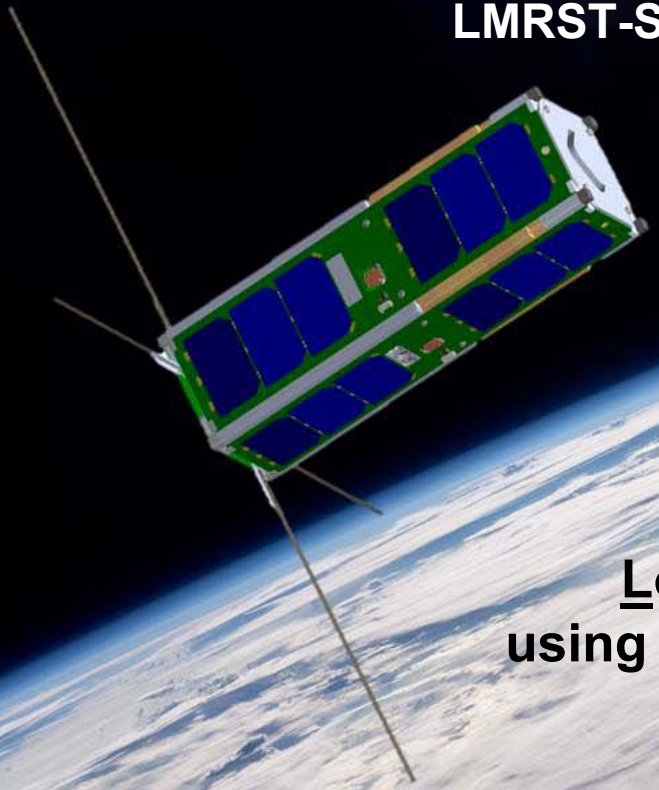
LMRST-Sat Flight Software Debugging & Mission Assurance

Andrew E. Kalman <aek@stanford.edu>

LMRST-Sat is a joint SSDL-JPL 3U CubeSat mission



LMRST-Sat's purpose: Flight-test the JPL Low-Mass Radio Science Transponder (LMRST) using a CubeSat and the Deep Space Network (DSN)

Host hardware: 12MHz 16-bit TI MSP430F2618 (116KB Flash, 8KB RAM)

Host software: SSDL FSW w/Salvo RTOS & EFFS-THIN SD Card driver

Nonvolatile storage: 2GB SD Card, FAT16 format

Communications downlink: >=9600bps

Development Timeline

FSW 1.x (2008-2010): Developed by SSDL students to be as non-autonomous as possible … all on-orbit activities were to be command-driven (real-time or uploaded for future execution), except for automated antenna deployment. *This substantially simplifies FSW design, as it places much of the burden of correct satellite operation on the operator.*

Additional core concepts of SSDL FSW, including Telemetry Capture, Storage and Retrieval System (TECTSARS), TAP/RAP/CAP packet schemes, and Vizon (I) ground station software, were developed in this phase.

FSW 2.x (2011-2014): The second phase of LMRST-Sat's FSW development @ SSDL, with a strong emphasis on code cleanup, additional MA-related features (e.g., external WDT), GPS integration, and long-term testing. A parallel ground station software development (Vizon II) began during this phase, and LMRST-Sat was "stretched" to 3U with a GPS receiver added.

FSW 3.x (2015): Release version (to work with Vizon I ground station), with changes incl. those suggested by JPL during acceptance testing.

Debug Tools Used

These tools proved particularly useful when debugging the LMRST-Sat FSW:

- The LMRST-Sat debug output terminal: With 10ms-resolution timestamps and printf() formatting support, this proved to be a great window into LMRST-Sat FSW internal operations. It proved especially useful in debugging startup / reset / restart issues, as it provides a window into the startup sequence as soon as MSP430 interrupts are enabled.

- Total Phase® Beagle I2C/SPI protocol analyzer: This analyzer provides high-level, packet-based filterable snooping of I2C or SPI transactions. LMRST-Sat utilizes an SD Card (SPI) extensively, and its RTC, GPS, EPS, battery and secondary HSS are all on the I2C bus.

- Desktop power supplies with digital displays: Having fast, digital outputs of the currents drawn by LMRST-Sat help verify that the system's major power consumers are working properly.

- A good oscilloscope: Invaluable for checking timing (e.g., accuracy of us-resolution delay functions), as well as characterizing analog behavior of digital signals.

- MCU programmer-debugger: Essential when stepping through FSW, watching variables, etc.

Code Cleanup (FSW 2.x)

TECSTARS concept (telemetry packets in multiple channels can be broadcast, stored and/or retrieved on command, with unlimited storage to SD Card) was already in place in v1.x. However, the code itself was poorly documented, without useful comments, and the limitations of the system were not clearly stated, leading to long software development "stalls" while SSDL attempted to add new features.

If it isn't documented in a clear, concise and comprehensive manner that others can understand, it might as well not exist.

Therefore a major effort of the v2.x coding effort was software cleanup … in particular, the FSW adopted:

- A standardized module format (*.c, *.h) for each TAP carton and TAP task
- Standardized naming conventions for all symbols
- The use of functional interfaces (instead of global variables) wherever possible / practical
- A common function(al) layout for all TAP tasks (the majority of tasks in the FSW)

These changes paid huge dividends, as the FSW became much easier to understand, debug, enhance/extend, and test.

Dedicated Debug Terminal (FSW 2.x)

LMRST-Sat's MSP430 has only two hardware UARTs. A **software** (Tx-only) **UART** running at 9600bps and occupying a single GPIO pin tied to one of the MSP430's spare output compare functions outputs verbose debug information.

An always-enabled (output) debug terminal with 10ms timestamps is extremely helpful in debugging and tracking various run-time problems. You can't fix what you can't understand. Timely use of printf() can yield insight into your problem(s).

The debug output was further enhanced to tag each message as informational, a warning or an error, with the ability to select any combination thereof for output (e.g., show only warnings and errors during a 72-hour test). The timestamp is in **dd:hh:mm:ss.tt** format.

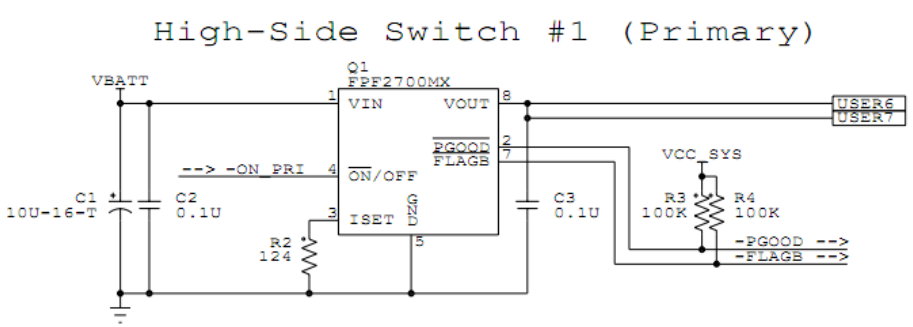
```
00:00:00:02.05 msg: time_set_time: Set time and date.
00:00:00:02.11 msg: mailman_init: Starting...
00:00:00:03.23 msg: mailman_init: Counting files in INIT folder ...
00:00:00:06.17 msg: mailman_init: Counted 545 INIT files.
00:00:00:06.18 msg: mailman_init: Counting files in OPS folder ...
00:00:00:08.48 msg: mailman_init: Counted 132 OPS files.
00:00:00:08.48 msg: mailman_init: Done.
00:00:00:08.48 msg: task_startup: Done.
```

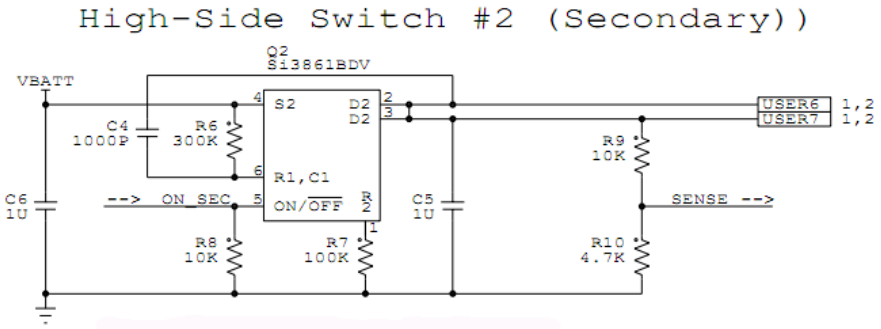
Careful coding of the software-timer-based Tx-only UART resulted in a 100% reliable debug terminal through all of LMRST-Sat's FSW operation. It was decided to leave it in the release code, as it was beneficial for start-up logging, etc.

Independent High-Side Switches (FSW 2.x)

The primary function of the LMRST-Sat bus is to enable the LMRST payload, and measure the associated analog voltage channels while LMRST is ON.

For LMRST-Sat we chose two high-side switches (HSS) from different manufacturers, with independent and functionally different means of activating each HSS. The primary HSS is driven via a PWM output from the MCU, with gentle/ramped turn-on and turn-off over a few seconds. The secondary HSS is driven via an external I2C latch and responds to direct [3.3V,0V] binary inputs.



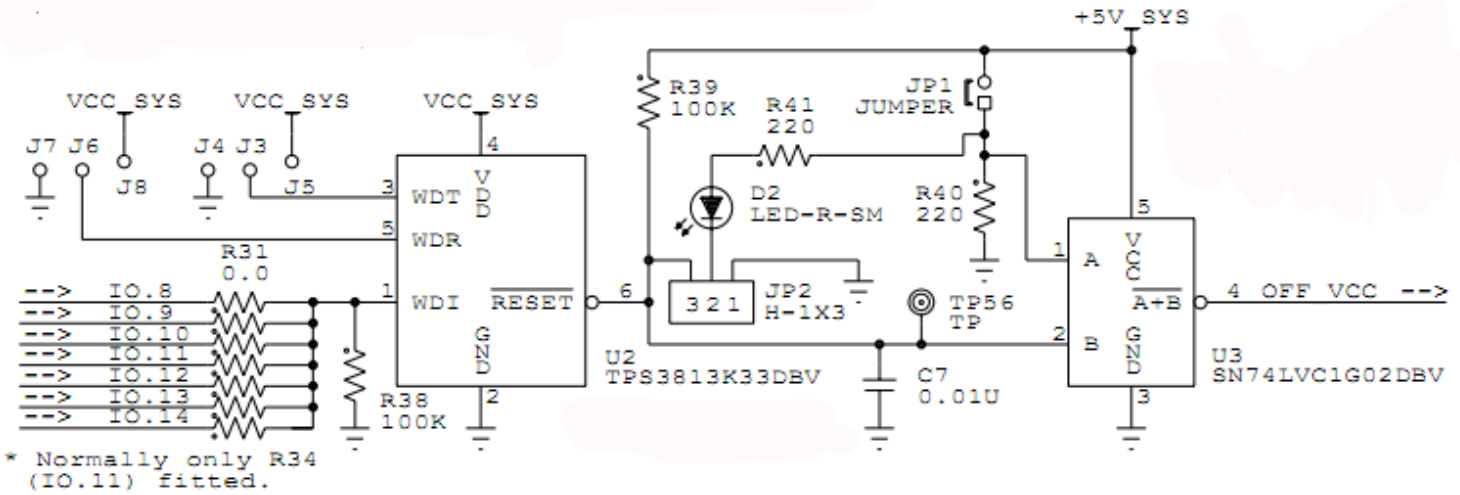


The FSW code (and commands) to control each HSS are completely independent and are contained in separate functions. Note that even the polarities for the HSS control inputs are different! LMRST-Sat appears to be happy using either method to control LMRST.

External Watchdog Timer (FSW 2.x)

While the MSP430 has an internal WDT peripheral, we observed cases in v2.x testing where the system locked up completely and only a full power cycling restored operation. Therefore an external WDT was added to the system, which drove the **OFF_VCC** signal (more far-reaching than just driving **–RESET**).

A reliable external WDT is a means of last resort to recover from coding errors, subsystem failures and/or external events – all unpredictable events. You'll be glad you have one, even if you see it trigger occasionally.



You must be able to temporarily disable the external WDT (for reprogramming the MCU, among other things); use an accessible RBF jumper!

Software WDT Scheme (FSW 2.x)

An external WDT must be “kicked” with specified timing to keep the system alive and free from a WDT-induced reset/restart. SSDL students developed a scheme that monitored each task in the multitasking operation, and kicked the external WDT as long as all of the tasks were exhibiting nominal behavior. When the WDT task detected out-of-nominal behavior, it suppressed the kick, resulting in (effectively) a system-wide power-on reset of all devices controlled by `OFF_VCC` within 1.25s.

Implementing reliable and comprehensive WDT-kicking software is a non-trivial undertaking. For LMRST-Sat, the WDT task’s “view” of each task is highly configurable (at compile time and in runtime). The API back to the WDT task was made as simple as possible, with the WDT task responsible for runtime bounds-checking of task-specific counters.

The WDT task maintained the expected upper and lower limits of a tagging counter for each task … each task was responsible for updating its counter periodically.

```
WDT_set_task_bound_lower(TASK_ID_BLINK, WDT_BOUND_LOWER_TASK_ID_BLINK);
WDT_set_task_bound_upper(TASK_ID_BLINK, WDT_BOUND_UPPER_TASK_ID_BLINK);

while(1) {
  P1OUT ^= BIT0;
  OS_DelayTS(blink_get_period());
  WDT_inc_counter(TASK_ID_BLINK);
} /* while */
```

Loop Terminators (FSW 2.x)

It was found that a variety of LMRST-Sat functions included loops with **conditional** terminators. Getting stuck in one of these loops was not uncommon, would reflect a system failure, and would normally trigger a WDT-based restart. We felt that rather than relying exclusively on the WDT, we would limit the number of tries in each loop to satisfy the condition, and upon exiting the loop, would test to see if the loop exited normally or timed out.

Every loop construct that the FSW can get stuck in must have an independent terminating condition. Upon exit, check if the loop terminated normally or abnormally.

As an example, a fault on the I2C bus would cause the previous implementation of this “Is the I2C bus busy?” check to loop indefinitely while waiting for the I2C bus to clear. If the code in the final FSW version terminates after a maximum number of attempts, this indicates a serious problem on the I2C bus, but the overall FSW can continue operating.

```
for(count=0; count<MAX_TRIES; count++) {
   if((UCB1STAT&UCBBUSY)!=UCBBUSY) break;
 }
```

I2C Bus Reset – The Problem (FSW 2.x)

In v2.x LMRST-Sat long term-FSW testing, pre-WDT, the following was observed:

- Only a power cycling could recover FSW operations
- It always hung inside an I2C read or write function, waiting for the bus to be not busy
- It always hung with the I2C bus' SCL HIGH and SCL LOW
- It never hung when the GPS module was not present
- (Almost?) all hangs happened when the GPS module was in the stack
- The only large (>10 bytes) I2C transactions were between the MSP430 and the GPS
- The MSP430F2618 has many I2C-related errata. We seemed to have addressed the relevant ones in our I2C driver code …

All of these observations pointed to a fault in the GPS module's I2C implementation (software or hardware) …

So:

- We tried various changes to the LMRST-Sat FSW's I2C driver code – no change
- We removed the PCA9515A I2C isolator between the GPS module and the MCU – no change
- We had the GPS module's manufacturer torture-test (with the same I2C commands and responses) the GPS module with a PIC24 host and a Linux host over several days – **no problems**

We were surprised to find that the fault lay in the MSP430F2618 and/or our FSW code, not in the GPS module … what to do?

I2C Bus Reset – The Solution (FSW 2.x)

Application Note AN686 from Analog Devices [1] perfectly described our situation, and proposed a “bit-banged” solution in pseudocode. Only the Master can cause a well-managed I2C reset, so it was up to the LMRST-Sat’s FSW I2C code to detect and implement this I2C reset.

The FSW already detected the faults (via the loop terminators discussed previously). Now it was time to implement the fix, for our MSP430F2618-based system:

```
void I2C_reset_bus(void) {
  unsigned int i;


  UCB1CTL1  =  UCSWRST;                    // All STOP I2C
  UCB1I2CIE =  0;                          // Disable all I2C interrupts
  UC1IFG   &= ~(UCB1TXIFG+UCB1RXIFG);      // Clear all interrupt flags?
  UCB1STAT &= ~UCNACKIFG;                  //  ""

  P5OUT    |=   BIT2 + BIT1;              // Prep SCL & SDA to be inactive / high
  P5DIR    |=   BIT2;                     // Prep SCL to be output
  P5DIR    &=  ~BIT1;                     //  ""
  P5SEL    &= ~(BIT2 + BIT1);             // Grab the two I2C pins for SPI. Now
                                          //  they are GPIO

  for(i=0; i<9; i++) {                    // Can't need to send more than 9 bits ...
    P5OUT &= ~BIT2;                       // Toggle SCL until the Slave
    time_delay_us(20);                    //  (hopefully) releases SDA ...
    P5OUT |=  BIT2;                       //  ""
    time_delay_us(20);                    //  ""
    if((P5IN & BIT1) ==BIT1) {
      break;
    }
  }

  user_debug_msg(DBG_ERR, "I2C_reset_bus" ": Attempted to reset I2C bus.");
}
```

I2C Bus Reset – It Works! (FSW 2.x)

With the bit-banged I2C reset code in place, the system automatically detects a hung I2C bus, and resets it cleanly:

```
00:00:46:03.03 WRN: I2C_write @51h: curr(>500)=5804, peak= 695 counts.
00:00:46:06.98 WRN: I2C_read  @51h: curr(>500)=3682, peak=   0 counts.
00:00:46:28.58 WRN: I2C_write @51h: curr(>500)=5805, peak=5804 counts.
00:00:48:11.55 WRN: I2C_write @51h: curr(>500)=5808, peak=5805 counts.
00:00:49:32.58 WRN: I2C_read  @51h: curr(>500)=9999, peak=3682 counts.
00:00:49:32.59 ERR: I2C_reset_bus: Attempted to reset I2C bus.
00:00:49:57.97 WRN: I2C_read  @51h: curr(>500)=3681, peak=   0 counts.
00:00:52:32.03 WRN: I2C_read  @51h: curr(>500)=3682, peak=3681 counts.

00:01:29:50.87 WRN: I2C_read  @51h: curr(>500)=9999, peak=3682 counts.
00:01:29:50.88 ERR: I2C_reset_bus: Attempted to reset I2C bus.
00:01:30:16.30 WRN: I2C_read  @51h: curr(>500)=3671, peak=   0 counts.
00:01:30:42.44 WRN: I2C_read  @51h: curr(>500)=3681, peak=3671 counts.
```

When running the LMRST-Sat FSW, we see I2C bus resets (only when talking to the GPS module @ I2C address 0x51) on an occasional basis. Errors once every 30 minutes or so (where there are transactions every 5s between the MCU and the GPS module) are typical …

A reliable system on-orbit is preferable to one you fully understand on the ground.

N.B. Above you see the interplay between I2C reads and writes to the GPS receiver that take a long time (i.e., the I2C bus is busy for more than 500 counts, often up to 3,000-6,000 counts) but continue to function properly, and those that take too long (**`MAX_TRIES`** = 9,999) and indicate an I2C bus hang, requiring a bit-banged I2C bus reset from the MCU. There are no resets when talking to any other I2C devices, but their commands and telemetry are very short.

Addition of Autonomous Behaviors (FSW 2.x)

Recall that apart from antenna deployment, all of LMRST-Sat's actions are driven from the ground station. As part of a review with JPL as LMRST-Sat neared its final testing dates, based on concerns when communications with LMRST-Sat might be spotty, it was jointly decided to add these autonomous behaviors to the FSW:

- LMRST-Sat will automatically reboot (via the external WDT) after not receiving any commands from the ground after **2.5 days**.
- LMRST-Sat will automatically turn on its GPS module after an initial **18-hr** delay, then every **25 hrs** thereafter.
- After the GPS module is turned on, LMRST-Sat turns it off after **45 minutes**.
- After the LMRST payload is turned on, LMRST-Sat turns it off after **40 minutes**.

This functionality was trivially implemented via four cyclic timers (in one-shot and continuous modes). Cyclic timers are a part of the RTOS API, and cost around 20 bytes of RAM each.

```
void gps_power(uint8_t power) {
  if(power == 0) {
    [SNIP]
  }
  else if(power == 1) {
    [SNIP]
    OSDestroyCycTmr(OSTCBP(CYCTMR_GPS_AUTO_POWERDOWN));
    OSCreateCycTmr(gps_auto_powerdown,  OSTCBP(CYCTMR_GPS_AUTO_POWERDOWN),
                   gps_auto_powerdown_period, ONE_SEC, OSCT_ONE_SHOT);
    user_debug_msg(DBG_MSG, "gps_power" ": Turned GPS receiver ON & enabled auto-powerdown.");
  }
}
```

Other Problems Detected and Fixed (FSW 2.x & 3.x)

It was discovered that one manufacturer's module was "bleeding" +5V onto the **+5V_USB** bus, even when no USB was present or connected to LMRST-Sat. This suppressed the natural action of the external WDT, and would have been deadly on orbit. A hardware fix was required. Modding / changing this module was not an option, so other modules (with their manufacturer's help / instructions) were modified with the downside that they can no longer be powered via an external USB connection. Once the mods were in place, testing resumed and showed that the external WDT was restarting LMRST-Sat properly.

The FSW could only send telemetry packets (TAPs) to the ground at a rate of 0.5Hz, irrespective of size. This is partially because of a lack of flow control in the FSW's UART code. An adaptive scheme that takes the size of the TAP into account sped this up tenfold for small TAPs.

The original hardware configuration of LMRST-Sat components at JPL prevented the simultaneous use of the USB and radio connections. The radio was rewired to interface via the **MHX_xxx** signals, and the FSW was re-coded to support the radio on this interface.

Post-release Fixes (FSW 3.x)

As part of testing prior to shake-and-bake, it was observed that sending two of a particular command caused the system to lock up/reset. This was traced to a coding error in the HSS module and thought to be fixed in v3.0.3r. Ultimately it was found that the bug was more serious – basically, the I2C latch to the secondary HSS was not being reset/initialized at startup. Therefore, IF LMRST was already on via the secondary HSS, THEN LMRST-Sat would only turn LMRST OFF after the startup delay (default: **45 minutes**) following any reset or WDT-based restart. This might compromise the batteries.

Since money and time were running out, and since this situation would/could be serious only if the initial startup delay of 45 minutes were still active and the secondary HSS were ON, it was decided to freeze the FSW without fixing this.

An independent code review, along with a rigorously-enforced standard for initializing all hardware peripherals as a group at the very start of the FSW, would likely have caught this error.

Lessons Learned

• Build on software you know to be **reliable**. Make the FSW **modular**.

• **Standardize** coding / (re)**use** (internal) **coding templates** for maximum consistency and understandability across the FSW codebase. Use a **checklist**.

• Use every tool at your disposal – including analyzers, 'scopes, app notes, datasheets – to **fully understand the hardware you're working with**, and to fully characterize faults you observe.

• The more **channels of observability** you have into the FSW (e.g., debugging IDE, debug output monitor, saved logs) the more confidence you can have in its correct operation.

• Testing (esp. by parties other than the FSW programmers) is likely to reveal problems that were **overlooked** by the FSW team. Independent code reviews are useful, but no one wants to review spaghetti code.

• Sometimes money and time will run out and the FSW must be frozen – that's life. **Understand** (i.e., "own") **the FSW** in terms of its operations and limitations well enough to propose CONOPS to get around known problems.

LMRST-Sat FSW Contributors

Andrew Nuttall, Ashe Magalhaes,
Avishai Weiss, Benjamin Blake,
Brendan Tseung, Brian Thompson,
Bryan Lin, Connor Beierle,
Cyrus Foster, Dawn Wheeler,
Grant McLaughlin, Nicolas Lee,
Randy Lum, Seiya Shimizu,
Sonja Brajovic, Stephen Wolf,
Yonas Tesfaye

A big thanks to JPL:

Courtney Duncan, Fernando Aguirre,
Dorothy Lewis, Maxime Bize, Scott Tripp,
Matthew Chase, Richard Rebele

Andrew Kalman

REFERENCES

[1] Jim Greene, AN-686 “Implementing an I2C Reset“, Analog Devices, Inc., 2003.