Figure 1 is a block diagram depicting the basic operation of the ME (Mega Executive). The ME is intended as an extremely rudimentary thread or task execution environment designed primarily for a Li-Ion battery power MPU (Sparkfun Mega Pro) system with solar power replenishment.

The basic operation begins immediately after “RESET”ing the MPU and the execution of the Arduino IDE’s required setup() function in the DataReporter\_REV\_n\_n.ino source file.

The thread execution is mechanized by the code in the loop() function in the DataReporter\_REV\_n\_n.ino source file. The “for” statement block in loop(), represented by the roterary switch in the Figure 2, executes thereafter at all times that the MPU is not “sleeping” (Power Management and SleepModes).

The figure is intended to show that all threads, a.k.a. tasks or state machines, execute as long as their respective series switch is closed and the MPU is not sleeping. Note that the MPU Is normally in the Power-Down mode waking up to full operation and at least one pass through the “for” block once each 8 seconds.

The series switch is mechanized as a global array of boolean flags. True equals call-the-thread function and false equals don’t-call-the-thread function.

The roterary switch is mechanized as an array of pointers to the thread functions. The threads are mechanized as a function having no arguments, returning a void and containing a switch statement whose argument is the current executing thread state of the thread. The thread state is held in a global array of integers the state is generally controlled by the associated state machine itself. Note that due to the global scope of the execuitive variables any thread may schedule and monitor any other thread.

Global, as used here, is the address space shared by the executive and all threads. All variables belonging exclusively to the threads, and referenced only by other states of the same thread are defined as static in the thread function.

To use the ME in an embedded application it is intended that you use the core set of threads and functions augmented by your specialized threads and functions added to taylor the ME to mechanize your application’s unique requirements. Some of the core threads may also have to be changed to provide further tayloring for your application’s unique requirements.

The core threads and functions are:

* setup()
* loop()
* Console
* Monitor
* Analog
* ResourceQueues
* Real Time Clock
* SD card
* Setup()
* SystemLog
* LogData

The Console thread is provided as a development tool to allow the Arduino’s IDE’s Serial Monitor to interface the ME via the USB-RS232 FTDE cable. The Console thread allows serial input from the IDE’s Serial Monitor to be parsed and interact with the ME threads, e.g. it is possible to set the RTC through the IDE’s Serial Monitor. See the Console thread’s existing command parser for examples of interacting with the ME through the IDE’s Serial Monitor.

The Monitor thread must be unconditionally scheduled at all times. The Monitor thread’s primary functions are:

* Maintain the MPU in the lowest possible current draw mode.
  + Sleep only if all threads indicate “sleep is don’t care for this associated thread”.
    - Mechanized as a global integer/long containing 16/32 bit flags.
* Wake up for any of the following interrupts:
  + Bucket tipped.
    - #define EXT\_INTERRUPT\_0 0
    - If bucket tip signal is LOW.
    - The MPU’s External Interrupt 4, Port E Pin 4 is wired to the MPU's pin marked PWM 2. This is the Arduino's Digital Pin 2 and is also Arduino's External Interrupt 0. This is the DataReporter's Bucket Tipped interrupt.
  + Watchdog timer. (≈ 9 secs.)
  + IDE console-input-character-to-be-received interrupt.
    - * EXT\_INTERRUPT\_1 1
      * The MPU’s External Interrupt 5, Port E Pin 5 is wired to the MPU's pin marked PWM 3. This is the Arduino's Digital Pin 3 and is also Arduino's External Interrupt 1. This external interrupt is tied to the MPU's pin marked RX0<-0. RX0 is the Arduino IDE's serial transmit signal.
* Schedule other threads as required when the MPU wakes up.

There are 4 Arudino A/D channels as follows:

* #define SOLAR\_OUTPUT\_MONITOR 0
  + This is the output of the solar panel.
* #define BATTERY\_MONITOR 1
  + This is the output of the Li-Ion battery
* #define RFEGULATOR\_MONITOR 2
  + This is the output of the 5 VDC Buck Regulator (KA278R05)
* #define LOAD\_MONITOR 3

Thia is the output of the Li-Ion Battery Charger Power Manager (MCP73871)

The floating point scale factors associated with the above channels are as follows:

* #define SOLAR\_SF 0.028758950
* #define BATTERY\_SF 0.004181460
* #define REGULATOR\_SF 0.006257822
* #define LOAD\_SF 0.006144431

The following ME Resources are shared by other threads:

* System Log.
  + Interrupts off.
  + PushSysLog()
  + Interrupts on.
* SD Read/Writes.
  + Interrupts off.
  + PushSdrw()
  + Interrupts on.
  + All file objects are created at setup time.
  + Only read/writes to existing file objects are allowed.
* RTC Real Time Clock.
  + Interrupts off.
  + PushRtc()
  + Interrupts on.
* GPRS Modem.
  + Interrupts off.
  + PushGprs()
  + Interrupts on.

The shared resources are controlled and communicated with by the client threads through communiction and control structures unique for each shared resource. The client fills in the thread’s control structure and pushes a pointer to the structure onto the shared resource’s FIFO. This queues the clients request and schedules the associated thread if it is not already scheduled. The client then monitors the the shared resource’s status in the structure. See the associated structure definitions for detains.

The GPRS modem is a shared resource. Client tasks queue for the use of the modem the same as any other ME shared resource. The GPRS operations task (GprsOperations()) functions to turn power onto the modem, register with the GSM network, connect to a internet server and safetly turn power off the modem. Data interchanges with the server are handled directly by a client task once it has established ownership of the modem resource. The steps for capturing and using the modem are as follows:

Initialize your modemControl structure:

gprsStatus🡨 GPRS\_POWER\_OFF.

clientStatus🡨 CLIENT\_WAIT.

Interrupts off.

Push your modemControl structure.

Interrupts on.

Wait for gprsStatus to indicate GPRS\_DATA\_EXCHANGE or GPRS\_OPERATION\_ERROR.

If GPRS\_DATA\_EXCHANGE then

Use the Serial1 objet to communicate with the indicated server.

When the server interchange is complete set the clientStatus to CLIENT\_DONE.

The GprsOperation task will:

Close the internet connection.

Power down the modem.

Check the queue for the next client.

or GPRS\_OPERATION\_ERROR

The GprsOperation has reported the error in the System log and waited for the system log to be written.

GPRS\_WAIT\_AFTER\_ERR

TASK\_INIT\_STATE🡨 No power down then check queue.

Disconnected from the internet.

Powered down the modem.

The client should try later.

Do not issue the +++ (Leave the Data Mode) AT command or use the 3 character “+++” sequence in any data sent to the server. The maximum time that the modem may be captured by any ME threads is 15 minutes.

Data point buffering d

Log Data:

Adding a thread:

**A/D Monitor (SOLAR\_OUTPUT\_MONITOR)**

**Digital Output (SOLAR\_REG\_ENABLE)**

SD Card (BOB-1140)

5VDC Buck Regulator (KA278R05)

Solar Panel

Li-Ion Battery Charger Power Manager (MCP73871)

Li-Ion Battery

2.5 VDC Voltage Reference (REF192)

5 VDC Boost-Buck Regulator (S7V7F5)

3.3 VDC Buck Regulator (REG710)

**Solar Panel**

**Charge**

**Load**

**A/D Monitor (RFEGULATOR\_MONITOR)**

**A/D Monitor (BATTERY\_MONITOR)**

**A/D Monitor (LOAD\_MONITOR)**

**Three Digital Inputs (BATTERY\_MNGR\_PGNOT BATTERY\_MNGR\_STAT2BATTBATTERY\_MNGR\_STAT1\_LBO)**

**AREF**

Cellular Shield

DataReporter Controller Arduino Mega PRO (ATmega2560)

**Arduino IDE Interface Sparkfun (DEV-09873)**

**External Interrupt (CONSOLE\_ACTIVE)**

**External Interrupt (BUCKET\_TIP)**

**Figure 1**

**DataReporter Hardware Block Diagram**

Real Time Clock (DS3234)

**SPI Bus**

**AREF**

**Antenna**

**RS232**

5 VDC Boost-Buck Regulator (S7V7F5)

**ATmega2560 A/D Channel Voltage Monitoring Point**

**Power (VCC)**

**ATmega2560 External Interrupt**

**ATmega2560 General Purpose**

**Digital Input**

**ATmega2560 General Purpose I/O**

**Figure 2**

**DataReporter Hardware Block Diagram**

Processor Control

State 1

State 2

State n

Thread 1

State 1

State 2

State n

Thread 2

State 1

State 2

State n

Thread n