HUNT ENGINEERING
Chestnut Court, Burton Row,
Brent Knoll, Somerset, TA9 4BP, UK
Tel: (+44) (0)1278 760188,
Fax: (+44) (0)1278 760199,
Email: sales@hunteng.demon.co.uk
URL: http://www.hunteng.co.uk

Implementing Filters with the HERON-FPGA Family

Rev 1.0 R.Weir 10-01-2001

Introduction

The HERON-FPGA family is ideal for many of the building blocks of digital signal processing (DSP) more typically implemented in DSP processors. They have a high gate count, often combined with peripherals – such as high-speed ADC or DAC converters. In signal processing systems, the FPGA can be used to implement functions like up-sampling (interpolation) or pulse shaping before transmission, or Digital Down Conversion (DDC) for a receive chain. Many other functions can also be implemented.

However, using an FPGA for DSP can appear daunting. Many developers view algorithms as a software problem, while the FPGA is considered "hardware" and therefore complicated. This is not the case.

Many systems can be developed on an FPGA in less time than required to code them for a DSP processor in software. No great in-depth knowledge of hardware is required to get a system working – in fact most software engineers can do it.

One of the most basic functions that can be implemented is the digital filter. We will use that as a basic example to show how a system can be built quickly and easily using FPGA – even without an in-depth knowledge of hardware.

Starting Out

We're going to assume that you've worked through the "Getting Started with Heron-FPGA" applications note. This gives an overview of working with the Xilinx tools, but doesn't show how easy it is to create powerful signal processing systems using Heron-FPGA.

The best place to start creating a filter is the Xilinx Core Generator tool. This can create many library components for us – including "Distributed Arithmetic FIR Filters". Start the Core Generator – it's under "Tools -> Design Entry -> Core Generator" – and find Distributed Arithmetic FIR Filters.

Double-clicking on this core will give you the options for the filter we will create. This is a powerful tool – it can create FIR filters with varying levels of performance, including sample rate changes, very quickly. The filters it generates seem efficient and compact.

The major options are:

- Filter Type. You can implement simple FIRs, or FIRs with a rate change – interpolating or decimating. In addition, you can select half-band filters, or even hilbert transformers – this flexibility should mean that the tool can generate the FIR you need.
- Number of channels. You can apply the same filter to many channels – for example, you might want to apply the same filter to 8 channels of audio. This option allows you to do this.
- Number of Taps & Coefficient File; specify how long you want the filter to be, whether it is a symmetric filter, and then load a file containing your filter coefficients. The tool will not calculate those coefficients for you – but there are many tools that will.
- Input data width & format. Specify how data is formatted when it arrives at the FIR's inputs. If the data is coming straight from an ADC, you may want to sign-extend it before sending it to the filter – for example, extend 12-bit ADC data to 16-bits for improved accuracy in the filter.
- Performance. This tool can generate FIR filters for a wide variety of applications. You can specify that the filter should generate a new sample on every clock cycle; but that is only sensible for very high speed applications. More generally, the filter will have several cycles available to it before an output is required.

  Try and specify as many cycles as possible for your filter. For example, specifying the highest performance option (1 cycle/sample) can use an order of magnitude more resource on the FPGA than specifying the lowest performance option.

- Latency & Registered Outputs. Many of the filter designs will be able to accept a new input long before the processing of the previous input has been completed. They do this by "pipelining" the filter's operation. For many applications, this doesn't matter – a delay of a few cycles is nothing in a wireless receiver's downconverters. However, it may be a point to consider if you are building a control application, in which case you may want to minimise this delay.

  The "registered output" option is worth selecting for ease of design. It's not strictly necessary; the register holds the filter's output, so that you can read it at any time after the filter finishes. Without it, there's a limited window in which you must read the value. Using it adds marginally to the filter's latency and complexity, but makes the rest of your design easier…

It is worth noting down the options you have used for this filter. You may want to modify the filter later, and the only way to do this is to re-enter these parameters. Once you've done this you can generate the filter – a process that may take a few moments.

"Linking" the FIR into your system

The FIR is now in your project's library. You can use the schematic editor to place it in the design. Do this as follows:

- Using the "bus" tool, draw a bus linking the data source to the FIR's input. A second bus should link the FIR's output to wherever the data is going next.
- Connect the FIR's CLK input to the system clock.
- Connect a "sample clock" to the FIR's "New Data" input. This sample clock is generated by your data source – for example, the ADC, or a FIFO; or another signal processing block which is acting as a data source. It indicates that a new sample is ready for the filter.
- Use the filter's "Ready" signal to indicate a new output is available. This can be used as a sample clock for the next signal processing block, or for a FIFO if you are passing data to the HEART communication system.

Watch out for asynchronous systems; you may want to halt the FIR's processing should an output FIFO become full. In that case, don't assert "New Data" until the output FIFO has space. You will need to consider how this affects the source of the data – it may be better to let the FIFO overflow if the data source has a fixed sample rate – like an ADC.

That's all there is to it! You can now save the design and select "Implement" – the FPGA equivalent of compiling and linking the project. When you load this code onto an FPGA module, you will have created a very high-performance filter – in a matter of minutes.

Other Types of Filter

The FIR is a commonly used filter, but of course there are many others. Two of great importance are the IIR filter, and the CIC filter.

IIR filters can be represented as two FIRs, one filtering the input, and one filtering the output - the reverse path. The results of the reverse path are added to the input. In this way, it is possible to construct IIR filters using only FIR filters and adders – which can be found in the Virtex library.

CIC filters are very high-performance filters using only adds and delays. More details can be found in the applications note "Implementing Digital Down Conversion in the HERON-FPGA Family" – but for here, suffice to say that they can be implemented with accumulators, adders and registers – all of which are available in the Virtex library. An example CIC is also available from Hunt Engineering – it's included in the DDC white paper.

How easy is it compared with developing C code?

For many things, developing for the HERON-FPGA family is faster than the equivalent C code for a DSP. For example, the FIR filter example we used here can be implemented and be running in a matter of minutes. A C-code solution would take significantly longer.

This isn't always the case, but it certainly isn't true to say that C code is easier than FPGA programming – in many cases, the reverse is true as we've seen here!