# Furnace Monitoring and Freeze-up Prevention

## What is it?

The problem is that I have one zone that is prone to freeze-up. The second floor zone. It passes through several unheated spaces and when the temperature has gotten down in the single digits, we have had it freeze-up twice. Fortunately, no damage to the pipes (at least that I know of).

Specifically, I think the passage from the bathroom to the front bedroom radiator is the likely problem. The pipe has a wrapping of pipe insulation on it. Prior to the refinishing of the sitting room ceiling, the soffit was unvented, the pipe was just laying on the old insulation, and it had no insulation on it. Now, the soffit is vented, the pipe is raised up above the insulation, but it is wrapped in pipe insulation.

So the idea is to monitor the outside temperature and periodically run water through the zone even when it isn’t calling for heat. Since when it is that cold, the furnace is likely to be running regularly anyway, I don’t think it is necessary to call for heat on the zone or to explicitly trigger the circulator. Rather, just detect when the circulator is running and open the zone valve. I think a couple minutes is enough to get any chilled water out and melt anything that might have already frozen (if the circulator is on, then the boiler is hot).

Once I start wiring up to the furnace, then collecting information is easy enough. So going to wire up to additional inputs and monitor (collect data) on its operation.

## Approach

Arduino because it is cheap and I have one. Could initially use the UNO, but may go to a smaller form factor if I want to go permanent. Could use the AdaFruit Feather 32u4 Adalogger ($22) as it already includes an SD card for recording data. It doesn’t have the real time clock (I don’t think), but that is OK. It is 3.3V logic, which drives some of the other decisions.

I currently have the AdaFruit Music Maker Shield (on loan) that has the SD card on it. So I can use that until I have to give it back.

TMP36 for temperature monitoring because it is cheap and I have one. Need to run a 3 (or more) conductor wire from the furnace to the outside. The signal coming back is analog (500 mV = 0 degrees C, 10 mV/degree C, so more than sufficient range) and may have to be adjusted due to the length of the wire and readings may have to be ‘filtered’ due to noise, or may have to use coax, or may have to go to remote sensor. But try the simple thing first.

Sensing the 24V AC signals used on the furnace is interesting. Some searching turned up a simple circuit on the Arduino site using a rectifying diode, capacitor, resistor, and a zener diode.

Din

Gnd

V24

R1

R2

D

Z

C

With a little experimenting, I came up with R1 = R2 = 10K ohm, C = 1 uF, and Z = 4.8 V. R2 is simply to bleed off the signal when V24 goes off (without it, it takes several seconds). The current through R1 is about 20V / 10K ohm = 2 mA and power of .04 W. Din ends up about 4.1V.

Microchip MCP23S17 is a 16-bit I/O Expander with a SPI interface. This will allow the connection to all of the 24V signals to be over a simple SPI interface (or I2C, but I need SPI for the SD card anyway). Also an 8-bit version. Makes the connection to the Arduino less dependent on its form factor in the event that I switch to the Feather. And it comes in a through-hole package.

All 3.3 V makes it easier, includes the SD card slot, and interfaces to the GPIO chip directly.

Another interesting option is the Adafruit MO Adalogger. It is actually a 48 Mhz ARM M3 rather than an AVR (not that I’m trying to avoid the AVR – either 8 or 16 MHz versions would do everything I need for this project). But it is the same cost as the AVR version. And from a power perspective, it is actually lower power than the AVR.

## Channels

So what to monitor?

5 calls (HW, main, family, second, basement)

5 zones (HW, main, family, second, basement)

Inducer

Burner

Circulator

## Anti-Freeze Algorithm

The idea for preventing freeze-up is to select a minimum circulate interval based on the temperature, and then make sure that water circulates at least that often. Each time the zone is opened by the furnace, the interval resets. Once the interval expires, it waits for the next time the circulator is running and opens the valve for N minutes. After the time is up, it resets the interval and returns control of the zone to the furnace.

So for this, need the zone wire and the circulator wire as inputs to the algorithm. Then need to be able to command the zone. The relay that came in the kit will do 3A @ 5V = 15W. No indication of max voltage. I’m willing to try 24VAC at 6.5W. Use the diodes to protect the digital pin from the inductive load of the coil.

In the absence of a thermometer, I can simply include an on-off button and an LED indicator. Push the button, the light turns on and it is active. Push the button, the light turns off, and it is inactive. Wa-la!

## Control

Use a relay for controlling zone valve. The control valves are 24V 6.5W, so ¼ A (or so).

Omron Electronics G5V-1-DC3. SPDT. 3V coil, 50 mA. 1A@24VAC, .5A @ 125 VAC contacts. Idea would be to put the zone valve on the common contact, pass through the furnace voltage in the normally closed position, and connect the other terminal to 24V (so when we turn it on, it overrides whatever the furnace is asking for). Do so prevents this from ever driving 24V back to the furnace output. $2.

## Purging Heat

Once a zone stops calling, the current control system will leave the valve for the last zone on to dump the remaining heat in the boiler into the house. This makes sense except that it doesn’t. In this particular case, if the last zone was zone 2, then it tries to bleed off the heat into a zone that doesn’t sink heat very well, so it takes a long time. Better would be to dump it into a zone like the main floor zones (or even the basement).

One approach could be to always use the main floor (since the heat rises to the second floor anyway). Another would be to come up with a most-likely-to-call. That would require some tracking of call frequency and the last call, though that seems too complicated.

Note that my current scheme for wiring up the zone relay doesn’t actually allow this – all it allows the Arduino to do is to activate a zone (not deactivate a zone). So could make it simply add the main floor zones to the heat dump (and not turn off the second floor). This does have the problem of not knowing whether the zones are turned on. For example, the water heater was running and it is dumping heat. During the winter, the best place is the house. During the summer, the best place remains the water heater.

This does require monitoring all of the call lines and zone valves (which I was going to do anyway) and adding two more (at least) zone relays for the main floor zones.

## Investigations

Writing to the SD Card. I disabled the write capability for the WMS project, so I need to turn that back on. I suspect there is some example code on the Adafruit site for the logging shield. Should be very similar to writing in any other file system (since reading is that way).

Need to finalize the zener diode values and the resister values (particularly R2).

## Engineering Prototype

Arduino Uno

11, 12, 13 are used by SPI. 10 is also associated with the SPI bus and seems to behave strangely in general use.

#define codecCs 7 // VS1053 chip select pin (output, set LOW)

#define codecDCs 6 // VS1053 Data/command select pin (output, set LOW)

#define codecCardCs 4 // Card chip select pin (output)

#define codecDReq 3 // VS1053 Data request, ideally an Interrupt pin (input, don’t enable)

Adafruit Music Maker Shield, for the SD Card

2 monitor circuits.

digital 8 – zone 2 valve.

digital 9 – circulator.

1 zone control.

digital 10 – output, zone 2 relay.

## Parts

(1) AdaFruit Feather 32u4 Adalogger $22. Currently out of stock, so maybe the Pro Tricket ($10) + SD Card Breakout board ($15).

(2) Relay $2 (digikey).

(1) GPIO chip $1.20 (digikey).

(13) rectifier diodes.

(13) capacitors, 1 uF 35V.

(13) zener diodes.

(13) 10k ohm resistors.

(13) ?k ohm resistors (was 10k, but maybe less going to 3.3V).

(1) Power supply (use 5V supply from Altera Eval Board).

(1) Prototype board (w)

(1) TMP36 (use one from Arduino kit).

## What I Did

Adafruit Metro Mini because it is 5V and has a regulator.

Adafruit SD Card Breakout.

Adafruit RTC Breakout.

½ size perma-proto board. This contains the Metro, SD card, and RTC. It is designed to sit on top of the full size board and pass certain connections down through headers. But can easily be removed to make a different monitoring system by replacing the lower board.

full size perma-proto board. This contains the I/O expander, the 24V sensing circuits, and the relays. It also contains the button and the LEDs because I ran out of space on the half-size board (oops).

Channel assignments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wire | Harness C(all) | Io pin | Harness V(alve) | Io pin |
| Brown-white | n/c |  | Zone HW valve | 7 |
| Brown | n/c |  | Zone 1 valve | 6 |
| Green-white | n/c |  | Zone 2 valve | 5 |
| Green | Zone 4 call | 11[[1]](#footnote-1) | Zone 3 valve | 4 |
| Blue-white | Zone 3 call | 12 | Zone 4 valve | 3 |
| Blue | Zone 2 call | 13 | Circulator | 1[[2]](#footnote-2) |
| Orange-white | Zone 1 call | 14 | Inductor | 2 |
| Orange | Zone HW call | 15 | n/c[[3]](#footnote-3) |  |
| Purple | Zone 1 call out[[4]](#footnote-4) | 10 |  |  |
| White | Zone 2 call out | 9 |  |  |
| Blue | n/c[[5]](#footnote-5) |  |  |  |

## Sleep Mode

Since the device is powered from a wall wart (and there is in fact a second AVR running just the USB/serial), power saving really isn’t a big issue with this device. Nevertheless, I wanted to explore how to run the device in a low power state.

My first crack at this involved a library called sleep\_n0m1.h (no idea what the n0m1 is about). This library claimed to allow you to do a low-power sleep for a specified amount of time. It does actually do so by using the watchdog timer (which is active in all of the sleep modes) to achieve an approximate delay, and going through multiple cycles if the specified delay is longer than the available watchdog timeout.

In some ways it does just what I want – a low power sleep/delay. And doing so allows the system to run just like it currently does by waking up every so often and checking its timers to see if there is something that needs to be done. Ah, there is the rub – in the interesting sleep modes (deeper than Idle), the timer isn’t running, so millis() isn’t updating, which means my timers don’t know any time has elapsed.

So while simple to use, it is really a poor man’s way to use the sleep modes. And since initially trying it, I have since added use of the watchdog timer as, well, a watchdog.

But the good thing is that I learned about using both the sleep modes and the watchdog. So now what I am interested in doing is using the sleep.h members directly.

I have two inputs that I want to use to trigger action on the part of the monitor. One is the button push (tied to pin 2, INT0) and the ioExpander (tied to pin 3, INT1). The challenge with INT0 and INT1 is that to wake from a sleep mode, they must be level triggered. And that means that I have to deal with making them go away (in the case of the ioExpander) or masking them off (in the case of the push button). Probably not too hard. Since I have no control over how long the user presses the button, the interrupt is set active LOW, detached when the interrupt occurs, and re-attached after the debouncing and push duration detection. The IO controller interrupt is cleared by reading a particular register, so no problem. The processing for both is actually done in the main loop.

There are also the pin change interrupts. They are active in all sleep modes. Pins 2 and 3 (INT0 and INT1) are also PCINT18 and PCINT19.

How to handle SimpleTimer? Since I’m not really running time critical tasks, I could take the approach that all timers have expired after any deep sleep and run them all.

But that doesn’t address my time-based functions, such as anti-freeze which triggers based on elapsed time (and possibly no change to external stimulus, though in the conditions it is needed it is likely that they are changing). Which then requires that I track elapsed time.

So use of sleep requires a system approach.

Elapsed time has to be based on the RTC, not on the timers.

Really can’t use the watchdog as a watchdog. Doing so requires that something be waking up the device at a regular interval less than the watchdog timeout. I don’t have such a thing as my external events could be occurring on the range of minutes or hours (the hot water keeps it going in the off season). That I am using it now is because I was having trouble for a while, so maybe time to turn that back off.

While I like SimpleTimer for an event-driven system, it really falls down when the millis() function doesn’t work. I have a couple of things running on my timer. 1 second processTestScript (DIAG\_STANDALONE), 500 msec ledTimerExpired (flash the LED), 1 hour checkForSecondsOverflow (since I am storing 16-bit seconds), 250 msec watchdog tickle, and 1 second processMonitorTimer (the main thing my device does). While the course RTC would be fine for basic operations, it doesn’t allow any timekeeping at lower resolution intervals.

Could sleep in 1 second intervals for the main processing, use the RTC to record actual elapsed time, in seconds. If sleeping for power, then having LED’s on sort of defeats that anyway. So likely that the LED’s would be used to indicate feedback to user button press, and then be off the rest of the time. For real power savings, would want an Arduino without USB and the corresponding AVR running it.

Or I can stick to Idle mode so that the timer clocks keep running and it works much like it currently does.

Hum, don’t like using the WDT? Could wire up the square wave output of the RTC to a pin and use it to trigger a 1 second (or whatever) interrupt to wake it up.

Ok, I got a strategy.

I had already modified SimpleTimer to have 1 msec and 1 second resolution to have long intervals fit in a 16-bit delay value. Now I have split them into ‘fine’ and ‘course’, with fine being driven by the millis() as is currently, and course being driven by the RTC. Both functions can actually be replaced by the client, which is how the RTC gets connected since I don’t want SimpleTimer to require the RTC.

Then the main loop asks SimpleTimer if there are any fine timers running. If so, it sleeps using IDLE. If not, it can do the deeper sleep that turns off the IO clocks. In either case, it asks SimpleTimer how long it is until the next timer expires, and uses that to decide how long to sleep.

The use of avr/sleep.h was driven by the race condition between checking for conditions under which I want to sleep and actually sleeping. There are descriptions of how to do this right, so I basically followed them. The use of the n0m1 library didn’t give that sort of control.

It still uses the watchdog to wake up from the sleep (or either of the two interrupts). It still loops as did the sleep library so that it goes back to sleep after each millis() interrupt.

The watchdog has an interesting feature. You can connect it to an interrupt as is done here, but you can still leave it enabled to trigger a system reset if the interrupt isn’t handled within a second watchdog interval. So I think that this is what I want to do with the watchdog.

Use it to wake from sleep as is currently. Make sure both WDIE and WDE are set (it is the later that will do the reset if the interrupt isn’t handled). When we are woken up, rather than disabling the watchdog, set it up for our usual timeout of 2 s, but don’t set WDIE so it will simply reset the system if violated. Have to think about whether I actually need to worry about tickling it. I’m inclined to say ‘no’ and simply include a wdt\_reset in any loop that may run a long time (like the button debouncing). And extend the timer during the attempted SD card initialization (since it can take a couple seconds if the card isn’t there).

May not be good as a general strategy since it relies on having a RTC. But even so, it allows more use of IDLE then would otherwise. And it seems like any really low power thing is going to use an RTC as a time base anyway (they don’t use much power).

1. I inadvertently reversed the order of the port B channels. Meant for the ‘out’ channels to be the upper three. [↑](#footnote-ref-1)
2. I inadvertently swapped orange-white and blue when connecting the harness to the sense circuit. [↑](#footnote-ref-2)
3. This channel was intended for the burner, but the sense circuit is not loaded (needs to sense 13V AC). [↑](#footnote-ref-3)
4. The two ‘call out’ relays are setup as active-short-to-ground. This works well for triggering a call because the signal is normally at 24V and pulling it low signals the call (that is how the thermostats do it). Each relay has one dedicated wire and shares the common ground. [↑](#footnote-ref-4)
5. Going to convert the third relay so that all three terminals go to the panel. This allows the relay to interrupt or activate a call. In this case, I’m going to tie the HW call input to the common terminal and the HW thermostat to the normally closed terminal, allowing the signal to pass unless my box actively interrupts the signal. The same could be used to control a valve by connecting the valve input to the common terminal, the panel valve output to the normally closed terminal, and 24V to the normally open terminal – so activating would tie the valve to 24V and open it. [↑](#footnote-ref-5)