Data Reporter Sketch Notes:

The DataReporter application consists of a set of tasks and supporting source files. The tasks making up the DataReporter are:

* Monitoring the rainfall.
* Monitor ambient temperature.
* Manage Li-ion battery charging.
* Minimize the power requirement.
* Report data to a cloud based database via GSM modem.

Tasks are mechanized using state machines driven by an Executive Task Scheduler in the Arduino loop() function. Each task performs some essential process of the DataReporter application such as:

* Monitor
  + Sleep/Wakeup.
  + Rain gauge.
  + Monitor battery.
* LogData
  + Time tag data.
  + Record data points.
* ReportData
  + Send data to cloud database.
* ManageResources
  + Queue resource requests.
* LogMessages

**DataReporter Sketch Source Files:**

DataReporter\_REV\_1\_1.ino

Has two functions.

void setup()

Enter:

Debugging.

Any time power is applied.

Only a single call to the SetupTask()

All hardware and software setup is done in the SetupTask().

void loop()

Contains only the scheduler.

Definitions.h

Application wide global definitions only.

Global.ino

All global variables.

All enums.

All structure definitions.

Monitor.ino

This is the main DataReporter task.

Scheduled at all times.

Discovers and reports all datapoints.

Sleeps the processor at all possible times.

Setup.ino

All required hardware and software initialization is done in this task.

LogData.ino

Logs data points to an SD file.

Determines when to report data points to the database.

ReportData.ino

Reports the data points in the SD file to the database.

ResourceQueues.ino

The resource were revised to use pointers instead of indexes.

Queues were moved from Global to the resource services source files.

Must revise all queue coding.

Must revise the SystemLogTask()

The other queues are not used.

Rtc.ino

The RTC is a shared resource.

SystemLogTask.ino

A

**Adding/Scheduling a Task**

A task is a state machine mechanized with a switch() statement that in turn is driven by the Executive Task Scheduler in the Arduino loop() function. A template of a task source file is shown in TaskTemplate.ino Every task is required to have an identifying 2 or 3 character prefix for all local definitions and local (.ino file) globals.

Add a task as follows:

* Add a MyNewTask.ino source file(Tab).
* Add template code to MyNewTask.ino file using TaskTemplate.ino template code.
* Replace TEMPLATE\_TASK with YOUR\_TASK\_NO.
* Replace TaskTemTask with YourTask.
* Add an index that identifies the new task added i.e.
  + #define YOUR\_TASK\_NO n in Definitions.h
* Initialize the new task entries in Setup.ino;
  + taskPointer[YOUR\_TASK\_NO] = NewTask;
  + taskScheduled[YOUR\_TASK\_NO] = false;
  + tasksState[YOUR\_TASK\_NO] = TASK\_INIT\_STATE;
* Add and code the states, for the new task, as required.

Schedule a task as follows:

* Check taskScheduled[YOUR\_TASK\_NO] with interrupts off.
* Set the entry state newTaskState[YOUR\_TASK\_NO].
* Set taskScheduled[YOUR\_TASK\_NO] true.
* Task is now running.

To end the task:

* Set taskScheduled[YOUR\_TASK\_NO] false.

**Shared resources:**

Shared resources, e.g. SD, system log messages, log data point, etc. are controlled and communicated with using FIFO queues. Each queue, e.g. the “Xyz” queue has the following global variables/functions:

* XyzQueueType \*XyzQueue[MAX\_XYZ\_QUEUE\_ITEMS];
  + Where XyzQueueType is the control structure used to control and communicate with the Xyz resource.
  + XyzQueue is an array of pointers to the task’s structures currently requesting he resource’s services.
* int myRequestIndex XyzPush(XyzQueueType myRequestStructure);
  + Copies the caller’s structure pointer onto the Xyz’s FIFO queue.
  + Schedules the Xyz task if it not currently scheduled.
* XyzPop();
  + Removes the structure pointer at the top of the queue.
* int XyzQueueCount;
  + The number of items in the queue.
* int XyzQueueOutIdx;
  + The index of the next item to be removed from the queue.
* xyzQueueInIdx
  + The index of the next queue position available for request queue insertion.

To use a shared resource:

1. Declare the associated control structure for the resource.
2. Fill out the resource control structure.
3. Turn the interrupts off.
4. Call the associated push method for the resource, with a pointer to your control structure.
5. Turn the interrupts on.
6. Monitor the status in your control structure to see if the resource has completed the task.
7. Process any status/results in the control structure.

**System Log**

The system log is a shared resource and is intended to allow tasks to record diagnostic or error messages to the Syslog.txt file on the SD card. The error/diagnostic messages, that are stored as character arrays in program memory, are prepended with a time stamp by the System log task, up to three parameters appended and the composed message is written to the Syslog.txt file.

To create a system message:

* In the SystemLogTask.ino source file:
  + Add the char array, containing the text of the message at the end of the MESSAGE\_TABLE.
    - prog\_char setupString\_0[] PROGMEM = "My Error Message.\0";
    - Note that the char array uses “\_n” to insure uniqueness nd must end with a zero byte (“\0” ).
  + Add a pointer to the message, that you added above, as the next iten in the sysLogPointers [] array. Don't forget the comma on the preceding pointer.
* In the Definitions.h source file:
  + Define n, the index to your message pointer, in the sysLogPointers[].
  + #define (the task’s identifying 2 or 3 character prefix)…. n.
    - #define TT\_TEST\_MSG\_IDX 2

To report a diagnostic/error:

* Declare your sysLogControl control structure.
  + sysLogControl mySysLogCtlStruct…;
* Initialize mySysLogCtlStruct.
* Turn the interrupts off.
* Push the control structure onto the FIFO (sysLogQueue[] in Global.ino).
  + PushSysLog(&mySysLogCtlStruct)
* Turn the interrupts on.
* Monitor the status in to see if the logging has completed.

**Monitor**

The Monitor task:

* Always scheduled.
* Sleeps/wakes the maximum sleep period.
* Mechanizes the noSleepFlags global control.
  + A set of 16 flags one for each possible task driven by the Executive Task Scheduler.
  + The task’s bit flag position is identified by it’s Task Number.
  + Used by individual tasks to indicate to the Monitor task to hold off sleep/wake cycles until the task has completed some processing task.
  + SetNoSleepTaskFlag(int taskNumber);
  + ResetNoSleepTaskFlag(int taskNumber);
* Monitors the rain bucket interrupt.
* Queues data points for the LogData task.
* Monitors the battery status.
  + Maintains the following measurement variables.
    - TBD

**Log Data**

The Log Data task:

* This task is a shared resource with a queue.
* The control structure:
  + TBD.
* Time tags the data.
* Writes the data point to the monthly history file.
* Flushes all history data file writes.
* Writes the data point to the DataQ\_n.txt file.
  + Flip-flops two data queueing files, "DataQ\_1.txt" and "DataQ\_2.txt".
  + Data queueing files allow the Data Reporter task to have a static view of the data points to be reported.
  + The ReportData task locks the current DataQ\_n file during the data reporting time.
  + Interrupts off during lock flag read/writes.
  + Once a data queueing files is locked it may be assumed closed.
  + Counts the data points written to the current data queueing file.
  + Data points ad records separated by line feed and carriage return characters.
* Flushes all data queueing file writes.
* Schedules the Report Data task if it is not already scheduled.
* Provides a “force” flag in the data point that he data reporter can see but is not placed in the database,
* The global bool dataQueueing\_1
  + Controlled by DataReporter task.
  + True indicates data points are being written to the DataQ\_1.txt file as opposed to the DataQ\_2.txt file.

**Report Data**

The Report Data task:

* Determines if a report to the database is required.
  + If debugging.
    - Dip switches?
  + If Battery power is good.
    - If its been > MAX\_REPORT\_INTERVAL minutes.
    - Its raining.
    - The force flag in set.???
* Controls the data queueing files.
  + dataQueueing\_1
  + Interrupts off.

**Resource Queues**

* Independent FIFO queue for each resource.
* Tracking indexes with pointers?

**Real Time Clock**

* A queued resource.
* Update my RTC location.
  + String
  + long
  + Unix
* This task is a shared resource with a queue.
* The control structure:
  + TBD.

**Setup**

* Opens two data logging File objects:
  + Open the Arduino IDE Serial Monitor communication.
    - Uses UART 0 at 9600
  + Initialize Solar 5.0 VDC regulator as enabled.
    - Digital I/O SOLAR\_REG\_ENABLE as output.
  + Enable the solar regular output to the battery charger.
  + Initialize Battery Charger as disable.
    - Set the bit state of Digital I/O BATTERY\_CHARGE\_DISABLE as disabeled.
    - Digital I/O BATTERY\_CHARGE\_DISABLE as output.
    - Disable the Battery Charger.
  + Initialize the SD card’s SPI bus Slave Select (SS) line.
    - Set the bit state of Digital I/O SD\_SPI\_SELECT as disabeled.
    - Digital I/O SD\_SPI\_SELECT as output.
    - Unselect the SD card.
  + Initialize the DS3234 RTC card’s SPI bus Slave Select (SS) line as not selected.
    - Set the bit state of Digital I/O RTC\_SPI\_SELECT as disabeled.
    - Digital I/O RTC\_SPI\_SELECT as output.
    - Unselect the DS3234 RTC card.
  + Show all tasks as not scheduled. Initialize all task pointers.
  + Initialize the SD card.
    - Set pin mode for Digital I/O 53 to output.
      * A requirement of the SD Library???
    - Open the System Log file.
      * Syslog.txt
      * FILE\_WRITE
    - Create the history file name for this month.
      * Hist(mm)(dd).txt
    - Open the history file for this month:
      * FILE\_WRITE
  + Open the data point queueing files.
    - DATA\_HOLDING\_FILE\_1
      * FILE\_WRITE
    - DATA\_HOLDING\_FILE\_2
      * FILE\_WRITE
  + Initialize dataQueueing\_1.
  + Initialize the battery voltage averager.
  + Schedule the Monitor task.

**SD Read-Write**

* The SD card is a shared resource and requires a control structure and associated queue.
* This task is a shared resource with a queue.
* The control structure:
  + TBD.
  + Char\*
  + File bject
* Functions:
  + Read.
  + Write-Flush.

**Battery Monitoring**

* In order for the modem to operate properly:
  + The battery must be above at or above a TBD level.
  + The difference between battery okay and battery not okay is measured in mV’s.
    - The ATmega2560 requires special design considerations to measure this differential.
    - A very stable VREF is required.
* There are two cases for battery monitoring.
  + The DataReporter does the battery management.
    - Current hardware.
    - Depends on obtaining a stable VREF for the ATmega2560.
    - Must use external reference.
    - Must use noise minimazaion. In measurement.
    - Must use running averager.
      * Mechanized by the Monitor task.
      * N deep averager.
      * Initialized.
        + 5 deep average.
        + Done insetup.
  + An IC is used as a battery management.
    - Uses TBD signals from the battery management IC.
    - The battery good status may not be sufficient.
      * Battery voltage monitoring still required.