

Designing Control Circuits for High-Performance DSP Systems

These simple techniques could save you days of work.

by Narinder Lall
Sr. DSP Marketing Manager
Xilinx, Inc.
narinder.lall@xilinx.com

Brad Taylor
System Generator Applications Manager
Xilinx, Inc.
brad.taylor@xilinx.com

FPGAs have made significant strides as engines for implementing high-performance signal processing functions, whether for ASIC replacement or performance acceleration in the signal processing chain with DSP processors. Although much has been written about how to use FPGAs as signal processors, not much has been published about building control circuits within such systems.

There are perhaps two key decisions to make when implementing control circuits for FPGA-based DSP systems:

- Should the control circuit be implemented in hardware or developed as a software algorithm?

- What building blocks are available to make the development of the control circuit as efficient and painless as possible?

Software or Hardware?

In this first stage, you can make tradeoffs between algorithms that are implemented in hardware, and those that are better implemented in software using a soft microprocessor (Xilinx® PicoBlaze™ and MicroBlaze™ processors) or hard embedded microprocessor (PowerPC™ 405). Table 1 shows the tradeoffs between hardware- and software-based approaches.

A number of attributes need to be considered when making tradeoffs between these approaches. These include:

- **Algorithm complexity.** You can easily implement simple algorithms (like those that do not need many lines of C-code) in both software and hardware. Although no absolute measure exists to correlate how many lines of C-code represent one slice, a good

rule of thumb is that one line equals 1 to 10 slices. When algorithm complexity rises, implementing and testing the algorithm in hardware becomes more challenging. You can more easily implement complex algorithms in lines of C code on a microprocessor, which is the preferred route most designers choose.

- **Need for an RTOS.** If an RTOS is a mandatory piece of the control algorithm, this again favors a software approach that exploits the use of the hard embedded PowerPC on Virtex™-II Pro or Virtex™-4 FX FPGAs, or on external microprocessors. RTOS support for these microprocessors currently includes support from Wind River and MontaVista.
- **Communication with the host.** Communication with a host processor will often – but not always – require a bus architecture of some kind. In this instance, a microprocessor such as the

MicroBlaze processor or PowerPC processor is ideal, as both support bus architectures such as the OPB. Hardware-based host communication using state machines, although possible, can be somewhat more cumbersome.

- **Speed of decisions.** If you specify speed of decisions as clocks per decision, then for decisions that are needed quickly it is obvious that a hardware circuit will be preferred, if not required. For decisions that can be made in hundreds or thousands of clock ticks, software-based algorithms will be sufficiently capable to handle this level of performance.
- **Need for floating point.** Although floating point is largely tangential to control functions, cases do exist where systems employ floating point for control. One example is the calculation of filter coefficients. In sonar systems that require matrices to be inverted, floating-point control is often preferred, as it is often easier to develop with. Floating-point control is also preferred when control precision is high and the algorithms are not available in fixed point.

Once you've decided on hardware or software, you have access to a number of building blocks. Each one is particularly suited to different types of control tasks.

Types of Control Tasks and Possible Circuits

Many different types of control tasks exist. For this article, we have chosen to focus on the following types of problems, which are commonly found in signal processing systems.

- Hardware-Based
 - Data-Driven Multiplexing
 - Implementing Finite State Machines (FSMs)
 - Sample Rate Control
 - Sequencing – Pattern Generation
- Software-Based
 - Implementing Low-Rate Control Algorithms

	Clocks Per Decision	Algorithm Complexity	RTOS	Floating Point	Communicate With Host
Hardware-Based Control	1-10	Simple	No	No	Difficult
Software-Based Control	100-100000	Complex	Yes	Sometimes	Easy

Table 1 – Hardware/software tradeoffs

- High Complexity Control of Physical Layer Data Paths (MAC layer) (outside the scope of this article)

Table 2 shows a summary of the tools and types of typical control tasks. These are not hard-and-fast recommendations – merely some suggestions for some of the better options. As Xilinx System Generator for DSP is the tool of choice for modeling and designing DSP systems onto FPGAs, in this article we'll also provide some examples using free demos contained within System Generator that demonstrate the use of the control circuit.

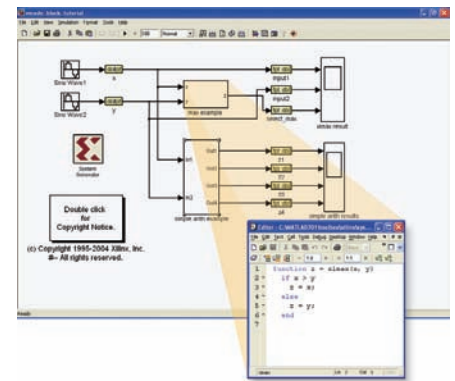


Figure 1 – Data-driven muxing using m-code

Task 1: Data-Driven Multiplexing

An example of a control task that does not require monitoring of the current state is data-driven multiplexing, in which data is monitored and tests are performed on that data. The results of those tests determine the output of the control circuit. Figure 1 shows an example of data-driven multiplexing. Here the function is determined by a simple MATLAB function called xlmax. Input y is selected unless $x > y$, in which case input x is selected.

Task 2: Implementing FSMs

Finite state machines are used when decisions must be made based on the current “stored” state of the input(s). For hardware-based high-performance DSP systems, it is not uncommon to see circuits where monitoring of state(s) is performed every clock tick.

Although you can implement them in many ways, perhaps the most common ways to implement FSMs within a System Generator design are through m-code CASE statements (popular with algorithm

Toolkit	Class of Control Problem				
	Sequencing	State Machine	Data-Driven Muxing	Sample Rate Control	Low-Rate Control Algorithm
Pattern Generation Components (ROMs, Expressions, Comparators, Counters, Delays)	X				
M-code		X	X		
Comparators/Muxes			X		
FIFOs/Clock Enables/Up/Down Conversion				X	
PicoBlaze					X
MicroBlaze/PowerPC 405					X

Table 2 – Types of control problems and tool kits available in System Generator

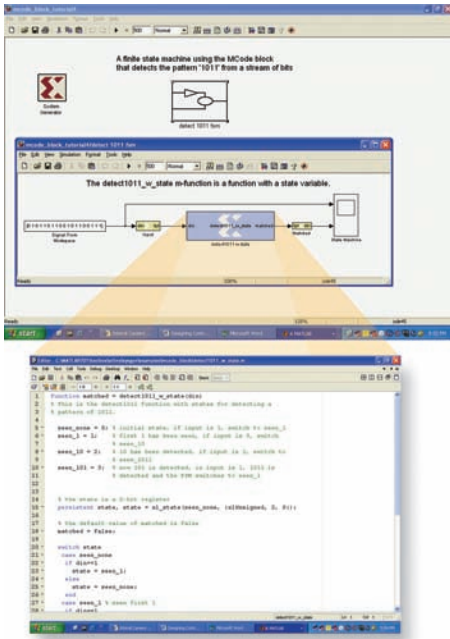


Figure 2 – Implementing an FSM in System Generator

developers) and writing HDL (preferred by hardware engineers). HDL can be easily incorporated into System Generator designs using a black box and co-simulated using ModelSim if necessary.

Figure 2 illustrates how easily you can implement an FSM in System Generator using the m-code block that pulls in a MATLAB script contained in the file detect1011_w_state. The purpose of this script is to detect a 1011 pattern from a signal passed through from the MATLAB workspace.

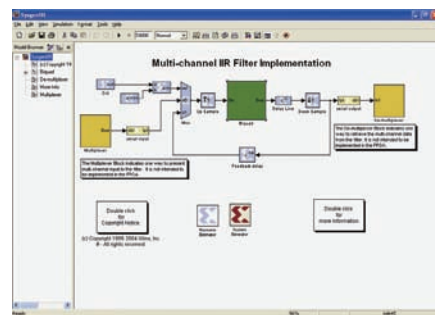


Figure 3 – Sample rate control

Task 3: Sample Rate Control

In high-performance DSP systems, samples often arrive into a system or a piece of a system at a different rate than that of the FPGA clock. We recommend that engineers therefore learn techniques for per-

forming sample rate control. Possible solutions within System Generator that facilitate the design of sample rate control circuits include up/down sampling, clock domains, FIFOs, and clock enables.

Figure 3 demonstrates how you can implement multiple IIR filters using a single time-shared second-order section (biquad). Specifically, 15 distinct IIR filters, each consisting of 4 cascaded biquads, are realized in a “folded” architecture that uses a single hardware biquad. Hardware folding is a technique to time-multiplex many algorithm operations onto a single functional unit (adder, multiplier). For low-sample-rate applications like audio and control, the required silicon area can be significantly reduced by time-sharing the hardware resources.

This design uses a number of control circuits, including a count-limited counter feeding into a two-input mux (that selects between the serial data and the feedback path) and up-sample and down-sample blocks (that control the data rate through the biquad).

Task 4: Sequencing (Pattern Generation)

Sequencing problems usually involve the need for a periodic control pattern that is predictable and not necessarily dependent on the current “stored” state. A common solution for sequencing is to use a simple pattern generator. You can build a pattern generator using building blocks like counters, comparators, delays, ROMs, or the logic expression block within the System Generator block set. The beauty of this underutilized technique lies in its simplicity – yet many designers often opt for more complex, unnecessary state machines.

An example of a pattern generator is contained in the biquad block in Figure 3. The address generator within the biquad block (not shown) generates all of the addresses of the RAMs and ROMs, as well as the write-enable signal for the single-port RAM in the folded biquad module.

Task 5: Low-Rate Control Algorithms

Implementing low-rate algorithms using a Xilinx microprocessor is becoming increasingly common. With a choice of three

mainstream processors – the PicoBlaze 8-bit processor, MicroBlaze 32-bit processor, and embedded IBM PowerPC 405 32-bit processor – you have the ability to scale depending on the task at hand. Common tasks that often necessitate the need for on-chip processors include calculating filter coefficients, scheduling tasks, detecting packets (such as in an FEC receiver), and RTOS implementation.

Figure 4 shows a simple control circuit built using the PicoBlaze microprocessor. This example forms the receive path of a 16-QAM demodulator that performs adaptive channel equalization and carrier recovery on a QAM input source. An

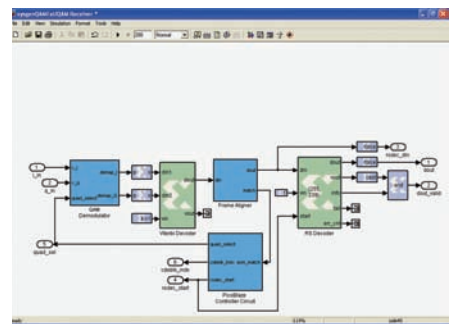


Figure 4 – QAM packet detection using the PicoBlaze microprocessor

attached synchronization marker (ASM) applied by the transmitter is stripped from the demodulated data before concatenated FEC is applied. The PicoBlaze microcontroller controls the RS decoder, maintains frame alignment of the received packets, and performs periodic adjustments of the de-mapping QAM-16 quadrant reference.

Conclusion

When implementing control circuits for high-performance FPGA-based DSP systems, you have access to a number of building blocks within System Generator to make this an easier task. Tables 1 and 2 list some of the tradeoffs to consider and summarize possible control solutions.

All of these designs – and many more – are included within the Xilinx System Generator tool, which retails for \$995. You can also try the tool free for 60 days by downloading the evaluation version at www.xilinx.com/systemgenerator_dsp.