Test Procedure: LunaH Digital Boards

Introduction:

This will outline the testing procedure Meg, Graham, and Erik will follow to verify the function of the LunaH project digital boards that will go into the spacecraft. This file should serve to inform on how the procedure will flow from one test to the next, what has been done so far, and what the individual verification tasks are for each function and/or part on the board or software. The document will outline tests for both hardware and software. The idea is to test from the bottom up; that is, test the low level functionality first, then build up from there to ensure that each smaller piece works before testing the larger ones. This will give a better granularity to the tests and allow us to debug faster and smarter.

The main project that is to be cloned for testing purposes is found on the RMD shared drive, here:

K:\users\MKaffine\LunaH\LunaHtest

Graham has moved this project to github to preserve the “base model”; that is, the code that Meg has determined works on the digital board with minimal necessary inclusions. We will expand and fill in clones or copies of this project with the tests which follow.

As much as possible, we will include detailed instructions for each test, but there will be a large amount of text copied from one test to the next, so the document will need to be read from the start so as to understand the context and continuity.

**Big Points** are bolded for clarity. Tests which have been performed with satisfactory results are marked in green. Tests which need further input are marked in yellow. No colored text indicates that those tests have not been reached yet.

FPGA to processor (works backwards to)

Tests:

1. Successfully locate the Zynq chip –done
   1. How was this performed?
   2. What were the voltages, time, date?
2. Successfully locate the ARM\_DAP -done
3. Successfully load a project using SDK (using simple bitstream sent to Sam-not used) done
   1. Which project is this? Where is it located on the Shared Drive?
   2. What is the confirmation that the project loaded successfully?
4. Successfully read a GPIO value from the FPGA done
5. Successfully write a GPIO value to the FPGA (en\_+5VSH to 1/ GPIO 7)
   1. Check that +5VSH TP is 5v (I think) (TP 8) done
   2. Set GPIO 4 High, check TEC\_CNTRLEN (Non-zero)
   3. Set GPIO 5 high, check TEC\_BLEEDEN (non-zero)

**SD**(1,0) -> testing to check that the sd card can be written from/to

These tests will build upon each other, as to test file access, we need to have mounted an SD card to the system and so on. The tests will verify SD card functionality first with a set of pre-created text and binary files, then with a set of application generated files. As we will be utilizing both ASCII/text and binary files, both must be fully checked out. A program to convert the binary files to ASCII for readability is located on the shared drive:

K:\users\GStoddard\Miscellany\b2t\_basic\b2t\_basic

The program is simple in nature and comments within the code should be able to aid with running the program. Any questions, call Graham (x6891).

**Tests for SD card:**

1. Mounting correctly

FATFS fatfs; //initialize a FAT file system

int ffs\_res = 0; //return variable for the SD card functions

ffs\_res = f\_mount(0, NULL); //unmounts anything that is currently mounted

ffs\_res = f\_mount(0, &fatfs); //mount a card at physical drive “0” or “0:”

1. Unmounting correctly

FATFS fatfs; //initialize a FAT file system

int ffs\_res = 0; //return variable for the SD card functions

ffs\_res = f\_mount(0, NULL); //unmounts anything that is currently mounted

ffs\_res = f\_mount(0, &fatfs); //mount a card at physical drive “0” or “0:”

ffs\_res = f\_mount(0, NULL); //un-mount the SD card

1. Successfully opening a txt file (pre-created)

FATFS fatfs; //initialize a FAT file system

int ffs\_res = 0; //return variable for the SD card functions

char cEventFileName[] = “file1.txt”;

…mount…

Ffs\_res = f\_open(&file1, cEventFileName, FA\_OPEN\_ALWAYS | FA\_WRITE);

1. Successfully closing a txt file (pre-created)
2. Successfully reading a txt file (pre-created)
3. Successfully writing to txt file (to a pre-created file)
4. Successfully creating a txt file
5. Successfully opening a bin file (pre-created)
6. Successfully closing a bin file (pre-created)
7. Successfully reading a bin file (pre-created)
8. Successfully writing to bin file (to a pre-created file)
9. Successfully creating a bin file
10. Check the usage of f\_stat (to examine the FATFS to see if a file exists)

**Uart** – (receiving values/sending values)

\*After being powered on, RE is high and DE is low as expected/intended

1. Check if receiver is enabled (~RE)
2. Check if receiving
3. Check if driver is enabled (DE)
4. Check if driving signals

**I2C** – (temp stuff)

**QSPI**(1,0)\*one is currently grounded out -> boot from it (QSPI 0 -> 1, QSPI 1, 2 -> 0, QSPI 3 -> X, preferably 0), (if not working, check if works in 2015.3. QSPI part changed, current part might not be supported in 2014.1)

(later)**ADC** -> check if signals transfer correctly (reading fpga -> processor)(this step after reading from FPGA) (control signal)

**interconnectedness**