

FPGA-Based Hardware/Software Co-Design Instrument Command-and-Control Solutions

Thomas A. Werne, Robert F. Jarnot, Mark A. Allen
Jet Propulsion Laboratory, California Institute of Technology

April 29, 2010

1 Introduction

TRADITIONAL design methodology dictates that an instrument electronics platform should consist of a printed circuit board (PCB)—often multiple boards—populated with many components and support electronics. More specifically, each subsystem is composed of numerous integrated circuits (ICs) spread across the board(s), connected with innumerable traces and wires. Additionally, it is often the case that the hardware and software design teams are quite separated—the hardware team designs a board and hands it to a software team, who must then deal with any hardware quirkiness and may need to rewrite software due to inadequately defined interfaces.

Instead, it is possible to significantly reduce many of the problems associated with this method by utilizing modern, flight-qualified, highly capable FPGAs (field-programmable gate arrays). Frequently the majority of an instrument electronics design (almost all of the command-and-control components) can be implemented within an FPGA, which decreases the number of ICs that populate a board and the associated support electronics and interconnect. The result is a much-simplified board design, with potential savings in both mass and power. By using an FPGA with a processor instantiated in the fabric (or a dedicated hard processor in the silicon), the hardware and software teams can consist of the same people, who understand the design decisions made at every level and can more easily make changes to software and hardware simultaneously as the design evolves. This allows the teams to excise strange hardware (and software) behavior—which often stem from poorly defined interfaces—as part of the natural hardware/software co-design process. Finally, since a majority of the electronics can be designed using HDLs (hardware description languages), the design can be simulated and tested much earlier and far more often in the design process.

This, coupled with the software-definition of both hardware modules and software processes, allows for unprecedented flexibility in moving modules between hardware and software as the need arises (e.g. as they are identified, repetitive, computationally intensive tasks can be moved into hardware, and slow, customizable tasks can be placed into software).

In order to show that this is a viable methodology for instrument electronics work done at JPL, a case study using the SOAR (Spectroscopic Observations for Atmospheric Research) instrument being proposed for the 2016 ExoMars mission has been implemented. In this report, electronics requirements for the instrument are discussed, and it is argued that the specific requirements for this mission generalize naturally to many other instruments. Differences between a proposed design using current methodology and an FPGA-based design are listed and described, and the results of implementing a subset of these required capabilities on a modern Actel FPGA are presented.

2 Instrument Design Requirements

SOAR is an instrument designed to collect spectroscopic atmospheric data. In order to do this, the instrument electronics need to interface with an existing application-specific IC (ASIC) (designed by U.C. Berkeley) that does the spectroscopic analysis. It also needs to communicate with a set of COTS frequency synthesizers. In order to point the sensor at a target, the instrument needs to be able to control a set of mechanisms (e.g. motors and encoders) as well. As with other instruments, SOAR will also need to collect housekeeping data and communicate with the main spacecraft computer. Due to the dynamic nature SOAR is targeted for, all of these capabilities need to be reprogrammable through in-flight uploads.

Generalizing slightly, these capabilities represent specific instances from broad categories of tasks that many instruments must accomplish: high speed digital communication, low speed digital communication, real-time digital control signal generation, low speed analog communication, and communication with the spacecraft. In the case of SOAR, high speed digital communication is manifested by the need to communicate with the spectrometer ASIC. Low speed communication digital communication is necessary to interface with the frequency synthesizers. Low speed analog communication is used for collecting housekeeping data. In order to produce finely-tuned and exact motor movements using encoder feedback, real-time digital command and control is necessary.

To summarize, there are several modifiable specific tasks that the SOAR instrument electronics must be capable of performing, each of which is an instance of a more general class:

1. **High Speed Digital** — Communicate with Spectrometer
2. **Real Time Digital** — Motor Control with Encoder Feedback
3. **Low Speed Analog** — Housekeeping Data
4. **Low Speed Digital** — Communicate with Frequency Synthesizer
5. **Spacecraft Interface** — 1553 for Command, SpaceWire for Data

3 System-on-Board vs. System-on-Chip Design

Classic instrument design practices have, up to this point, been restricted due to existing technology. At his/her disposal, a design engineer has many different components for a solution, including: microprocessors (e.g. RAD, Mongoose), microcontrollers (e.g. 8051), custom ASICs, FPGAs (e.g. Actel RTAX), discrete logic, etc. From this set of components, the engineer chooses those which “make sense” for a particular subsystem. These different subsystems are laid out on dedicated PCBs, and interfaces between subsystems on a single board and between boards are designed and implemented. The final product is a collection of many different dedicated components and subsystems, all connected together electrically via wires and traces—which are susceptible to electromagnetic interference/compatibility (EMI/EMC) concerns—and mounted in a chassis—which presents vibrational and mass concerns.

As FPGA technology has improved and matured, a new design methodology has emerged: system-on-chip. This term describes the concept of consolidating onto a single chip a design that would normally fill an entire (or even multiple) electronics board, except perhaps for specialized devices (sensors, high voltage/current drivers etc.). For instance, to fill the role of a microprocessor, Actel FPGAs are now capable of holding a Cortex-M1 (ARM architecture) or LEON (SPARC architecture) processors. Along with these soft microprocessors comes a well-documented bus architecture—in the case of these two, the Advanced Microcontroller Bus Architecture (AMBA).¹ Commonly used peripherals such as timers, UARTs, etc. and custom digital electronics are instantiated in the FPGA fabric along with the processor and communicate via this same bus architecture. In addition, many of these commonly used peripherals can be purchased in a pre-validated, bus-compliant form from RTL vendors.

¹Designed by ARM, AMBA specifies at least two buses: the Advance High-performance Bus (AHB) for high-bandwidth devices, and the Advanced Peripheral Bus for low-bandwidth devices

4 Design Methodology Comparison for SOAR

For the purposes of this demonstration, consider only the command and control portion of the electronics. Spacecraft interface (presumably via 1553 for S/C communication and SpaceWire for data link) could be added to the discussion with little/no effect on the conclusions, so they are not examined at this point. Indeed, validated versions of these are available for purchase that are AMBA-compliant, and so integrating them into any design comes down to writing the requisite software.

Figure 1 shows block diagrams for a SOAR instrument electronics design using the traditional techniques—Figure 1(a)—and using FPGA-based system-on-chip design methods—Figure 1(b).

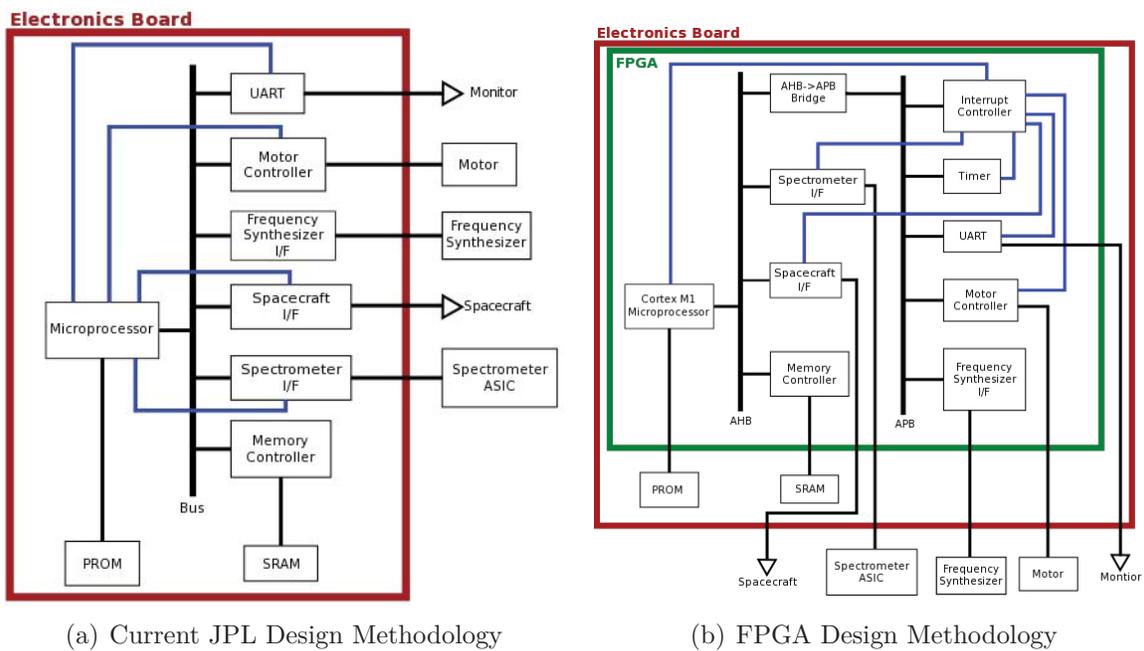


Figure 1: Block diagrams of example solutions. Separate boxes in red boxes are different components on an electronics board; components in the green box are different peripherals in an FPGA; blue signals are interrupts.

The blocks in Figure 1(a) show discrete components that would need to be placed on an electronics board. This actual implementation of each component depends on its specific role, but could be accomplished by:

- Microprocessor
- Microcontroller
- ASIC
- FPGA
- Discrete Logic
- etc.

The blocks in Figure 1(b) show essentially the same blocks, although there is not a 1-1 mapping due to the ARM AMBA bus architecture.

As noted above in Section 3, there are a few differences in the design that should be examined.

First, there are more functional blocks in the proposed FPGA solution (the APB, AHB-to-APB bridge, interrupt controller, and timer); however, the reasons for these differences are due simply to the architecture of dual-bus architecture of AMBA and the implementation of the Cortex-M1 (no built-in interrupt controller or timer). These differences would have to be reconciled with the selected processor and bus architecture—it is entirely possible that the standard design would have to incorporate these extra blocks. In fact, since these blocks are required only for interconnection with Actel cores, Actel provides them free-of-charge with their design tools.

Second, the bus architecture is predefined by ARM and all of the vendor-provided cores interface directly using that specification. Connecting custom HDL cores to this bus is as simple as implementing an wrapper layer that conforms to the specification.^{2,3} Also, Actel provides a tool called a “bus functional model” (BFM) which allows the designer to completely test the interface layer and peripheral using logical memory read/write operations instead of attempting to drive all of the bus signals manually. This simplifies the designer’s job by removing all uncertainty that the peripherals will communicate properly. All operational scenarios can be completely tested before-hand, including all bus transactions.

Third, many of the components shown (Cortex-M1, memory controller, AHB-to-APB bridge, interrupt controller, timer, and UART) are available as pre-verified, customizable cores from the vendor and come with driver software. This means that these routine interconnect parts

²Available with registration from ARM at: <https://silver.arm.com/download/download.tm?pv=1062760>

³Available for download: http://polimage.polito.it/~lavagno/esd/IHI0011A_AMBA_SPEC.pdf

can be assumed to work from the beginning. The designer does not need to spend time implementing, testing, and verifying/validating these. Also, errors that occur in the finished product can be guaranteed to not originate in these devices.

Finally, nearly all of these blocks (i.e. except the PROM and SRAM), are instantiated in the FPGA fabric. Once again, this means that an entire functional design can be simulated and tested before ever loading the design onto a physical device. Since this testing can be done at a behavioral level, all of this simulation and testing can be performed prior to the FPGA place-and-route—the most time consuming step in FPGA implementation. Also, since there are fewer ICs to be placed on a PCB, there is also a savings in the number of auxiliary components (impedance-matching resistors, filtering capacitors, etc.). As a byproduct, this decrease in parts-count also helps lower concerns about parts failure, bad solder joints, etc. Last, the decreased parts count and confinement of high-speed signals to within the FPGA decreases the sources of EMI and components that need to be checked for EMC.

5 Current Implementation State

As described Section 2, there are several critical capabilities that a complete electronics solution needs to possess: high speed digital communication, real-time digital communication, low speed analog communication, low speed digital communication, and spacecraft communication. To show that an on-board microprocessor can be used to coordinate all of these duties and satisfy the in-flight programmability, an additional requirement is adjoined requirement: software command and control.

5.1 Simplifications for Demonstration

Since the goal of this project is to demonstrate that this design methodology can be applied to instrument electronics, rather than produce a completely functional design, a few simplifying assumptions were applied:

1. Frequency synthesizers were not received in time to integrate them into the system. However, their communication link is standard SPI, so an Actel-provided SPI module was included into the design—which comes with pre-built device drivers. Thus, while the devices are not integrated into the system, doing so only requires writing small amounts of software, which is only a few days of work.
2. The motor control interface is a pre-existing motor-controller board rather than direct drive signals to the motor. Designing and implementing a motor controller in HDL in the FPGA is a tedious, yet standard task, and can be completed in a few weeks.

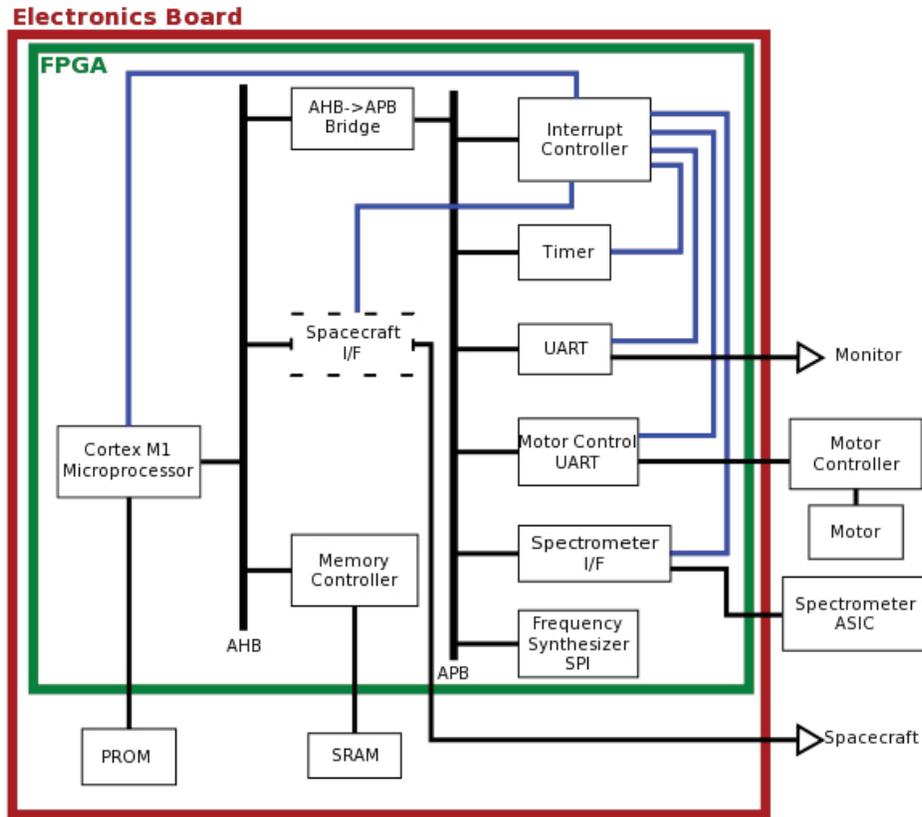


Figure 2: Block diagram for implemented design: blue signals are interrupts, components in the green box are separate peripherals in the FPGA, components in the red box are separate ICs on the electronics board, dashed components are available for purchase and have not been implemented

3. Low-speed analog communication is another standard endeavor which can be completed in a short amount of time and, due to its device-specific nature, was not attempted.
4. Actual spacecraft interfaces were not examined in any specificity. These are quite complex peripherals, but independently validated core that interface directly to the AMBA bus are available for purchase (1553 from Actel or Gaisler, SpaceWire from Gaisler). In-house development would undoubtedly be far more expensive and doing so is marginally beneficial at best. In the event that JPL purchases licenses for this IP, interfacing them to a design should take a matter of days.
5. A spectrometer simulator that outputs a simulated processed data stream was used instead of raw ASIC output to provide known, easily verifiable data. The processing that is done has already been implemented on a similar Actel FPGA (and exists on the Berkeley prototype flight-like spectrometer board), so migrating it to the Actel part

and interfacing with the actual ASIC is trivial. Conversely, the simulator with the Berkeley board containing the actual spectrometer can replace the simulator at any time, since both the simulator and the board have identically formatted output and timing.

5.2 Implementation

Keeping these points in mind, Figure 2 shows a block diagram for the simplified design that was implemented. To reiterate, these changes are not critical to the design—a competent engineer familiar with digital design and software can easily implement these deltas over the course of several weeks. Specifically, the changes from Figure 1(b) are:

1. Left the frequency synthesizer physical SPI interface unconnected and did not write the driver software.
2. Controlled the motor through a UART interface instead of writing a dedicated motor controller.
3. Wrote a custom peripheral to communicate with the spectrometer simulator at 4 MHz instead of folding the Berkeley ASIC controller code into the FPGA.

The design in Figure 2 was completed in ~ 1.5 man-months for less than \$30K. This includes time to learn the design tools and flows, understand the design of the AMBA bus architecture, get up-to-speed with Actel’s BFM, and write C code to run on the microprocessor and drive the peripherals. From tests and verification, the design works flawlessly and has no problems fitting within the FPGA and meeting timing requirements. This short time-to-solution is not a fluke. Given a few more months a full implementation can be completed.

5.3 High Level Design Verification

As mentioned early in this section, in addition to the instrument design requirements, software command-and-control is also desired. In order to accomplish this requirement, a top-level user interface was implemented in the form of device-specific control functions and a port of uShell—a shell written by this report’s first author and originally targeting Xilinx FPGA Microprocessors (e.g. PowerPC 440, Microblaze). When connected to the UART via a terminal emulator (e.g. HyperTerminal on Windows, Kermit on Linux), the operator is presented with a command-line interface, wherein he/she can use these device-specific functions to communicate with the different devices and send them commands. With the addition of standard system functions (e.g. sleep), a rudimentary command sequence can easily be written and sent out to the electronics package. More sophisticated commands can

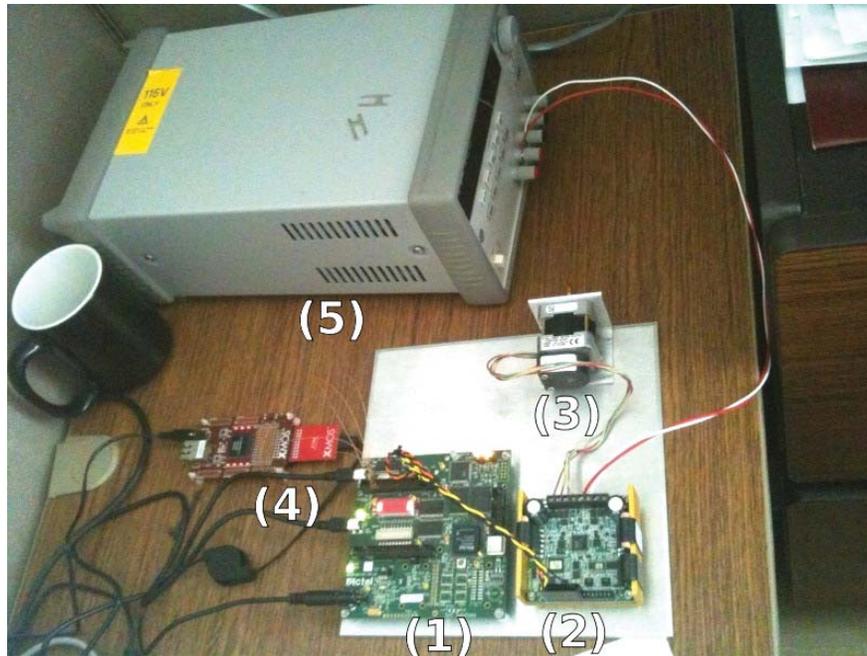


Figure 3: Demonstration platform and components: (1) FPGA control board, (2) motor controller, (3) motor, (4) spectrometer/simulator, (5) power supply

be written and registered with the shell, providing higher-level control capabilities.

For the purposes of this demonstration, successful verification is defined as:

1. Communication with, and commanding of, the motor controller.
2. Communication with, and commanding of, the spectrometer simulator.
3. Validation of the output data from the spectrometer.

All of these criteria have already been successfully demonstrated and verified by commanding the devices using the shell interface and ensuring that the expected result of the command occurs, as per the test plan:

1. Command the motor and watch for the shaft to turn appropriately.
2. Enable and disable data collection from the simulator.
3. Dump collected data and compare it to the known simulator output.

6 Conclusion

This report has discussed and shown many of the benefits that can be realized by using system-on-chip and hardware/software co-design methodologies, targeting FPGAs, for instrument electronics work. In particular, all of the critical portions of the command-and-control (except large memory devices) for a modern instrument—in this case SOAR—can be completely rolled into a single IC. This leaves volatile and nonvolatile storage ICs—e.g. RAM and PROM—as the only other command-and-control related chips on the electronics board (which must still be populated with application-specific devices and support electronics). There are many implications, including (but not limited to): decreased mass/power, lower parts count, fewer problems due to EMI/EMC, and tighter coupling of hardware and software. This method allows for rapid development of solutions that can easily be simulated and tested ad nauseam before ever being burned onto a chip.

The work contained herein has the potential to produce significantly decreased costs due, in part, to the quicker development and smaller team size as compared to more traditional approaches. A final result of this non-standard methodology is an increase in component reusability. Once vendor-supplied and custom cores are integrated into a design, successful operation verified, and thoughtful and appropriate documentation collected—e.g. complete I/O descriptions, timings diagrams, descriptions of parametrizable options, testbenches, etc.—these identical peripherals can be transferred to any other project and dropped in place without any new design or major testing required. As more projects adopt this methodology, these technologies can be collected, shared, updated, and used as building-blocks in later tasks. JPL can leverage this collection of proven, reusable, vetted modules for future instrument and flight designs.

The research described in this report was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

© 2010 California Institute of Technology. Government sponsorship acknowledged.