ArduSat Pi:

A redundant satellite computer based on Arduino and Raspberry Pi

NASA SpaceApps Challenge 2013, Santiago, Chile – Team 7

Abstract

This document describes a possible architecture for an extended ArduSat payload, produced during SpaceApps Challenge 2013. The proposed system includes an applications processor (Broadcomm BCM2835) and a microcontroller (Atmel SAM3X). This means the system is roughly compatible with the Raspberry Pi and Arduino Due platforms, connected with an I2C bus.
To address the problem of Single Event Effects produced by radiation, and to improve the system's reliability, a redundancy mechanism is introduced. This mechanism is implemented using a FPGA.

Introduction

The ArduSat is a cubesat-compatible satellite based on GOMspace's platform and the Arduino development board. Its purpose is to open satellite technology to hobbyists and developers around the globe, allowing people to "hack" into space. The potential this presents is huge: opening satellite data to the masses might be the space industry's equivalent to the PC (which triggered fundamental advances in the computer industry). Moreover, such a simple and light satellite opens a wide range of research opportunities, like creating an orbiting sensor mesh or planning missions beyond LEO with reduced propulsion mass requirements.

Fig. 1: GOMspace's GOMX platform

The purpose of this project is to extend this platform by adding processing power, provided by a Raspberry Pi computer. This would improve the satellite's capabilities by introducing a complete Linux system that can be hacked from Earth and gathers data in Low Earth Orbit.

Given the limited time frame for the development of this solution, a general architectural description is presented, trying to cover as many details as possible but not providing CAD files of any sort. This is pending for further development (and requires obtaining access to Broadcomm's design files for BCM2835, not publicly available). It should be noted that the BCM2835 could easily be replaced by a similar applications processor, like Freescale's i.MX or TI's AM335x, and most of the proposed solution would remain the same.

The problems

There are several difficulties in designing flight hardware; this solution focuses on two of them:

On the one hand, a cubesat is a small and constrained system: there are power, mass and volume limitations. The solution must be as least power-hungry as possible (less than 4W approximately, assuming a worst-case 3U cubesat) and should not cause mechanical complications.

On the other hand, space hardware suffers from radiation. Among other things, this can cause logic gate latchups and bit flips which can ruin a digital circuit's functionality. The solution should try to involve commercial-grade components (to reduce costs and simplify manufacturing) but should be as radiation-resistant as possible.

Proposed solution

A block diagram for the proposed system is shown in Fig. 2. It is the most important output of this project so far.

As we mentioned earlier, the main components are the BCM2835 and SAM3X ICs. Program memory for the applications processor resides in SD-compatible NAND flash (like SanDisk's iNand). The processor and microcontroller are interfaced using the I2C protocol, which minimizes the amount of connections required. This protocol also connects the BCM2835 (which acts as master) to the rest of the platform (radio, attitude control, etc) and another I2C bus connects the Arduino to the sensors. A simplified functional diagram, ignoring redundancy and monitoring, is presented in Fig. 3.

There is also a UART connection between the Raspberry Pi and Arduino processors, that allows programming of the SAM3X in-flight. This is essential, as one of the main objectives of this mission is to open the platform for hacking.

Redundancy is introduced on several subsystems:

- BCM2835 is duplicated (dual redundancy). The primary processor works while the other one sleeps; if it fails (the mechanism for failure detection will be explained in the section “FPGA Circuitry”), the backup processor is activated.
- NAND flash has triple redundancy. This is a very effective error correction mechanism, as a bitflip will not cause data corruption. It should be noted that NAND flash also includes an internal ECC.
- SAM3X has triple redundancy, and all of its' outputs pass through voting circuitry. The three processors are clocked with the same external oscillator so that they can work in lockstep. The clock also synchronizes a monitoring system that resets the processors if one fails repeatedly.

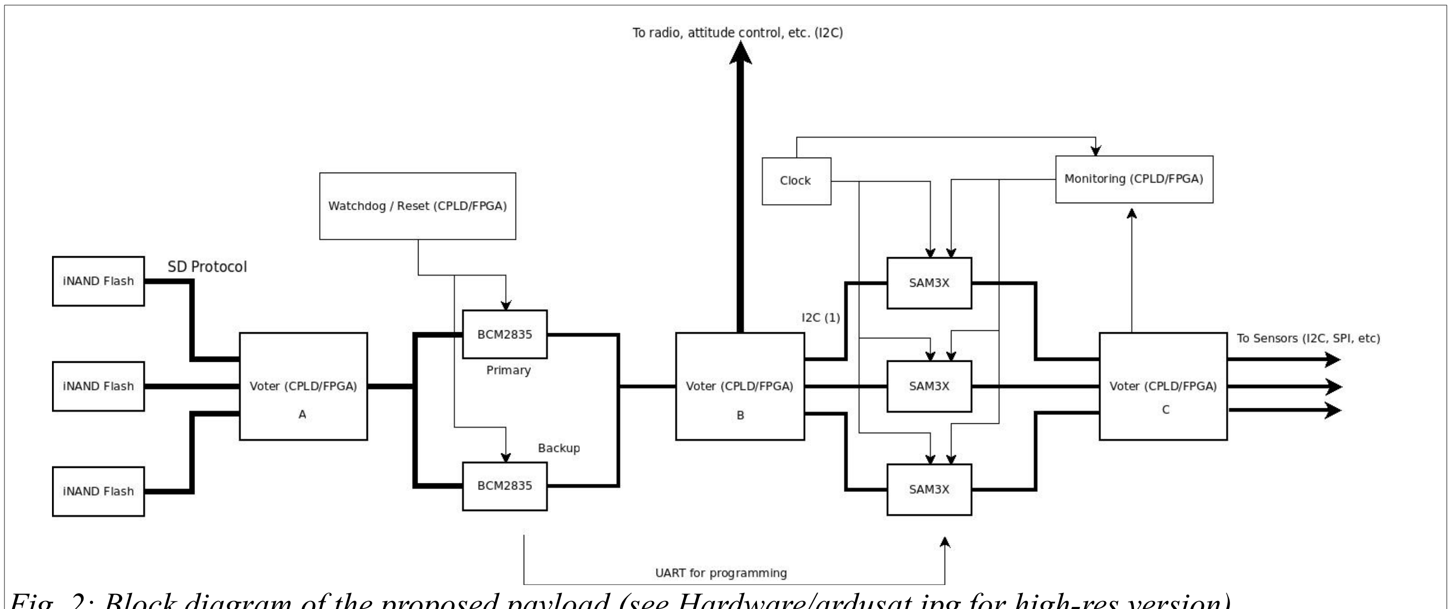
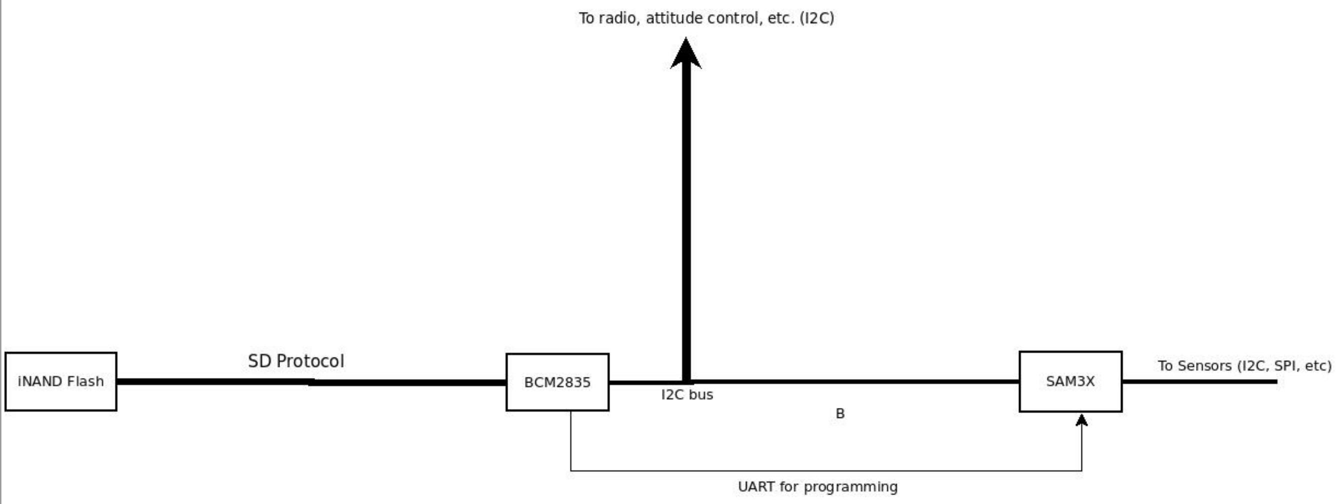


Fig. 2: Block diagram of the proposed payload (see Hardware/ardusat.jpg for high-res version)

Redundancy in flash is critical as a bit error can cause unexpected results (for instance, a bitflip in a processor opcode could result in erratic behaviour or jumping to an invalid address). Redundancy in processors helps to mitigate the effects of latchups and bitflips in the processors' internal circuitry. In our initial design, we had used triple redundancy for the BCM2835; however, it was clear that this involved an excessive power consumption (it's the highest frequency device and therefore the most demanding). A tradeoff solution is dual redundancy in which only one processor works at a time.

SAM3X is a lower frequency device and its power consumption is well below the BCM's, so adding a few more processors shouldn't have a big impact on the power budget.
Regarding power budget, it should be noted that processors can be under-clocked, if necessary, dramatically reducing power consumption at the cost of reduced processing power.


To radio, attitude control, etc. (I2C)
iNAND Flash
SD Protocol
BCM2835
I2C bus
B
SAM3X
To Sensors (I2C, SPI, etc)
UART for programming
Fig. 3: Simplified block diagram, ignoring redundancy and monitoring

FPGA Circuitry

The FPGA or CPLD (which might suffice if the circuit ends up being simpler than expected) is responsible for managing the redundancy in a transparent way, so that at software level it appears that we're running a single processor with a single memory. This allows for more efficient software development and a closer compatibility with Pi and Arduino devices.

The FPGA has several blocks in different parts of the circuit:

Voter

The Voter circuit is responsible for combining outputs from three redundant modules and producing a single output. The most basic implementation (for the unidirectional case) of a 1-bit voter is presented in Fig. 4.

This should be replicated, for instance, in each data line for the SD memory. Handling bidirectional lines involves a more complex circuit.

The greatest disadvantage of this circuit is that the OR-gate is a single point of failure (SPOF).

There are alternative designs involving triple-redundancy for the voting circuit, producing what is usually called Triple-modular-redundancy (TMR).

Watchdog / Reset

This block is responsible for managing the BCM2835 dual redundancy. A low-level routine should be programmed into the Linux kernel that periodically toggles a GPIO pin. This block, then, assumes that the processor has failed if the signal isn't toggled within a certain time. In that case, it wakes up the backup processor, resets the primary and inverts their roles from that moment on. (This description can easily be implemented in VHDL with an overflowing counter module).

SAM3X Monitoring

This block is analogous to the Watchdog: It controls that the SAM3X processors are working properly. It is important that they work in lockstep, so a GPIO signal should also be used to ensure they are synchronized. This block also includes an error counter for each processor, that is incremented every time a core outputs a different value than the other two. If a core fails more than a certain amount of times consecutively, we can assume that the code has entered an erroneous state (for instance, it has branched opposite than the others), so the system should be reset to fall back to a valid state.

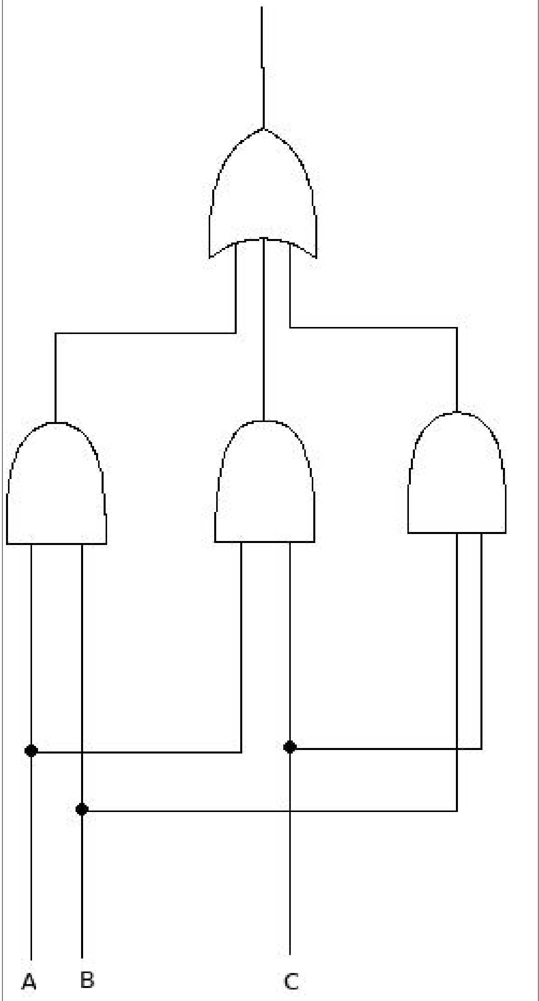


Fig. 4: 1-bit unidirectional voting circuit

Software architecture

A software architecture for this platform should be designed in accordance to the way experiments will be carried out. In general, an experiment might consist in some or all of the following steps:

- A user program is loaded into the Raspberry Pi via radio. The code for Arduino is also received by the Pi, which in turn programs the Arduino.

- The Raspberry Pi instructs the Arduino (via I2C commands) to perform a series of sensor measurements.
- The Raspberry Pi interacts with the attitude control computer (also via I2C) to align the satellite for the experiment (for instance, pointing the camera in a certain direction).
- The Arduino performs the measurements and sends the data to the Raspberry Pi.
- The Raspberry Pi processes and compresses the data and produces an output that is transmitted back to Earth via radio.

Services and permissions

Some of the satellite's functionality should only be accessible to the end user through a Service API, to prevent errors from destroying the platform. Linux permissions and multitasking can be very helpful for this: the application can run in an isolated environment, accessing the mission-critical hardware with a limited set of commands (for instance, using a pipe or socket to communicate with another process that is serviced by the driver). This is especially important for the attitude control, power and radio subsystems.

Controlling permissions for the Arduino is more complicated. The most important thing is to ensure that the I2C that is connected to the Raspberry Pi is configured as a slave at all times, so that the Arduino cannot access the flight control, radio and power subsystems. The best way to do this is to develop a testbench that analyzes the program before uploading, that monitors the I2C registers and provides a decent code coverage.

Conclusion

We have presented a complete description of a satellite payload subsystem that extends the functionality of the ArduSat by adding a Raspberry Pi computer. We have designed a solution that takes radiation issues into account and takes action to extend the satellite's life and improve its reliability.

This solution uses redundancy in an almost transparent manner, producing a hardware platform that is compatible to these popular boards. Redundancy reduces the probability of failure in a degree we expect to be sufficient given the low-cost philosophy of these type of satellites.

We have also programmed a proof-of-concept using the development boards that shows the potential and simplicity of this solution, by transmitting a set of sensor data from the Arduino to the Raspberry

Pi.

The next step towards effectively building this satellite is to design a PCB (and a FPGA RTL) following these specifications and perform thorough testing.

We hope this product, like its predecessor, will help open space technology to a wider audience and in this way contribute to mankind's quest for the stars.

Bibliography

Angilly, Ryan, (date unknown), TREMOR: A *triple modular redundant flight computer and fault-tolerance testbed for the WPI Pansat nanosatellite*
http://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1713&context=smallsat

Lightsey, E. Glenn (2012), *Operational Considerations for CubeSats Beyond Low Earth Orbit.* iCubeSat Workshop, Cambridge, Massachussets
http://icubesat.files.wordpress.com/2012/06/icubesat-org-2012-a-2-5_presentation_lightsey_201205261035.pdf

Raspberry Pi schematics
http://www.raspberrypi.org/wp-content/uploads/2012/04/Raspberry-Pi-Schematics-R1.0.pdf

ArduSat schematics
http://tinyurl.com/ArduSatPayloadv1

Raspberry Pi with Arduino as I2C-Slave
https://github.com/binerry/RaspberryPi/tree/master/snippets/c/i2c-arduino

SanDisk iNAND datasheet
http://www.spezial.com/commercio/dateien/produktbeitraege/DS_-_iNAND.pdf

Broadcomm BCM2835 ARM Peripherals
http://docs-europe.electrocomponents.com/webdocs/1072/0900766b81072cb7.pdf

AT91SAM ARM-based Flash MCU - SAM3X SAM3A Series

http://www.atmel.com/images/doc11057s.pdf

#ardusat freenode channel (collaboration with other teams around the world)