc(ThermValue);

      //Add the new readings to the total
      Tintotal =  Tintotal + TinArray[readIndex];    //This process has swapped an old
```

```
Touttotal = Touttotal + ToutArray[readIndex];  //value for a new one and updated the
Tcyltotal = Tcyltotal + TcylArray[readIndex];  //total.  It is efficient with regard
Tstagtotal = Tstagtotal + TstagArray[readIndex]; //to minimised double handling.

//move the pointer along a bit.
readIndex = readIndex + 1;
//check to see if we've exceeded the array positions
if (readIndex >= numReadings)
{
  readIndex = 0;  //reset the pointer position to start of arrays.
}

//Calculate the 8 second average for use in the control olgic to follow.
Tin = Tintotal / numReadings; // Calculated Mean Temperature Values
Tout = Touttotal / numReadings;
Tcyl = Tcyltotal / numReadings;
Tstag = Tstagtotal / numReadings;


//Check whether there is a frost risk.
if(Tstag < FrostTemp)
{
  //Frost risk conditions are present.
  FrostRisk = true;
  //Has the solar water heater cooled to SWHFrostThresh temperature?
  if( Tout < SWHFrostThresh && Tin < SWHFrostThresh)
  {
    PumpFlag = true;  //Both SWH sensors are cold, turn on the pump
  }
  else
  {
    //The panel is still a little warm
    if(Tout > SWHOutHeatedTemp)
    {
      //The SWH has now got a fresh shot of hot water in it so the
      //pump can be turned off.
      PumpFlag = false;
    }
  }
}
else //The conditions are not at risk of freezing
{
  FrostRisk = false;
  //Check to see if the stagnation temperature is greater than the
  //hot water cylinder temperature.  If it is then it may be worth
  //running the pump.  If it's not then it's probably a fairly dull day.
  if(Tstag > Tcyl)
  {
    //It is a moderately sunny day and there is heat to be gained.

    //Is the SWH outlet temperature greated than the Tank cylinder temp
    //plus the threshhold value SWHCylTdiff.
    if(Tout > Tcyl + SWHCylTdiff)
    {
      //The SWH is hotter than cylinder by an appropriate margin so
      //start the pump.
      PumpFlag = true;
      if(Tout < Tin + SWHTdiff)
      {
        //The temperature difference across the panel is too low.  The quality of solar
```

```
            //intensity is too low.  Turn off the pump.
            PumpFlag = false;
           }
          }
          else
          {
           //The temperature differential between panel and cylinder to too low.
           PumpFlag = false;
          }
         }
         else
         {
          //There's no point running pump on a dull day.
          PumpFlag = false;
         }
        }
        //Checking the stability of the Pump state for both normal and frost operation.

        if(PumpFlagOld != PumpFlag)
        {
         CycleCount = 0;  //Unstable conditions therefore reset the counter.
        }

        // Display the readings on the LCD Display
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print(Tin);
        lcd.setCursor(7, 0);
        lcd.print(Tout);
        lcd.setCursor(0, 1);
        lcd.print(Tcyl);
        lcd.setCursor(7, 1);
        lcd.print(Tstag);
        delay(200);

      }



      ConfirmPumpRun = PumpFlag;  //At last a status is decided for normal operation.
     }


     digitalWrite(LEDPump, ConfirmPumpRun); //This makes the pump run and lights pump LED.
     digitalWrite(LEDFrost,FrostRisk);


}


double TempCalc(int ThermIn)
{
 double TempVal = 1/((1/298.00)+(1/4100.00)*log(1024.00/ThermIn-1.00));
 TempVal = TempVal - 273.00;  //Note the extra decimal places force float usage.
 return TempVal;
}
```

Real World Solar Water Heater Controller

In the real world there are imperfections. These require some modification of the perfect world solution described previously. This section describes the modifications I made to the solar water heater controller to suit the conditions and challenges encountered as I installed it into our solar water heater system. If you are looking at this project and attempting it on your own system you will find that some of the challenges mentioned here don't apply.

The description of the “perfect world solution” (for this solar water heater controller that is) can be found here: **Perfect World Solar Water Heater Controller**

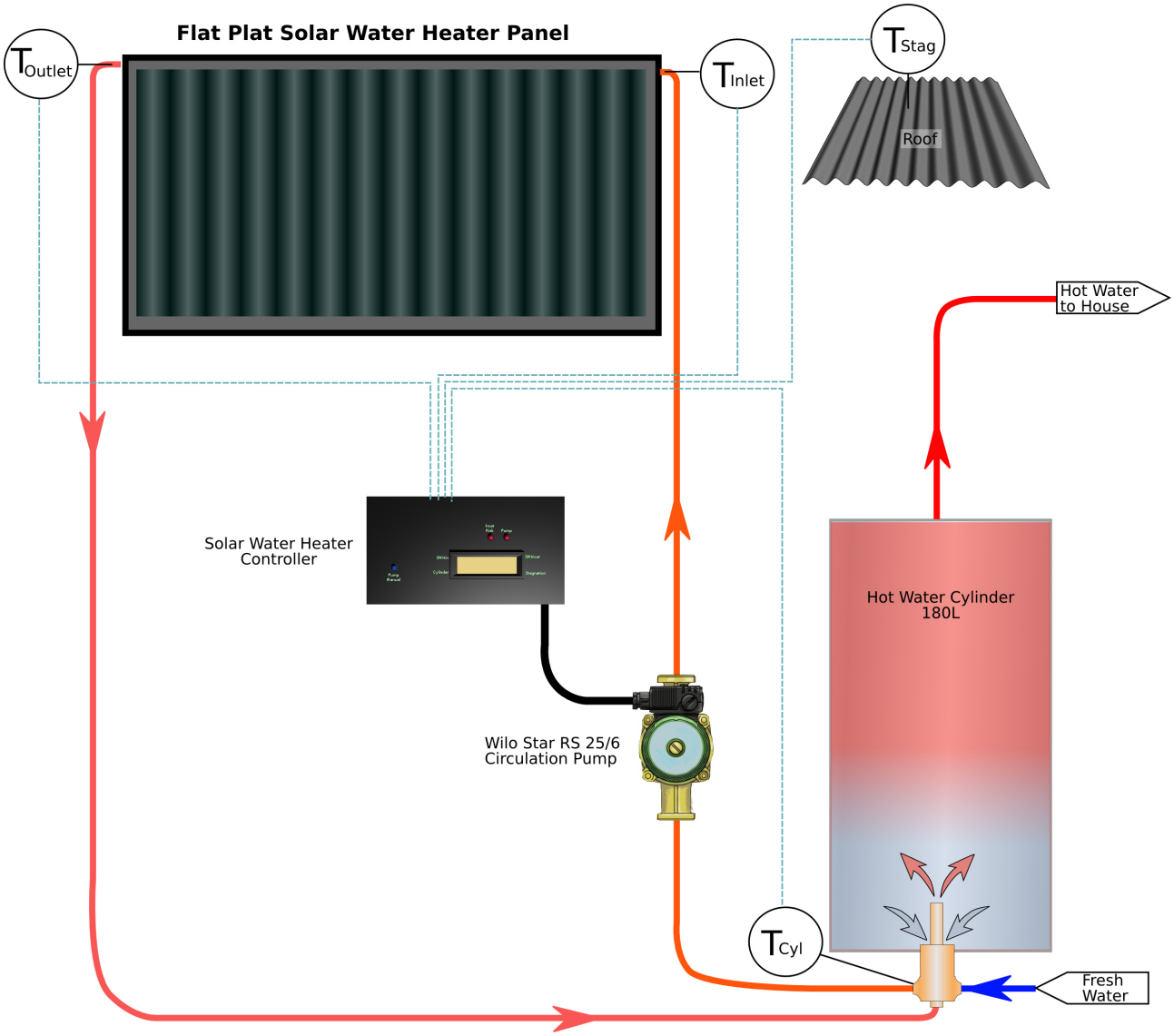
What Challenges the Real World Has in Store

When the time comes to install a retrofit system like this there are all sorts of challenges that rear their heads when you are scrambling around on the roof trying to connect this to that while still ensuring it will work properly. Here are the main issues I grappled with with my system.

- **Dealing with an existing system restricts some of your options to non-ideal solutions.**
- **Placement of the temperature sensors are non-ideal.**
- **Agreement between temperature sensors.** I found that even a selection of thermistors that were supposed to be the same did not agree and so I compared the temperature sensors against each other by putting the sensors together in a glass of water. By taking a number of readings at different temperatures I was able to develop an expression to relate the wayward sensor to the others. This appears as a one-line adjustment in the code.
- **Mystery temperature sensors.** When I finally got onto the roof and tried to install my temperature sensors I found it very hard to get a good location and ensure good contact with the water pipes. Seeing as the original controller had two thermistors already installed in some purpose made sockets I decided to use these. The problem was that these were unknown sensors with unknown curves. I needed to test these and develop the expressions for these sensors and add that to the code. This appears as a second function for converting the thermistor voltage readings to temperatures for the `Tout` and `Tcyl` sensors. This process is described here: **Finding an Unknown Thermistor's resistance curve**.
- **Electrical noise and interference**. All my testing had been done with a 12V battery because the 12V supplies had not arrived from Banggood at that time. This meant that there was very little electrical noise anywhere in the system. Once the system was running with the transformer, and connected to the pump, the action of the relay would sometimes upset the LCD display. All attempts to improve this never quite solved it. An oscilloscope indicated that after the addition of a number of 0.1µF capacitors (aka “Fairy-dust” according to Adafruit) the system noise was pretty minimal. Because the Arduino and everything else was not being upset by the noise, I concluded that the LCD screen itself was just too sensitive and if I was the build the system again I would use a different display. Incidentally I tried using a separate 5V supply to the Arduino and fully isolating the Arduino and 5V system from the relay but this made no difference.
- **A solar water heater panel is a very effective absorber, trying to simulate its performance is very hard**. The *Stagnation Temperature Sensor (Tstag)* had a critical role in the solar water heater controller logic in the “Perfect World” system. I had thought it would be easy to simulate the stagnation temperature of the panel by making a small black metal panel, mounting the sensor on it, and encapsulating it in a glass boiling tube. When I tried it out initially it looked promising. The plate got very hot with little need for careful arrangement (60°C). Unfortunately when it was mounted on the roof next to the panel I very soon found out just how effective at capturing heat the panels are. The stagnation temperature sensor was reading a high temperature, but the panel inlet and outlet temperature were at least 10°C higher so the logic did not work. I changed the set points so the stagnation temperature would not prevent the pump from running except on days when it would be bitterly cold, gray, dismal, and there would be no solar heat to capture. Because the stagnation temperature sensor still loses heat quite quickly, it still has a governing role detecting frost risk conditions.

Arrangement of the System

The original system has a not got ideal locations for the temperature sensors, and so there is some lag between target temperatures on the panel being reached and the temperature reaching the sensors when the pump is off. This is because the heat has to travel through the metal pipework and stationary water to get to the sensor.
No temperature sensors are located in the hot water storage tank either. The temperature used is on a mixing junction where the hot water is drawn from the tank to be circulated through the panel and where fresh make up water is added. This has the advantage in being responsive to cold water being added to the system. Newly added cold water will be drawn into the solar water heater circulation system before going into the hot water storage tank.



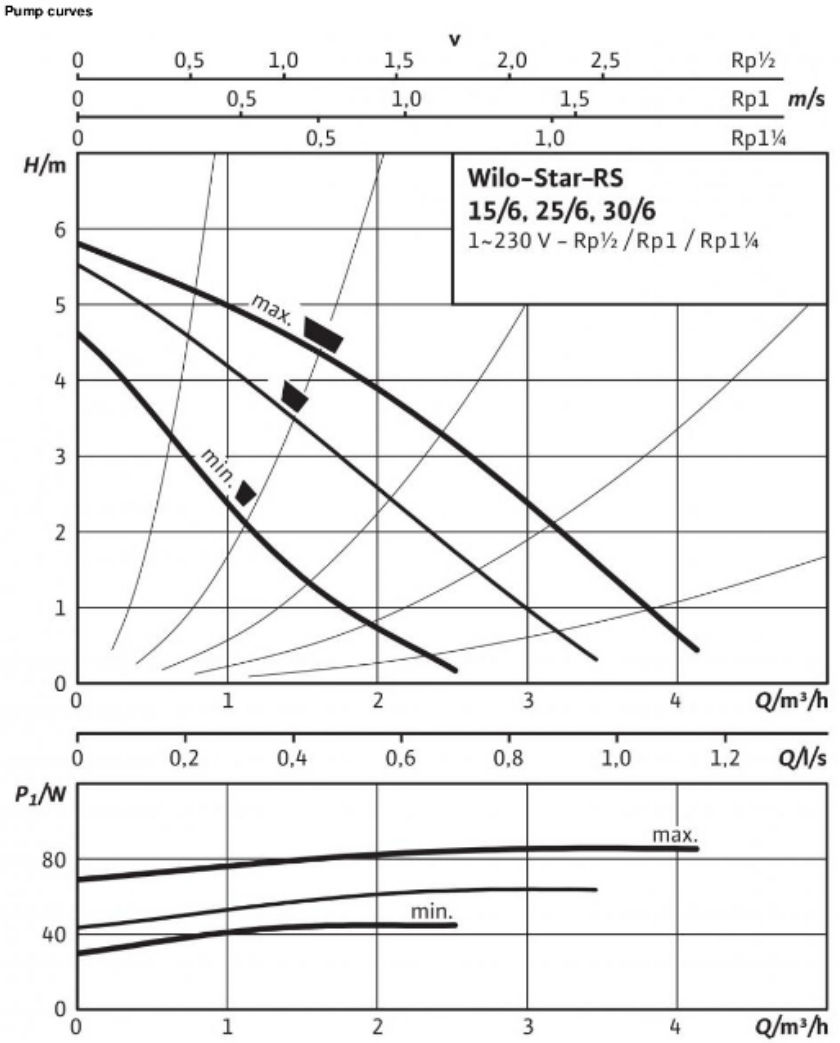
Where it would have been nice to have the stagnation temperature sensor (Tstag) attached to a bit of solar panel absorber within the panel enclosure, this was not possible. The best approximation was mounting it on the roof beside the panel where it would be exposed to the same amount of sunlight, wind, rain, and all other weather conditions affecting the solar panel.

Existing Equipment

Because I was retrofitting the controller to an existing system there were a number of key items that were reused (aside from the obvious ones like the panel and the hot water storage tank).

Pump and it's Protection

The pump is a Wilo Star RS 25/6 circulation pump. These are pretty common and well made. It has run for many years and never missed a beat and it is likely to run happily for many years yet. When I was trying to solve the electrical noise issue one of my first thoughts was “Grrrr! Noisy cheapo pump.” This wasn't the case at all, the Wilo pump was well protected with built in capacitors and compliant with various standards concerning electrical noise and interference. So, good pump, no problems there.
The pump is set for 99W output and so will draw no more than 0.41A.
If you are interested, here is the pump curve from the Wilo Star RS 25/6 datasheet.



Source: Wilo Datasheet: Star RS-25/6

Thermistors

As discussed above, I decided to make use of the existing Thermistors because they were designed to fit snuggly into the sensor pockets in the pipe fittings around the solar water heater system. I probably would have struggled to get as good thermal contact between the sensors and the pocket walls as the original sensors were getting.

The thermistors were unknown so I had to investigate how their resistance changed with temperature in order to find the ***Beta value*** for the thermistors and hence the expression for their temperature readings. I used a method as described here: **Finding an Unknown Thermistor's resistance curve**.

Because the thermistors were already mounted in a place where I had no control over the temperature being measured, I took a number of readings over the course of a day from when the system was cold to when it was running full power on a bright sunny day. My method was to disconnect the sensor and read its resistance and then reconnect it to the existing controller (it was still able to read the sensors) and get it to

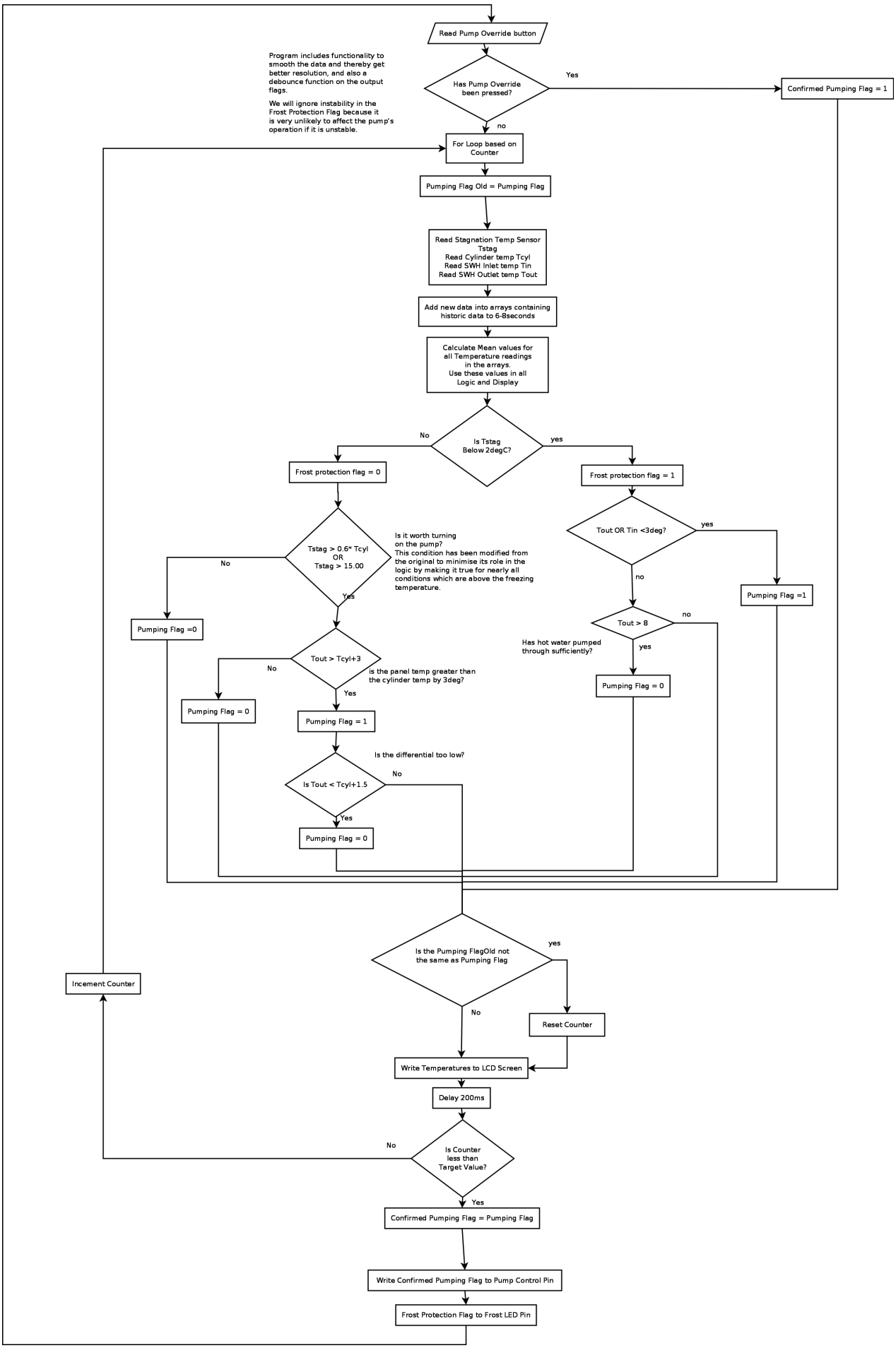
display the temperature. When the system was running full power during the heat of the day, I would bypass the controller and connect the pump directly to the mains supply so the water would still circulate.
The end result was a series of readings spanning the typical range of temperatures the system would be likely to encounter.

I was able to calculate a Beta value of 3410K.

Now that the system is installed and running I have observed that the two existing thermistors appear to read consistently 1.5°C lower than the temperature readings from the other thermistors with the known Beta Value. I suspect the problem is not with the method used to determine the Beta Value but with the readings from the original controller. Given that the Senztek controller was not all that well set, I shouldn't be surprised that it doesn't calculate the temperatures from the thermistors correctly.

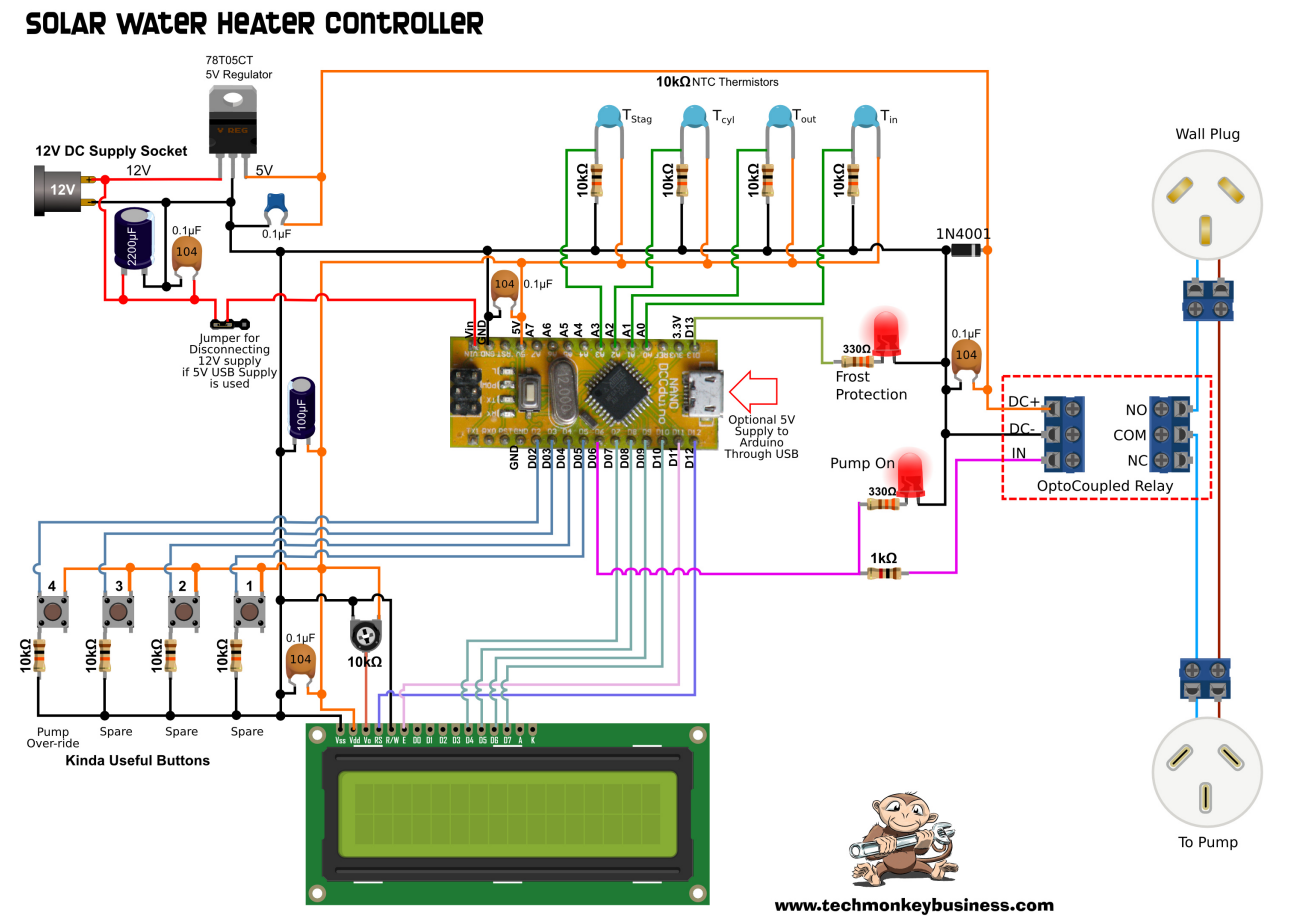
Logic for the Real World System

The logic shown here differs a little from the “perfect world” solar water heater controller system in that the Stagnation temperature sensor's role has been diminished by using set points that mean that it has the pump enabled most of the time. Also, the inlet temperature sensor is only for reporting now rather than having a role governing the pump operation. This change was partly due to the mismatch between the inlet and outlet temperature sensors and the potential risk of this causing poor performance. Instead the temperature differential between the mixer at the base of the hot water storage cylinder and the solar water heater collector outlet was used to govern the pump's operation.

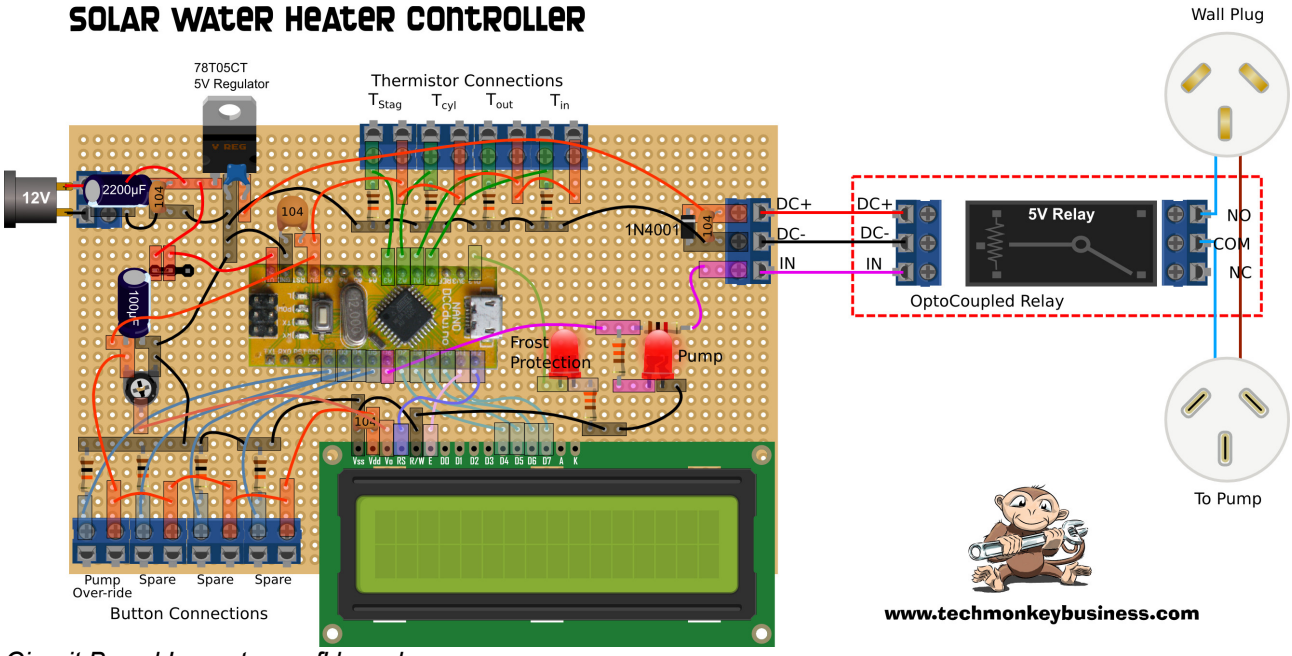

Read Pump Override button
Program includes functionality to smooth the data and thereby get better resolution, and also a debounce function on the output flags.
We will ignore instability in the Frost Protection Flag because it is very unlikely to affect the pump's operation if it is unstable.
Has Pump Override been pressed?
Yes
Confirmed Pumping Flag = 1
no
For Loop based on Counter
Pumping Flag Old = Pumping Flag
Read Stagnation Temp Sensor Tstag
Read Cylinder temp Tcyl
Read SWH Inlet temp Tin
Read SWH Outlet temp Tout
Add new data into arrays containing historic data to 6-8seconds
Calculate Mean values for all Temperature readings in the arrays. Use these values in all Logic and Display
Is Tstag Below 2degC?
No
yes
Frost protection flag = 0
Frost protection flag = 1
Tstag > 0.6* Tcyl OR Tstag > 15.00
Is it worth turning on the pump? This condition has been modified from the original to minimise its role in the logic by making it true for nearly all conditions which are above the freezing temperature.
No
Yes
Pumping Flag =0
Tout > Tcyl+3
No
is the panel temp greater than the cylinder temp by 3deg?
Yes
Pumping Flag = 0
Pumping Flag = 1
Is the differential too low?
Is Tout < Tcyl+1.5
No
Yes
Pumping Flag = 0
Tout OR Tin <3deg?
yes
Pumping Flag =1
no
Tout > 8
no
Has hot water pumped through sufficiently?
yes
Pumping Flag = 0
Is the Pumping FlagOld not the same as Pumping Flag
yes
No
Reset Counter
Write Temperatures to LCD Screen
Delay 200ms
Is Counter less than Target Value?
No
Yes
Incement Counter
Confirmed Pumping Flag = Pumping Flag
Write Confirmed Pumping Flag to Pump Control Pin
Frost Protection Flag to Frost LED Pin

Circuit for the Real World System

The circuit diagram and perf' board arrangement for the system are shown below. This is unchanged from the circuit for the “Perfect World” solar water heater controller.

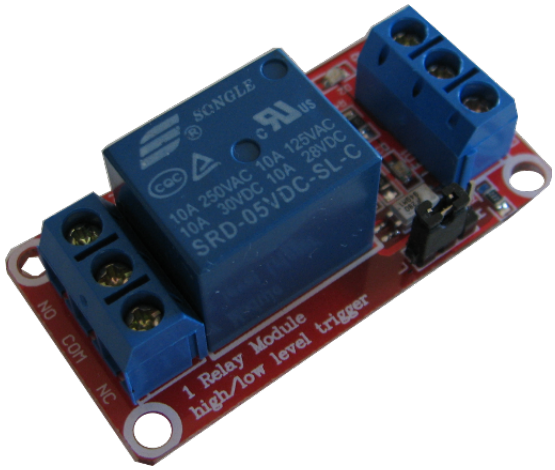


Circuit Diagram



Circuit Board Layout – perf' board.

At the heart of the system is an Arduino Nano. The demands on the Arduino are not large and so any Arduino could be used.



An opto-isolated relay module from Banggood was selected to control the pump. Flyback diodes and various LEDs are included in the package. All it requires is a ground connection, 5v supply, and a signal. Using the jumper on the module the relay can be set to trigger with a high or a low signal. The manufacturer claims the relay can handle 240V 10A AC. I have seen a few comments in various forums that suggest the terminal connections themselves are not capable of carrying that sort of current, but the pump used in our solar water heater system only draws 0.5A anyway, so the module is quite suitable for the task.

The LCD screen is a typical 16x2 type. I have not connected the back light because that would be a little pointless.

The other components of significance are the thermistors and the 5V regulator.

Here is a list of the components used.

- Arduino Nano
- LCD display – 16x2. This is not the i2c version – although I would suggest the 12c version might be better.
- Two NTC 10kΩ thermistors with Beta Values of 4100K.
- Two NTC 10kΩ thermistors with Beta Values of 3410K from the original system.
- 12V DC Plug pack supply.
- A button
- Two LEDs
- MC78T05CT regulator capable of outputting 3A at 5V not that we're using anywhere near that much.
- A 2200µF electrolytic capacitor.
- A 100µF capacitor.
- some 10kΩ resistors.
- A couple of 330Ω resistors.
- A 1kΩ resistor.
- A 1N4001 diode acting as a flyback diode for the relay coil, but probably completely superfluous with the flyback diode built into the relay module already.
- A handful of 0.1µF capacitors liberally sprinkled throughout the circuit like “fairydust” as suggested by Adafruit.

Coding for the Real World System

Here is the code for the real world system.

Hopefully the comments through the sketch will explain how it works.

You can download the sketch from here: **SWHControllerv5.ino**

The sketch does not use any libraries that don't come with the standard Arduino IDE. I think it works on older versions of the Arduino IDE too. In short, there is nothing particularly fancy in this coding.

```
/*
 * SWHControllerv5.ino
 * Hamish Trolove 2017 - www.techmonkeybusiness.com
 *
 * This sketch is to run a solar water heater and optimise the solar heat
 * yield.  It is also designed to be easily modified to suit local conditions.
 * A process similar to the Debounce routine in the Arduino Cookbook (pg155)
 * has been used to protect against short cycling of the pump. In addition to
 * this the Array average process (https://www.arduino.cc/en/tutorial/smoothing)
 * has been used on all data to smooth the temperature readings and gain better
 * resolution.
 *
 * This version reduces the effect of the Stagnation temperature for running
 * the pump while heating.  The reason for this is that the Stagnation Temperature
 * enclosure as built cannot quite get up to the temperature of the panel which
 * suggests it suffers from too much heat loss.  If the stagnation temperature
 * sensor can be installed in the Solar Water Heater enclosure then
 * SWHControllerv4.ino or SWHControllerv4a.ino would be able to be used.
 *
 * In this version Tcyl and Tout are the previous controller's sensors which appear to be
 * 10kohm NTC thermistors with Beta values of 3410.
 *
 * Tin has also been sidelined and in this version is only supplying information
 * for display on the screen.  It is not used in the logic during normal conditions.
 *
 * There are four thermistors which are located;
 * Solar Water Heater Inlet - Tin
 * Solar Water Heater Outlet - Tout
 * Tank freshwater and SWH intake junction - Tcyl
 * and Tstag on a black plate in a pyrex tube to measure stagnation
 * temperatures and frost conditions.
 *
 * The temperatures will control the operation of a pump through an optically isolated relay.
 *
 * Each thermistor reading is displayed on the 16x2 LCD screen.
 *
 * Four buttons are connected.  At this stage only one is used and that is an override to turn
 * on the pump.
 *
 * Circuit is described on www.techmonkeybusiness.com.
 *
 * Connections are:
 *
 * 10K B=4100 NTC Thermistors with 10K resistors in voltage dividers on A0, A1, A2, and A3
 * Push buttons with pull up resistors on D02, D03, D04, and D05
 * LED to indicate Frost Protection mode on D13
 * Pump relay control line on D06 (also connected to an LED)
 * LCD connections:
 * D12 to LCD RS pin
 * D11 to LCD E pin
 * D10 to LCD D7 pin
 * D09 to LCD D6 pin
 * D08 to LCD D5 pin
 * D07 to LCD D4 pin
 *
 *The LCD panel is not lit.
 */
```

```
#include <LiquidCrystal.h>
#include <math.h>  // We need this to calculate temperature values from thermistors.
const int ThermPinA = A0;  // The Tin thermistor on Analogue Pin 0
const int ThermPinB = A1;  // The Tout thermistor on Analogue Pin 1
const int ThermPinC = A2;  // The Tcyl thermistor on Analogue Pin 2
const int ThermPinD = A3;  // The Tstag thermistor on Analogue Pin 3

const int ButtonA = 02; // Some button pins defined (Pump Override)
const int ButtonB = 03;
const int ButtonC = 04;
const int ButtonD = 05;

const int LEDFrost = 13; // Front protection mode indicator LED
const int LEDPump = 6; // Pump running indicator LED.

boolean PumpFlag = false; //This is the flag to control the pump.  All
                    //control logic acts on this flag.
boolean FrostRisk = false; //Frost risk flag for driving LED.
boolean PumpFlagOld = false; //Flag to hold previous PumpFlag State
boolean ConfirmPumpRun = false; //Once all debouncing has occurred this is
                        //the flag to run the pump.

int CycleCount = 0;  //A counter for the number of times the loop has run for
                //while awaiting stable readings.
int ThermValue = 0; // Somewhere to stick the raw Analogue Pin Value.
double Tin = 0.00; // Calculated Temperature Values after smoothing
double Tout = 0.00;
double Tcyl = 0.00;
double Tstag = 0.00;

//Arrays to hold historic data
const int numReadings = 40;

double TinArray[numReadings];      // the readings from the Temperature sensors
double ToutArray[numReadings];
double TcylArray[numReadings];
double TstagArray[numReadings];

double Tintotal = 800;   // the running totals for the temperature readings
double Touttotal = 800;  //Starting value is 20 x 40 ie starting value
double Tcyltotal = 800;  //in arrays.  If the arrays had been loaded with
double Tstagtotal = 800; //zeros then these figures would also be zero.

int readIndex = 0;              // the index of the Historic Data Array Pointer

double FrostTemp = 4.00; //Temperature below which Frost is a risk.
                //Adjust for local conditions.
double SWHFrostThresh = 5.00; //Temperature at SWH at which pump
                //will start if running in Frost protection mode.
double SWHOutHeatedTemp = 8.00; //Temperature which will indicate that
                //hot water is in SWH for frost protection.
double SWHCylTdiff = 2.50; //Temperature difference between cylinder
                //and solar water heater before pumping starts.
double SWHTdiff = 1.50;  //Min temperature difference between cylinder
                //and solar water heater before pumping is stopped.

LiquidCrystal lcd( 12, 11, 7, 8, 9, 10 );   //Pins for the LCD Screen
```

```
void setup()
{
  lcd.begin(16, 2);  //set up the LCD's number of columns and rows:
  lcd.setCursor(0, 0);
  lcd.print("Starting");  //Print a message to the LCD
  pinMode(ThermPinA, INPUT);
  pinMode(ThermPinB, INPUT);
  pinMode(ThermPinC, INPUT);
  pinMode(ThermPinD, INPUT);
  pinMode(ButtonA, INPUT);  //This is the pump override button.
  pinMode(ButtonB, INPUT);  //spare
  pinMode(ButtonC, INPUT);  //spare
  pinMode(ButtonD, INPUT);  //spare
  pinMode(LEDFrost, OUTPUT);
  pinMode(LEDPump, OUTPUT);

  //Load the arrays with normal(ish) values to begin with.
  for (int Indexer = 0; Indexer < numReadings; Indexer++)
  {
    TinArray[Indexer] = 20.00;
    ToutArray[Indexer] = 20.00;
    TcylArray[Indexer] = 20.00;
    TstagArray[Indexer] = 20.00;
  }
  delay(2000);
}

void loop()
{
  //Look to see if the pump override button pressed.
  if(digitalRead(ButtonA)==HIGH)
  {
    ConfirmPumpRun = true;
  }
  else  //if pump override button not pressed then all is running automatically.
  {
    //The debounce process looks to see if the PumpFlag has remained stable for
    //eight seconds or to a count of about 40 ... more or less.

    for(CycleCount = 0; CycleCount < 40; CycleCount++)
    {
      PumpFlagOld = PumpFlag;


      //The next few bits are the smoothing process.
      //Remove the reading being replaced from the running total.
      Tintotal =  Tintotal - TinArray[readIndex];    // the running totals for
      Touttotal = Touttotal - ToutArray[readIndex];   //the temperature readings
      Tcyltotal = Tcyltotal - TcylArray[readIndex];
      Tstagtotal = Tstagtotal - TstagArray[readIndex];


      //Read the Thermistor Data into the arrays
      ThermValue = analogRead(ThermPinA);  //What is the raw value from the Voltage Divider?
      TinArray[readIndex] = TempCalc(ThermValue);  //Just duck out to the calculation routine.
      ThermValue = analogRead(ThermPinB);
      ToutArray[readIndex] = TempCalc2(ThermValue);  //Note using TempCalc2 function for Beta 3410
      ThermValue = analogRead(ThermPinC);
```

```
TcylArray[readIndex] = TempCalc2(ThermValue);  //Note using TempCalc2 function for Beta 3410
ThermValue = analogRead(ThermPinD);
TstagArray[readIndex] = TempCalc(ThermValue);

//By comparison with the other thermistors, Tin reads low.  The following lines
//have been added to correct this value.  Comment out as desired.
TinArray[readIndex] =  TinArray[readIndex] + 0.0092 * TinArray[readIndex] + 1.28;


//Add the new readings to the total
Tintotal =  Tintotal + TinArray[readIndex];    //This process has swapped an old
Touttotal = Touttotal + ToutArray[readIndex];  //value for a new one and updated the
Tcyltotal = Tcyltotal + TcylArray[readIndex];  //total.  It is efficient with regard
Tstagtotal = Tstagtotal + TstagArray[readIndex]; //to minimised double handling.

//move the pointer along a bit.
readIndex = readIndex + 1;
//check to see if we've exceeded the array positions
if (readIndex >= numReadings)
{
 readIndex = 0;  //reset the pointer position to start of arrays.
}

//Calculate the 8 second average for use in the control logic to follow.
Tin = Tintotal / numReadings; // Calculated Mean Temperature Values
Tout = Touttotal / numReadings;
Tcyl = Tcyltotal / numReadings;
Tstag = Tstagtotal / numReadings;


//Check whether there is a frost risk.
if(Tstag < FrostTemp)
{
 //Frost risk conditions are present.
 FrostRisk = true;
 //Has the solar water heater cooled to SWHFrostThresh temperature?
 //This is indicated by either Tin or Tout being below the threshhold Temp.
 if( Tout < SWHFrostThresh || Tin < SWHFrostThresh)
 {
  PumpFlag = true;  //Both SWH sensors are cold, turn on the pump
 }
 else
 {
  //The panel is still a little warm
  if(Tout > SWHOutHeatedTemp)
  {
   //The SWH has now got a fresh shot of hot water in it so the
   //pump can be turned off.
   PumpFlag = false;
  }
 }
}
else //The conditions are not at risk of freezing
{
 FrostRisk = false;
 //Check to see if the stagnation temperature is greater than 15degrees
 //or greater than 60% of the cylinder intake temperature.
 //If it is then enable the the pump.  If it's not then it's
 //probably a pretty awful day.
 if(Tstag > 0.6* Tcyl || Tstag > 15.00)
```

```
      {
       //Is the SWH outlet temperature greated than the Tank cylinder temp
       //plus the threshhold value SWHCylTdiff.
       if(Tout > Tcyl + SWHCylTdiff)
       {
        //The SWH is hotter than cylinder by an appropriate margin so
        //start the pump.
        PumpFlag = true;
       }
       if(Tout < Tcyl + SWHTdiff)
       {
        //The temperature difference across the panel is too low.  The quality of solar
        //intensity is too low.  Turn off the pump.
        PumpFlag = false;
       }

      }
      else
      {
       //There's no point running pump on a dull day.
       PumpFlag = false;
      }
    }
    //Checking the stability of the Pump state for both normal and frost operation.

    if(PumpFlagOld != PumpFlag)
    {
     CycleCount = 0;  //Unstable conditions therefore reset the counter.
    }

    // Display the readings on the LCD Display
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(Tin);
    lcd.setCursor(7, 0);
    lcd.print(Tout);
    lcd.setCursor(0, 1);
    lcd.print(Tcyl);
    lcd.setCursor(7, 1);
    lcd.print(Tstag);
    delay(200);

   }



   ConfirmPumpRun = PumpFlag;  //At last a status is decided for normal operation.
  }


  digitalWrite(LEDPump, ConfirmPumpRun); //This makes the pump run and lights pump LED.
  digitalWrite(LEDFrost,FrostRisk);


}


double TempCalc(int ThermIn) //For Thermocouples Tin and Tstag with beta values 4100
{
  double TempVal = 1/((1/298.00)+(1/4100.00)*log(1024.00/ThermIn-1.00));
```

```
  TempVal = TempVal - 273.00;  //Note the extra decimal places force float usage.
  return TempVal;
}

double TempCalc2(int ThermIn2) //For Thermocouples Tout and Tcyl with beta values 3410
{
  double TempVal = 1/((1/298.00)+(1/3410.00)*log(1024.00/ThermIn2-1.00));
  TempVal = TempVal - 273.00;  //Note the extra decimal places force float usage.
  return TempVal;
}
```

What Could be Better.

So this project was a one off, and I have no intention of going through it again to make it all perfect. But having gone through the project here are a few thoughts about what could be done better if I were to do it again.

- Initially I designed the system for a 12v supply. As it happened there was nothing in the system that could not have run off a 5V supply. Rather than dump energy through the regulators on the board and on the Arduino, I should have just used a 5V wall transformer adapter thing.
- It would have been nice to have the same sensors throughout the system. If they had all been calibrated and compared, then there would be the ability to use more tightly defined set points rather than having to account for the uncertainty created by having mismatched temperature sensors.
- A session calibrating the sensors properly – as mentioned above this would have allowed me to define the set points more tightly and so potentially squeeze just a little more heat out of the sun.
- A better display – Yes. The display. Well there are several solutions that are possible here. I do not have an i2c version of the 16x2 LCD display to try but the i2c pins on the Arduino are available for use so it would be a relatively simple modification to the circuit and coding to cater for this. An alternative would be to use a simple 4 digit display and a couple of extra buttons to step through the sensor readings. The other alternative would be to not use a display at all, although the display is quite useful when setting up the system and tuning it.
- Solid State Relays. To reduce noise in the system solid state relays would be a useful addition. Because the pump's current draw is relatively low, it is not hard to find suitably rated solid state relays to do the job. The modules available use all the same connections as the opto-isolated module used in this project. The only change would be inverting the trigger signal because the solid state relays modules that I have encountered all seem to use a Low trigger.
- Stagnation Temperature sensor mounted in the SWH enclosure and using the same materials/ absorptive surfaces as the actual panels. Yep, accurately simulating the panel would be fantastic, but it would be expensive and probably not gain enough of a performance advantage to make it worthwhile.

Isn't hindsight is a wonderful thing?

The information provides is accurate to the best of my knowledge, but describes what I did and what worked for me. No responsibility is accepted for any failure of equipment or damage caused by attempting to follow these instructions. You're probably working with main electricity, so watch what you're doing with that stuff. It bits hard. The information is also provided in an as-is-where-is basis and as such I cannot provide support to help you solve your own problems you may encounter.

www.techmonkeybusiness.com