**Combustion Monitoring code outline**

***Summary of scan sequence***

* *Capture time at top of second* 
  + *Timestamps can be 1-sec resolution*
* *Scan inputs*
  + *Read all input parameters*
  + *Check for new data from XBee*
  + *If we can launch a process to query pressure sensor repeatedly, we can continue with other steps up to setting pressure valves. (Must not alter valve settings until pressure measurements are complete.)*
* *Process data* 
  + *Apply zero offset correction to pressure values*
  + *Convert to engineering units: degrees F, Pa, etc.*
  + *Hold results; will decide how to use them later*
* *Determine status of both burners*
* *Assign operating mode of wh and furnace*
* *Assign monitoring system state*
* *Pressure control routine*
  + *Keep track of start time at each valve setting*
  + *Select pressure valve status for next scan*
* *CO2 control routine*
  + *Check whether CO2 sampling is active*
  + *Keep track of start time at each valve setting*
  + *Select CO2 sampling valve and pump status for next scan*
* *Set valve and pump control ports to state required for following scan*
* *Record control*
  + *Determine whether to write a record based on accumulated data*
  + *Determine whether to write a record based on 1-sec (single scan) data*
  + *Call Accumulator, Resolve\_Accumulator, and Clear\_Accumulator functions as needed*
* *Diagnostic record control*
  + *Analogous to Record control*
* *Cleanup*
  + *Reset counters, timers as needed*

**Status, Mode and State**

Status is the off/on status of each burner at any time. Mode is the current combustion system operating condition, based on the status of the 1 or 2 burners during the current and prior scan, and cooldown timers. Water heater mode and furnace mode are each represented by a value of 1 to 5 (or 0 if the appliance is not present).

System state is a value controlling the monitoring system operation, with values of 1 to 6. Complete specification of system state would include a cooldown count and current valve settings, but those will be handled separately.

**Appliance modes (modes for 2 appliances are independent of one another)**

|  |  |  |  |
| --- | --- | --- | --- |
| **mode\_wh** | **mode\_f** | **Description** | **Error conditions** |
| 1 | 1 | Burner just started, was off in prior scan. This mode set actively by analysis of moving average temperature. | Burner indicated as OFF following a mode 1 |
| 2 | 2 | Burner on, moves to mode 2 by default after a mode 1, and stays in mode 2 until actively changed. |  |
| 3 | 3 | Burner just stopped, was in mode 2 in prior scan. This mode set actively by analysis of moving average temperature. | Burner indicated as ON following a mode 3 |
| 4 | 4 | Cooldown. This mode set automatically after a mode 3, invokes a timer to provide detailed data for a period after burner stops. Mode 4 ends at the top of the minute falling 2-3 minutes after burner stops. |  |
| 5 | 5 | Burner off |  |
| 0 | 0 | This appliance not present |  |

**System states**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **New mode\_wh** | **New mode\_f** | **System STATE**  **1 to 6** | **Record period (sec)** | **State Timer** | **Notes** |
|  |  |  |  |  |  |
| 1 | 0 1 3 4 5 | **1** | 1 |  | **START** 1st burner start. Takes priority over all other states except 2. |
| 0 1 3 4 5 | 1 |
| 2 | 0 1 2 3 4 5 | **2** | 1 |  | **ON** 2nd burner start doesn’t alter monitoring state. Takes priority over all other states. |
| 0 1 2 3 4 5 | 2 |
| 3 | 0 4 5 | **3** | 1 |  | **STOP** Both burners now off. Start state 4 timer. |
| 0 4 5 | 3 |
| 4 | 0 4 5 | **4** | 1 | 120 to 180 sec | **COOLDOWN** Use single timer, started when last burner turns off. End at top of minute. |
| 0 4 5 | 4 |
| 5 | 0 5 | **5** | 60 |  | **OFF-CO2** First minute of a 15 min interval. Includes CO2 measurement. |
| 0 5 | 5 |
| 5 | 0 5 | **6** | 60 |  | **OFF** Minutes 2-14 of 15 min interval. No CO2. |
| 0 5 | 5 |
| any | any | **7** | 1 | 120 to 240 sec | “Error” state triggered by various error conditions. Write data at 1-sec interval for a period of time (following same timeout rule as state 4). Not used. |
| any | any | **9** | 1 |  | Manually triggered “test” state for use during setup. Write data at 1-sec interval plus print selected data to laptop screen. Not used. |

**Variable Names**

See xls for data variables to be recorded

**Pseudocode**

\*\*\* Add

Control of 24V power via P8-15

Seconds of operation of each burner, and either burner

Make sure seconds of operation is set to 0 on burner off

Site ID, via MAC address?

**Constants**

*# Ctrl ports:*

*PORT\_PZERO = 1*

*PORT\_PWH = 2*

*PORT\_PF = 3*

*PORT\_PZONE = 4*

*PORT\_CO2WH = 5*

*PORT\_CO2FURN = 6*

*PORT\_CO2ROOM = 7*

*PORT\_CO2PUMP = 8*

PRESSVALVE\_CYCLE = 3 *# Start with 3 seconds for pressure valve cycling*

PRESSCLEARANCE\_TIME = 1 *# Pressure data settling time; start at 1*

CO2VALVE\_CYCLE = 20 *# Start with 20 seconds for CO2 valve cycling*

CO2CLEARANCE\_TIME = 10 *# CO2 system clearance time; start at 10*

**Config**

# Allow interactive input from laptop if practical, else set up an installer config file

# for field-determined settings

Is a wh being monitored?

Is a f being monitored?

Is a magnetic door sensor connected to xx1 ADC input?

Is a current sensor connected to xx2 ADC input?

Name for this current sensor

Is a current sensor connected to xx3 ADC input?

Name for this current sensor

Is a spare thermocouple connected to xx4 ADC input?

Name of this thermocouple

Is a spare thermocouple connected to xx5 ADC input?

Name of this thermocouple

Number of XBee inputs

Name XBee input 1

Name XBee input 2

Name XBee input 3

Name XBee input 4

**Initialize**

Set up I2C

Set up UARTs

Set up GPIO output ports

Set up other hardware, communications??

Query RTC, set OS time to RTC time

Attempt to query NTP server for accurate time update

\*\*\* Consider ntpd daemon approach

If successful, set OS time, set RTC time per NTP update

Read config file

ADC setup parameters (continuous conversion, SPS=8, gain=1)

Setup parameters from config file, or live inputs from screen

If no wh, whmode.status = 0 if yes, whmode.mode = 5 # Assume off at start

If no f, fmode.mode = 0, if yes, fmode.mode = 5 # Assume off at start

Set up ADCs

Use setup parameters from config file

Set sysstate.state = 6 *(start operation on the assumption both appliances are off)*

Set up background operations

NTP check (daily?)

r-sync (hourly?)

Set up scan timing loop

Timer

**Scan Loop**

**Wait for top of second**

Capture time *#(os time in integer form to 1-second resolution or equivalent)*

**Scan all inputs**

Read all active channels on ADCs

Capture individual input values.

*# Note we are not sending new data to an accumulator at this point, but will wait until after we decide what operating mode we need to apply, based on the data plus timers.*

*# CO2 values must be allocated to 1 of 3 parameters (representing 3 different physical locations) depending on valve.status (one bit will indicate which valve is active)*

*# Similarly, pressure values must be allocated to 1 of 4 parameters (representing 4 different physical locations) depending on valve.status. And, we will subtract the current value of zero offset (one of the 4 locations measured) from the other 3 values in every case.*

Convert data to engineering units

*# Convert new data to temperature, pressure, CO and CO2 concentration. Some values may be left as voltages.*

*# Pressures: Have to decide how to handle oversampling of pressures, maybe 25 values. Probably want to simply average them, but also capture the range (max – min) within each scan as a diagnostic check.*

*# Pressures: subtract the current value of p.zero.avg from p.whvent.avg, p.fvent.avg, and p.zone.avg within each scan. The range doesn’t depend on zero offset, doesn’t need adjustment.*

Copy flue temperatures into a cyclic (?) array

*#(This copy of flue temperatures is used to determine burner status. The array holds n values, probably ~10, is updated every scan, and NOT cleared when records are written. Used for calculation of moving average temperature, revised every scan. Comparison of the new flue temperature to the moving average is the basis for determining burner on or burner off status.)*

Monitor XBee UART, write new data to objects.

*#(Allow for as many as 4 data objects/arrays for XBee data, though 1 - 2 is likely. These objects should also accept multiple values, since we may receive multiple XBee data values within any one record interval, e.g. data coming in every 10 seconds, record written at 60 seconds.)*

# Not sure cnt\_seqential\_scan below is needed, have added a counter for number\_of\_scans in the Accumulator function

Increment cnt\_sequential\_scan (to be used as divisor in calculating averages over time)

**Determine burner status**

WH burner status

If current temperature < T\_WHFLUEMIN then set whmode.status = OFF

Calculate moving avg temp for prior n seconds *#(not including current value)*

Compare current value to moving average

If temperature is more than DT\_WHON vs moving avg then set whmode.status = ON

If temperature has decreased more than DT\_WHOFF vs moving avg then set whmode.status = OFF

*#(May add some additional logic to allow for startup when a wh or f is already on. i.e. if moving avg is > x and no decline is detected… e.g. if flue > 180 F and not falling. Similarly for burner off.)*

*#(add similar code for furnace)*

**Determine wh mode**

whmode.priormode = whmode.mode *# Capture prior mode for use in following scan*

# The next few lines establish priority order; exit function where indicated

If whmode.mode = 0 then

Exit function *# No wh present, nothing to do*

If whmode.priorstatus = OFF then *# Covers prior modes 4 5 6 (burner off)*

If whmode.status = ON then

whmode.mode = 1 *# wh just started*

time.whburnerstart = time *# Track burner run time for each burner*

Exit function *#(don’t consider any other options)*

If whmode.priormode = 1

If whmode.status = ON then

whmode.mode = 2 *# Normal transition to mode 2*

time.whburnerrun = (time - time.whburnerstart)

Else

whmode.mode = 4 *# Set to mode 4 since it was on, and to collect diagnostic data*

error.modes = b0001 *#(Error if we see burner go off immediately after startup)*

Exit function *#(don’t consider any other options)*

If whmode.priormode = 2

If whmode.status = OFF then

whmode.mode = 3 *# wh just stopped*

*# Not sure we need explicit stop time. We should write burner run time and cooldown*

*# time each second, and track system cooldown as a single parameter*

time.whburnerstop = time *# Capture time of burner stop*

Else

time.whburnerrun = (time - time.whburnerstart)

whmode.mode = 2 *# Stay in mode = 2 (this line not needed)*

Exit function *#(don’t consider any other options)*

If whmode.priormode = 3

If whmode.status = OFF then

whmode.mode = 4 *# Normal transition to cooldown*

time.whburnerrun = 0

Else

whmode.mode = 1 *# Unexpectedly back on*

error.modes = b0010 *#(Error if burner turns on immediately after going off)*

Exit function *#(don’t consider any other options)*

If whmode.priormode = 4

If whmode.status = OFF then

If ((time.whburnerstop – time) >= 120 sec AND time mod 60 = 0) *# Top of minute*

OR If (time.whburnerstop – time) >= 180 *# > 3 min, missed top of minute*

then

whmode.mode = 5 *# wh off and cooldown period expired*

Else

whmode.mode = 4 *# Stay in mode = 4 (for clarity, not required)*

*# End Determine wh mode*

**Determine f mode**

*# Furnace routine exactly analogous to wh routine above*

**Select state**

*# I think this allows for either wh or f mode to be 0 (not present) at all times*

If whmode.mode = 2 OR fmode.mode = 2 then # At least one burner already on

state.state = 2

Exit function

If whmode.mode = 1 OR fmode.mode = 1 then # First burner has just started

state.state = 1

Exit function

If whmode.mode 3 = OR fmode.mode = 3 then *# Last burner has stopped*

state.state = 3

time.lastburnerstop = time *# Time of last burner stopping; use for cooldown timer*

Exit function

If whmode.mode = 4 OR fmode.mode = 4 then

If ((time - time.lastburnerstop) >= 120 sec AND time mod 60 = 0) *# Top of minute*

OR If (time - time.lastburnerstop) >= 180 *# > 3 min, missed top of minute*

then

state.state = 6 *# wh off and cooldown period expired*

*# Sort out state 5 vs 6 below*

Else

state.state = 4 *# Stay in state = 4 (for clarity, not required)*

# Do not exit function here; need to continue to state 5 and 6 selection

If state.state = 6 then

If time mod 900 < 60 then *# If we’re in first 60 sec of a 15 min interval*

state.state = 5 *#(collect CO2 data)*

set flag\_start\_CO2 # Seem to need a flag to initiate CO2 operation?

Else

state.state = 6 *# Stay in state = 6 (for clarity, not required)*

# Do not exit function here

If state.state = 5 then

If time mod 900 >= 60 then *# We’re now beyond the 60 sec state 5 period)*

state.state = 6 *# Toggle back to state 6, no CO2 data*

**Pressure sampling control**

*# Written on the assumption that we use a fixed measurement period (e.g. 3 sec) at each valve (i.e. measured location), and that we will probably not use data from the first second at each location due to settling time of pressure in the system*

*# Like with CO2 sampling below, this routine must deal with the valves active in any given system. We’ll always have a zero-offset valve and a zone pressure valve, plus one or both of a wh vent valve and furnace vent valve, i.e. 3 or 4 pressure valves.*

*# Switch valve positions (i.e. move to next valve in order) at modulo 3-second intervals,*

*# or if more than 3 seconds passes*

*# Use a constant PRESS\_VALVECYCLE to establish cycling period, probably 3 sec*

If (time mod PRESS\_VALVECYCLE = 0) OR if ((time - time.pressvalveset) > PRESS\_VALVECYCLE) then

move to next valve position

# Use when accumulating data to determine whether valve settling time has passed

time.pressvalveset = time

**CO2 sampling control**

*# Switch valve positions (i.e. move to next valve in order) at 20-second intervals,*

*# or if more than 20 seconds passes*

*# Can execute every scan, I think (?), or bypass if CO2 data collection is not active*

*# Start time may be triggered by either burner start or by state=5, i.e. at any time;*

*# this valve cycling logic should work in either case*

*# If in one-sec data collection (burner on or cooldown), exclude room valve for first 5 min*

*# This routine needs modification – it assumes both appliances are present. Instead,*

*# probably should use a list established during initialization (will always include room,*

*# and will include either a wh, furnace, or both). Will always have either 2 or 3 CO2 valves.*

*# state 5 first iteration must trigger start of new timer; uses flag set in state controller –*

*# see* flag\_start\_CO2

*# Note we do want a burner timer: use to hold off CO2 room sampling for 5 minutes of burner time*

*# For this, need to create* time.lastburnerstart*, the most recent start time of an active burner*

*# Ctrl ports: 5 = PORT\_CO2WH, 6 = PORT\_CO2FURN, 7 = PORT\_CO2ROOM, 8 = PORT\_CO2PUMP*

if just starting CO2 sampling (per flag flag\_start\_CO2), get start time

*#(go to furn first, allows wh to take priority if it starts at same time)*

*# Start of valve routine when either burner starts, state = 1*

*# state 1 triggers start of new timer*

If state.state = 1

set valve.status(*PORT\_CO2PUMP*) #

If whmode.mode = 1 then *# If wh started, start sampling at furnace*

clear valve.status (*PORT\_CO2FURN*, *PORT\_CO2ROOM*)

set valve.status (*PORT\_CO2WH*)

Else *# Else start sampling at furn (incl if both start at same time)*

clear valve.status (*PORT\_CO2WH*, *PORT\_CO2ROOM*)

set valve.status (*PORT\_CO2FURN)*

time.co2valveset = time

*# Start of valve routine on first execution of state = 5, initialize timer and clear flag*

*# Start sampling at wh (final sample will be room, so maybe lowest values)*

If (state.state = 5 AND flag\_start\_CO2 = 1) then

time.co2valveset = time

clear flag\_start\_CO2

clear valve.status (*PORT\_CO2FURN*, *PORT\_CO2ROOM*)

set valve.status (*PORT\_CO2WH, PORT\_CO2PUMP*)

If in state (2, 3, 4, 5)

If (time - time.valveset) >= CO2VALVE\_CYCLE then *# Capture exact time interval and over-runs*

If valve.status *PORT\_CO2ROOM* = ON then

valve.status.clear ports *PORT\_CO2FURN and PORT\_CO2ROOM*

valve.status.set *PORT\_CO2WH* and *PORT\_CO2PUMP*

Else If valve.status *PORT\_CO2WH* = ON then

valve.status.clear *PORT\_CO2WH* *and PORT\_CO2ROOM*

valve.status.set *PORT\_CO2FURN* and *PORT\_CO2PUMP*

Else if valve.status *PORT\_CO2FURN* = ON then

If state.state = (1 or 2 or 3 or 4) AND

time - time.lastburnerstart < 600 seconds then # Skip CO2\_room

valve.status.clear *PORT\_CO2FURN and PORT\_CO2ROOM*

valve.status.set *PORT\_CO2WH* and *PORT\_CO2PUMP*

Else *#(if beyond 5 min of run time or no burner is operating)*

valve.status.clear *PORT\_CO2WH* *and PORT\_CO2FURN*

valve.status.set *PORT\_CO2ROOM* and *PORT\_CO2PUMP*

time.co2valveset = time

*# End of valve cycling function*

If state.state = 6 then *# Don’t do any CO2 sampling*

clear valve.status (*PORT\_CO2WH*, *PORT\_CO2FURN*, *PORT\_CO2ROOM, PORT\_CO2PUMP*)

*# Track accumulated pump operating time in hours, save to diagnostics file*

If valve.status.pump = ON, time.pump\_on on = time.pump\_on + 1/3600

**Set control ports**

Set ports # Always set ports per new value of valve.status

**Record control**

# There are 2 decisions to make about writing records:

# 1) Are we closing out a 60-sec record (which include data collected through

# the PRIOR scan). Do this FIRST, to keep records in chronological order.

# 2) Does current data get written as a 1-sec record?

# BOTH of these happen when the first burner turns on after an off period.

# Should we reset accumulator values to NaN every cycle when we’re not using it? This

# should not be necessary, as long as accumulator is cleared when moving out of state

# 5 or 6 and when a record is written

*# If system is in low-resolution data collection*

*# Write record at top of minute, or if it’s run over a minute*

*# Do this BEFORE writing 1-second record (if active) to maintain chronological order*

*# If we transition into state 5 or 6 at the top of a minute, there will be no data*

*# in the accumulators, and we can’t write a 60-sec record. Screen for*

*# this condition by checking that the prior state was a 5 or 6*

*# before calling Resolve Accumulators.*

***\*\*\* Might address this by using state.prior state as the entry point.***

***\*\*\* If PRIOR state was 5 or 6, and at top of minute, save 60-sec data***

***\*\*\* And then check for current state to trigger calling accumulator for current data***

# old version: If (state.state = 5 OR 6 AND state.priorstate = 5 OR 6 AND time mod 60 = 0) OR

If (state.priorstate = 5 OR 6 AND time mod 60 = 0) OR *# check carefully*

(time - time.lastrecord) >= 60 OR # May cause trouble when RTC adjusted

(state.state = 1 AND state.priorstate = 5 OR 6) # Close out long record using available data

then

Increment record number

Call Resolve\_Accumulators

Assemble data values in csv format

Write to file on disk Call Clear\_Accumulator

time.lastrecord = time

# Call Accumulator whenever current state is 5 or 6, i.e. accumulating data over time

**\*\*\*Revised: Add line to check for state 5 or 6 again:**

If (state.state = 5 OR 6)

Call Accumulator

*# When state = 1 we write both a 60-sec record and 1-sec record*

*# Don’t exclude data based on pressure or CO2 settling time during 1-sec*

*# data collection; can sort out unusable data later, and it may have diagnostic value*

*# Testmode forces 1-sec data collection for use in checking operation.*

*# Add testmode control somewhere (physical inputs, timeout)*

If (state.state = 1, 2, 3, 4) OR (error value > 0) OR (testmode = ON) then

Increment record number

number\_of\_scans = 1

Assemble string of data values in csv format

Write to file on disk

time.lastrecord = time

Call Clear\_Accumulator # Not sure this is needed

If testmode = ON then

send output to laptop screen

D**iagnostics record control**

*# Various diagnostics info*

*# Write once a day at ~ 8 AM GMT (28800 seconds after midnight)*

If (time mod 86400 = 28800) OR (time - time. lastrecorddiagnostic) >= 86400

Increment record number

Assemble string of data values in csv format

Write to file on disk

Call Clear\_Diagnostics\_Accumulator

time.lastrecorddiagnostic = time

Call Diagnostics\_Accumulator # Every scan

**Accumulator**

# Should be able to accept at least 60+ values

*#(We need a pointer or counter e.g. “cnt\_seqential\_scan” to track the “row” or index for storing the next data value. The object or array will be cleared whenever a data record is written, and the counter will be set back 0. Current plan is to write data records at either 1 sec or 60 sec intervals, depending on operating mode, so need at least 60 “rows” plus padding for duplicate time stamps when clock is adjusted, and maybe best to allow a much larger number in case of operating anomalies.)*

If (time - time.pressvalveset) < CO2CLEARANCE\_TIME then *# Data assumed no good*

set current CO2 value to NaN

If (time - time.co2valveset) < PRESSCLEARANCE\_TIME then *# Data assumed no good*

set current pressure value to NaN

Write data (engineering values for each parameter) to object/array

number\_of\_scans = number\_of\_scans + 1

**Diagnostics\_Accumulator**

# Similar to Accumulator

# Some data will always be sent to a diagnostics accumulator, e.g. pump operating time

**Resolve\_Accumulators**

Calculate average values of data as called for in data table *#(see xls)*

*# (Averages should use the count of values in each object/table as the divisor, since it could vary from the nominal number of seconds included in the record)*

Find min and max values in data table

Use the number of observations in each parameter (i.e. not counting NaN) to calc averages

Find number of scans in record (might not be the same as every individual value?)

Sum binary error bits to form a single (hex?) value to be saved

**Resolve\_Diagnositcs\_Accumulators**

Similar to Resolve\_Accumulators

**Clear\_Accumulator**

Set all array values to NaN (clean out arrays before starting next scan)

**Clear\_Diagnostics\_Accumulator**

Set all array values to NaN (clean out arrays before starting next scan)

**Cleanup**

whmode.priormode = whmode.mode # Capture new prior mode

fmode.priormode = fmode.mode

state.priorstate = state.state

set all incoming data parameters to NaN ?

