

How to Apply FPGAs to Robotic Control Systems

With Spartan-3E FPGAs, your custom IP, and a PicoBlaze soft processor, you can create an exceptional solution to robotic design challenges.

by Jim Harris
Senior Design Engineer
Sunrise Telecom
jkh_jr@mindspring.com

In robotics design, you often find yourself stuck between developing custom CPLD IP to handle a particular hardware interface and sending the information to a processor to handle the data, or wasting processor cycles polling the relatively slow hardware interface and using the processor to directly interface to the hardware. Both methods are inefficient and costly.

It would be great if one device could use custom IP to interface to the hardware, process the data, and send the response back to the robot's hardware. One device would replace a CPLD, processor, and memory. This device exists in the Xilinx® Spartan™-3E family of FPGAs using your custom IP and a PicoBlaze™ soft-processor core.

In this article, I'll discuss methods of implementing robotic control systems using a Spartan-3E device and a PicoBlaze processor. This is a very simple project that illustrates how the subsystems interact. I'll also explain how to divide the hardware and processor duties for an efficient system.

This project will decode pulse-code-modulated (PCM) input signals from a standard radio control receiver using custom IP. It will process these signals using the PicoBlaze processor. The result will control the wheels on a differential-steer robot chassis using more custom IP to drive the motors. The project will also implement two UARTs: one for connection to a radio frequency modem and the other for connection to a GPS receiver. (The project is not currently using either of these interfaces, but I have provided them for reference and future expansion.)

Implementation Methods

Robotic control systems require the following subsystems:

- Systems that are ideal for hardware processing:
 - A machine interface system: this includes motor drives, sensors, timers, analog-to-digital converters, counters, actuators, or anything that connects to the real world or measures a real-world parameter.
 - Control and digital signal processing (DSP) systems: these control the low-level actuators by using sensors that provide feedback for the robot. The simplest form of this is when you control the system from visual range and perform the processing based on your visual feedback. A more complicated system could use proportional-integral-derivative (PID) loops or state-space equations that use sensor inputs to keep the robot under control; for example, using wheel encoders and a PID loop to make a robot go in a straight line.
- Systems that are ideal for software processing:
 - A user interface system: this provides you with a means to control the robot. In its simplest form, this could be a radio control transmitter and receiver. In its most complex form, it could be a real-time satellite uplink. The bandwidth of the system is determined by the amount of data that is sent between the robot and the user control station.

- Autonomous navigation system: having a robot navigate its environment with onboard navigation systems and waypoints is highly complicated and best done with a processor.

There are basically four methods for implementing robot control systems:

- A microcontroller-based implementation – using a microcontroller to interface to the hardware, handle the user interfaces, and system processing. A processor is ideal for running a user interface and autonomous navigation systems. However, using a processor to control hardware interfacing, asynchronous signal interfaces, and signal processing systems is computationally intensive.
- A programmable logic-based implementation – using an FPGA to control everything. FPGAs are ideal for controlling hardware interfaces and signal processing, but state machines become very complicated for user interfaces and navigation processing.
- A microcontroller plus programmable logic-based implementation – previ-

ously the best-of-both-worlds solution. The processor is used for the user interface, navigation processing, and signal processing. The CPLD handles custom hardware interfaces. The only problem with this approach is that this method uses a lot of processor time to perform DSP in the processor. This could be an issue if you use complex inputs like video images, RADAR, SONAR, or laser ranging systems.

• An FPGA with embedded processor core and custom IP implementation – today's best-of-both-worlds solution. The embedded processor is used for the user interface and navigation processing. The FPGA is used for hardware interfaces and signal processing. By implementing the hardware interfaces and signal processing in logic, the embedded processor is freed up to handle the user interface and navigation. Also, implementing the signal processing in pure hardware is the fastest way to perform DSP. This means higher bandwidths for processing complex inputs.

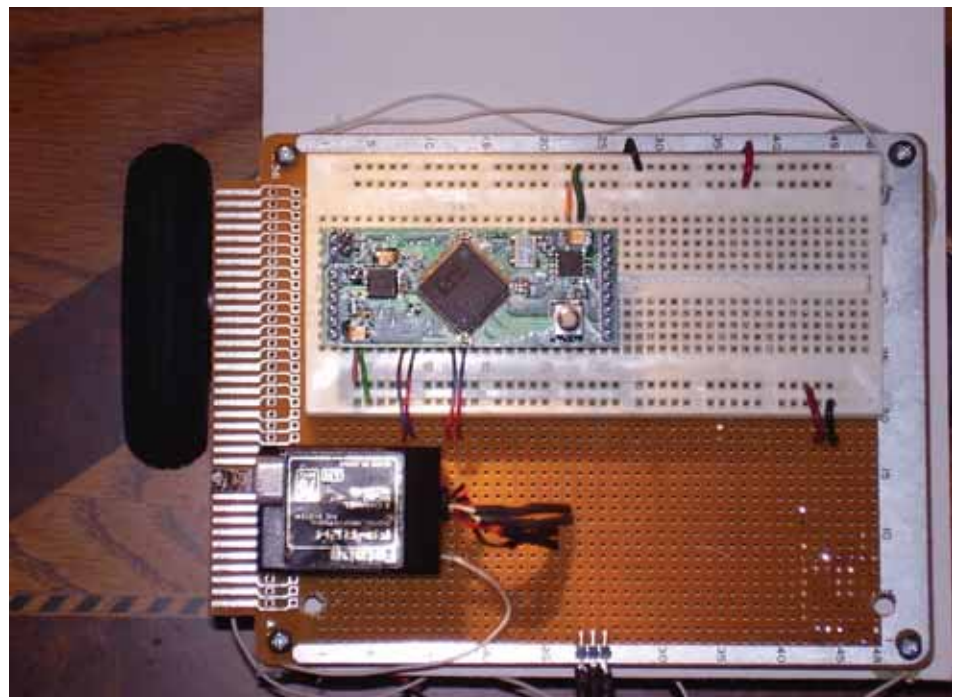


Figure 1 – Top view of Slo-bot showing the Spartan Stamp plugged into a breadboard.

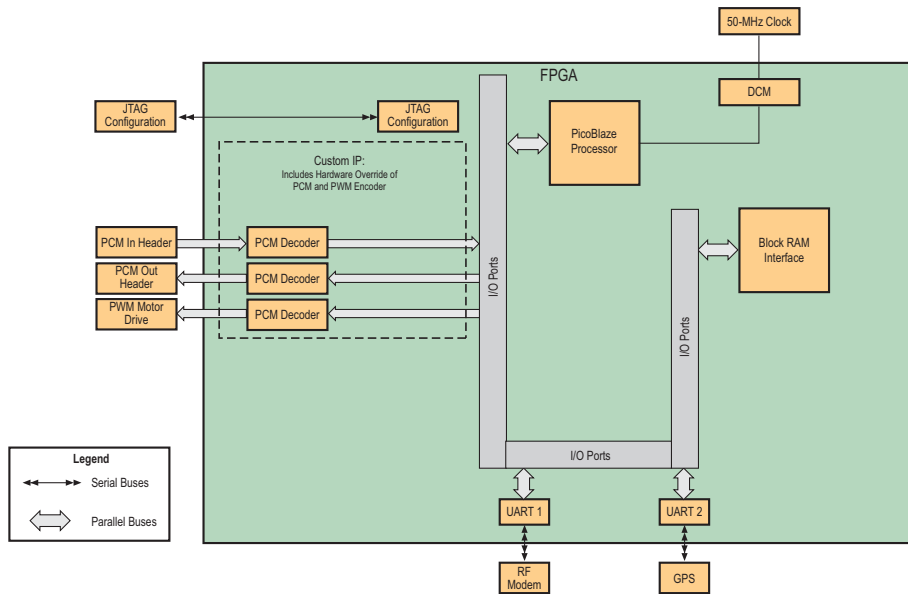


Figure 2 – Spartan Stamp architecture showing PicoBlaze instantiation

Top-Level Hardware

For this application I am using a very minimalist FPGA board designed to be inexpensive and easy to use. I call it the “Spartan Stamp” because it looks similar to a Parallax Basic Stamp embedded processor system (see Figure 1). Instead of having an embedded processor on it, it has a Spartan-3E XC3S250E device. Future low-cost versions will use the Spartan-3E XC3S100E device.

This system comprises:

- The XC3S250E – VQ100. The Spartan-3E FPGA has 250K gates, 12 hardware multipliers, 27K bytes of block RAM, and 66 I/Os in a 100-pin package.
- An inexpensive 8-Mb SPI serial flash. The Spartan-3E device requires 2 Mb for configuration, leaving 6 Mb available for user storage.
- A 50-MHz crystal oscillator – very fast for such a simple system. A PicoBlaze embedded processor core executes one command every two clock cycles, yielding 25 MIPS for the system. This should be fast enough to handle just about any processor job.

- One user push button: I use it for master reset, but you can use it for anything.
- One user LED that I use as a “heart-beat LED” to indicate that everything is okay. If the system is running, it beats once per second. It can be used for anything.
- Digilent Parallel III JTAG port. My configuration port of choice, the Parallel III cable is available from Digilent at a very reasonable price. I use it to configure the FPGA and to reconfigure the block

RAMs used as PROMs on the PicoBlaze processor.

- Digilent Parallel III SPI flash programming port. Xilinx Application Note 445, “Configuring Spartan-3E Xilinx FPGAs with SPI Flash Memories” (www.xilinx.com/bvdocs/appnotes/xapp445.pdf), shows you how to program the flash using a DOS command-line application.
- A mode jumper allows JTAG or SPI flash configuration.
- A PROG_B jumper pulls up all I/O pins and allows you to program the SPI flash.
- A UART port is the standard interface to the PicoBlaze core.
- 36 general-purpose FPGA I/Os for anything you like, such as PCM inputs, PCM outputs, UART ports, SPI ports, I2C ports, switches, or LEDs.
- 40-pin DIP footprint comprising 2 rows of 20 pins, 0.1-inch headers each. They have the same footprint as a 40 pin by 0.6-inch DIP. Of the 40 pins, 36 are the I/Os, 2 are 3.3 VDC supply pins, and 2 are ground. By using the 40-pin DIP footprint, you

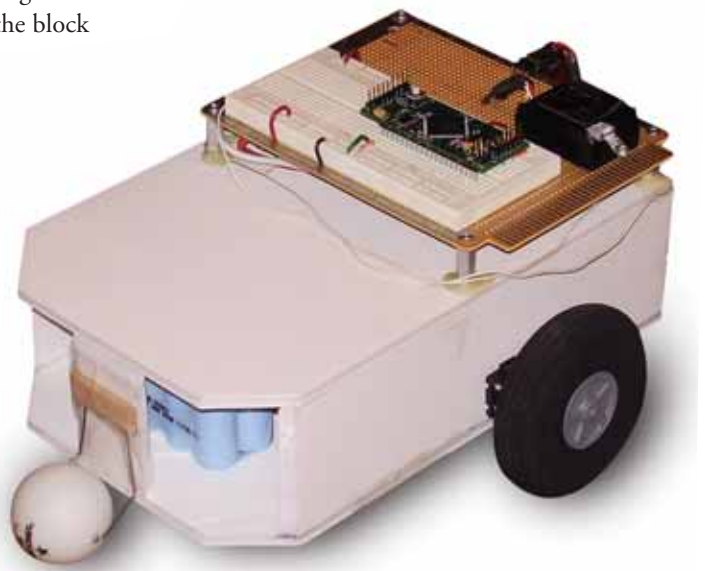


Figure 3 – The Slo-bot, including Spartan Stamp radio control receiver, robot chassis, and NiMH battery pack

can use the board with any general-purpose bread board, or plug it on to a proto board using a 40-pin socket. This makes it easy to use.

- Two voltage regulators on the board: one provides the 2.5 VDC V_{CCAUX} voltage. The other provides the 1.2 VDC V_{CCINT} voltage. You provide the board with a 3.3 VDC supply.

FPGA Embedded Systems Architecture

My chosen architecture illustrates a simple embedded robotic application (Figure 2). This system controls a robot through a radio remote control, which uses steering and throttle channels.

The system comprises these IP blocks:

- The radio control receiver's PCM is decoded and fed to a PicoBlaze 8-bit microprocessor core.
- The PicoBlaze processor mixes the steering and throttle and sends the results to the PCM encoder IP. Mixing is required because this system is a differentially steered robot. The PicoBlaze embedded processor also has two UARTs (on future applications these will be used for a GPS and radio frequency modem interface). When reset, the system sends a "hello world" message out both ports. Then data sent to one port is echoed out of the other port. This way, the GPS telemetry could be sent to the radio frequency modem interface and vice versa.
- The information from the PicoBlaze processor is pulse-code modulated by hardware. This PCM signal is sent to the wheel drive units, which use these signals to control the speed and direction of the wheels.

This is a very simple system that you can use as a foundation on which to build. By using the custom IP, the robot interfaces are offloaded to pure hardware. This frees up the 8-bit PicoBlaze processor to be used for signal mixing and user-interface purposes. Note that you could also implement the signal mixing in pure hardware, freeing up even more processor resources.

Development Platform

The platform I use for all of my development work is a small differential steer robot I call "Slo-bot," shown in Figure 3. Slo-bot gets its name because it uses standard radio control servos modified for continuous rotation operation. This technique provides a motor speed controller and gear motor in a small package. The only problem is that the wheels turn slowly – hence the name Slo-bot. However, this is not a problem for a proof-of-concept system I drive around on my desk.

This system comprises:

- Two continuous rotation servos and wheels
- A NiMH 7.2 VDC battery
- A carrier board for the Spartan Stamp breadboard with voltage regulators for 5V and 3.3V DC on it
- Voltage dividers for the radio control receiver that output 5V DC signal levels, which must be dropped down to 3.3V DC levels
- Connectors for the continuous rotation servos

Conclusion

By using an inexpensive but powerful Spartan-3E FPGA with a PicoBlaze embedded processor, you can solve many problems in robotics quickly and inexpensively.

The FPGA offers the capability of designing any hardware interface you can imagine without bogging down the processor with details. The processor is free to do what it does best – parse strings, crunch numbers, and handle user interfaces.

With more advanced techniques, you could process complex discrete state-space equations in hardware. This would allow you to implement high-end Kalman filters and other signal processing equations at high speed without tying up the processor.

If you have any questions or suggestions, e-mail me at fpgarobotics@mindspring.com or visit <http://home.mindspring.com/~fpgarobotics/>.

GET PUBLISHED



WOULD YOU LIKE TO BE PUBLISHED IN XCELL PUBLICATIONS?

It's easier than you think!

Submit an article draft for our Web-based or printed Xcell Publications and we will assign an editor and a graphic artist to work with you to make your work look as good as possible.

For more information on this exciting and highly rewarding program, please contact:

Forrest Couch
Publisher, Xcell Publications
xcell@xilinx.com

