DSP Design Flows For Microsemi FPGAs

Table of Contents

Introduction . . . 1
Overview of DSP Solutions . . . 2
Development Flows . . . 4
Types of Core Generated . . . 14
More Development Flows . . . 15
Summary . . . 15
References . . . 16

Introduction

Digital signal processing (DSP) occurs in communications, audio, multimedia devices, imaging and medical equipment, smart antennas, automotive electronics, MP3 players, radar and sonar, and barcode readers. This algorithm can be implemented in Microsemi® system-on-chip (SoC) Products Group flash based field programmable gate array (FPGA), mixed signals FPGA, and also radiation-tolerant FPGA. This application note describes the generic DSP design flow in FPGA and the advantage of using Microsemi FPGA.

The following sections are covered in this application note:

- Overview of DSP Solutions
  - DSP platforms
  - Parallel Computing vs. Turing Machine
  - Practical considerations
    - Choosing between microprocessors and FPGA
    - Choosing between ASIC and FPGA
    - Choosing between FPGA vendors
- Development flows
  - Typical DSP development flow
  - FPGA design flow
  - Intellectual property (IP)
  - MATLAB interface
    - Algorithm design
    - Conversion to Fixed Point
    - RTL design generation
    - Physical design implementation
  - DSP-centric FPGA design flow
  - Design flow for Mixed Signal FPGAs
    - Nonvolatile on-chip storage
    - Handling multiple channels of data stream
    - Context management for adaptive coefficients
  - Types of core generated

© 2012 Microsemi Corporation

- More development flows
  - High-level flows
  - 'Manual' implementation or back to FPGA design
- Summary
- References

Overview of DSP Solutions

DSP Platforms

The DSP algorithms can be implemented in many ways. The following are the most popular:

- Microprocessors/Microcontrollers: General-purpose microprocessors (for example, Pentium) and general-purpose microcontrollers (for example, 8051) can run DSP algorithms of arbitrary complexity.
- Programmable DSP chips or DSP microprocessors (uPs): The internal structure of DSP microprocessors are optimized to run many DSP algorithms much faster and more efficiently. For example, the DSP chips have one or more built-in fast hardware multiplier-accumulators (MAC) to perform MAC operations that are used heavily by DSP algorithms.
- FPGA: An FPGA can be configured to run a particular DSP algorithm, thereby dedicating FPGA resources to particular DSP tasks. Also, the FPGA can run hundreds of MAC units in parallel. As a result, the performance may far exceed that of DSP microprocessors.
- ASIC: An ASIC offers even higher levels of "dedication" than the FPGA. ASICs are champions while comparing performance per square millimeter of silicon. It is important to note that the gap between the ASIC and the FPGA tends to narrow as the FPGA grows in size (for example, larger than 1 million gates).

In addition, FPGAs are also used to implement partial DSP algorithms, i.e., part of the DSP algorithm was done in the FPGA and the other part was done in software using a microprocessors or microcontrollers.

Parallel Computing vs Turing Machine

There is a big performance difference between the DSP platforms based on how the platform performs computations. Both general-purpose and specialized DSP microprocessors belong to the class of Turing machines, which perform instructions one at a time. For example, to add two numbers A and B, the turing machine would need to do something similar to the following:

- Fetch instruction 1 and decode it
- Execute the instruction 1, i.e., fetch data A and put it in the accumulator
- Fetch instruction 2 and decode it
- Execute the instruction 2, i.e., fetch data B and add it to the accumulator
- Fetch instruction 3 and decode it
- Execute the instruction 3, i.e., put the accumulated result where it needs to be

The FPGA and ASIC are 'deprived' of this limitation. In fact, there are a few flexibility and performance limitations a modern FPGA puts on a system developer. The FPGA can run parallel processing (i.e., execute multiple instructions at a time); implement turing machine as needed, including instantiation of the soft microprocessor; and carry virtually any practical combination of parallel processors and turing machines on the same silicon. The parallel processing dramatically improves performance of common DSP functions, such as FIR filter, FFT, and correlator.

Figure 1 shows a 4-tap FIR filter structure. While the microprocessor needs to run the computations one by one, the FPGA instantiates all the necessary components; four multipliers, four adders, and three delay elements which enables them to work in parallel. As a result, such a structure can process a new input sample every clock period as compared with the 8 clock cycles per data sample required by the microprocessor.

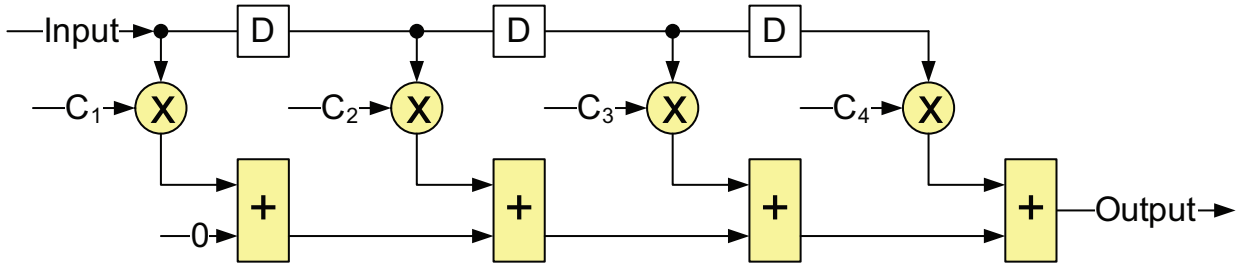

***Figure 1* • FIR Filter Structure**

Not every DSP algorithm can efficiently utilize parallel processing. An IIR filter is an example of such a category. On the other hand, there are several techniques, such as CORDIC or error-correction algorithms where the FPGA technology, despite the limited application of parallel processing, has been proven to be more efficient than the DSP microprocessor.

The DSP and general-purpose processors are trying to catch up with parallel computation machines. In some cases, modern DSP microprocessors can perform a few instructions at a time, such as certain HW co-processors or accelerators (for example, Viterbi decoder and FFT engine). But, the FPGA does not sit quietly as it too can carry “soft” microprocessor, thereby enjoying all the benefits provided by Turing machines.

Practical Considerations for DSP Applications

Choosing Between Microprocessor and FPGA

If the FPGA is good, you can buy a microprocessor to run a DSP application.

First, microprocessors have a longer history than FPGAs. Further, the necessary support infrastructure has been developed over time which includes compilers, assemblers, and automatic converters from high-level language to assembly code and extensive libraries. At some point, almost any practical DSP application could be implemented on the microprocessor theoretically, so that one could grab a ready, off-the-shelf implementation. Additionally, many new DSP algorithms emerge as software routines so they are pretty much ready for the microprocessor platform.

Second, dealing with the FPGA requires a different set of skills than those common in the DSP community. While deciding which platform to choose, the thumb rule: if a general-purpose processor can keep up with the specification, stay there. If not, pick the DSP processor when it meets the necessary MAC rate.

Finally, if one needs outstanding performance, FPGA is the best choice. Again, the challenge is that the number of software development experts far exceeds the number of DSP microprocessor programmers, which in turn is larger than the number of the FPGA designers capable of implementing the DSP algorithms. Even with appropriate FPGA expertise available, creating a good DSP design that capitalizes on the FPGA’s major benefits is a time consuming and elaborate process.

Choosing Between ASIC and FPGA

FPGAs offer the same advantages as ASICs, such as reduction in size, weight, and power dissipation, higher throughput, better design security against unauthorized copies, reduced device and inventory cost, and reduced board test cost.

ASICs lose to FPGAs when it comes to reduction in development time by a factor of three to four, ability to modify the configuration including remote in-circuit programmability and lower NRE costs that a customer pays prior to obtaining an actual ASIC device.

Choosing Between FPGA Vendors

Microsemi antifuse and flash-based FPGAs have several benefits compared to the other FPGA vendors, such as SEU immune, low power, and security. Microsemi antifuse and flash-based FPGAs are not susceptible to configuration loss due to single event errors (SEE) caused by alpha or neutron radiation, whereas SRAM-based FPGA can have upsets due to neutrons and alpha particles. The FuseLock and Flashlock advantages ensure that unauthorized users cannot read back the contents of Microsemi antifuse and flash-based FPGAs.

These features provide a key advantage while using Microsemi FPGAs in system critical or Hi-rel application. Microsemi RTAX-DSP space-flight FPGAs have embedded radiation-tolerant 18 bit x18 bit DSP blocks in addition to radiation-tolerant standard logic and ram block available in the tried and tested industry standard, RTAX-S product family. The DSP blocks allow dramatic increase in device performance and utilization while implementing arithmetic functions for DSP algorithms in RTAX-DSP. For more information about RTAX-DSP, refer to:
www.microsemi.com/soc/products/milaero/rtaxdsp/default.aspx.

Development Flows

Typical DSP Development Flow

Figure 2 on page 5 presents a typical DSP development flow chart. Usually, an algorithm makes its first appearance as a floating-point software model. The algorithm gets tested, evaluated, and verified using an appropriate test bench. It is worth noting that the test bench development often takes the same or even more effort as the algorithm design. Floating point representation lets algorithm creators take advantage of high precision computations while not caring about dynamic range. At this stage, the algorithm is independent of implementation.

As soon as the algorithm is found to be useful, it needs to be converted into the fixed-point representation. Implementing floating point calculations directly is possible in principle but takes a huge amount of silicon resources, and/or making the computation rate painfully slow. This stage is implementation dependent. In many cases, the conversion is not a straightforward process. On the contrary, it may take several iterations and experiments to obtain acceptable results. Whoever does the conversion, the person or group needs to possess a good knowledge of the algorithm and reasonable familiarity with the implementation platform.

After the fixed-point algorithm gets verified, the actual implementation can start. Until that point the DSP architects use one of the common programming languages like C/C++, or an environment supporting a higher level of abstraction, like the MATLAB-Simulink package from Mathworks. In order to implement the algorithm on the FPGA, it needs to be handed over to the FPGA designers.

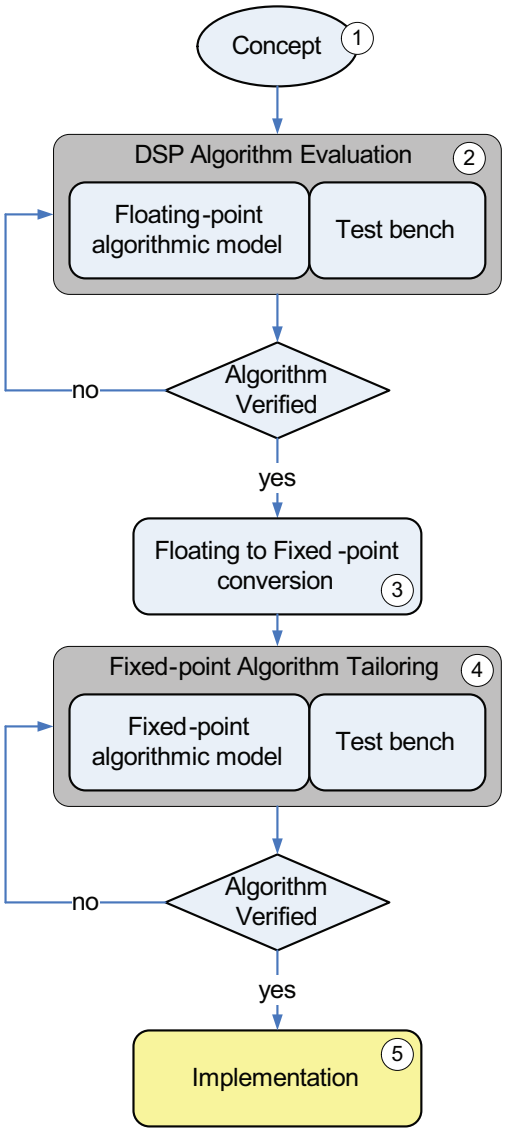

***Figure 2* • Typical DSP Development Flow**

FPGA Design Flow

Figure 3 on page 6 depicts an FPGA design flow. It starts with a design capture stage, which is usually the most time consuming and skill demanding portion of the design. The way design engineers typically envision their domain is a collection of blocks described in Verilog or VHDL captured at a register transfer level (RTL) [1]. An even more important fact is that simulation and synthesis tools used in the FPGA design flow expect the RTL design entry. Therefore, the DSP algorithm needs to be converted into the HDL RTL. Along the way the designer has to “unroll” the algorithm to make it suitable for parallel synchronous processing. Often this is not an easy task and requires familiarity with efficient VLSI DSP structures [4].

Obviously, manual conversion is quite a time consuming process and is prone to errors. One key concern is that there is no clear handoff between the DSP architect and the hardware design engineer working in the implementation domain. To avoid this scenario, it requires an engineer who is an expert in both domains and such people are “few and far between” [1].

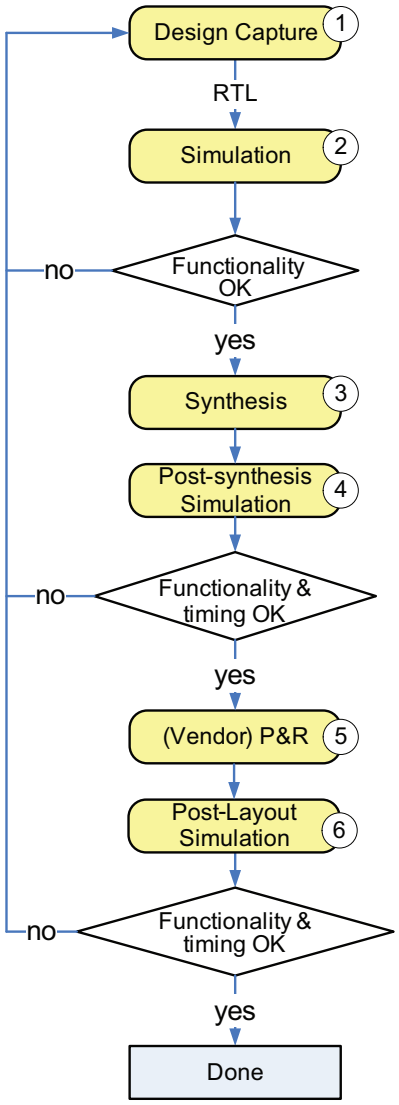

***Figure 3* • Typical FPGA Design Flow**

Fortunately, there is a growing variety of ways to mitigate the problem of implementing DSP design in FPGA hardware.

Intellectual Property (IP)

FPGA and third-party vendors create highly parameterized HDL models that implement some popular DSP functions such as FIR filter, FFT, CORDIC, etc. Microsemi provides CoreFFT, CoreFIR, CoreDDS, and CoreCORDIC DirectCore IPs as part of the Libero® Integrated Design Environment (IDE) design flow tool. These IP cores are supported in Microsemi flash based FPGA, mixed signals FPGA and also radiation-tolerant FPGAs. In addition, these IP cores have special features for RTAX-DSP radiation-tolerant FPGA as RTAX-DSP FPGAs has dedicated DSP blocks, Table 1 provides the overview of these DSP IP cores.

In addition to these DirectCore IPs, there are several third party DSP cores available. Please refer to the Microsemi SoC Solutions and IP Catalog.

Table 1 • **Microsemi DSP Cores**

IP Core Name	Description
CoreFFT	• Highly parameterizable DirectCore RTL generator optimized for the RTAX-DSP family supports forward and inverse complex FFT • Transforms sizes from 32 to 8,192 points • 8 to 32 bits I/O real and imaginary data and twiddle coefficients • Two's complement I/O data • Bit-reversed or natural output order • Selection of unconditional or conditional block floating point scaling • Embedded RAM-block-based twiddle LUT • Built-in memory buffers with optional extensive or minimal memory buffering configurations • Handshake signals to facilitate easy interface to user circuitry • Radix-2 decimation-in-time in-place and radix22 decimation-in-frequency streaming FFT implementation for RTAX-DSP
CoreFIR	• Highly parameterizable RTL generator • Supports up to 1,024 FIR filter taps • Fully enumerated (parallel), single rate folded (semi-parallel) filter and multi-rate polyphase interpolation FIR filter implementation for RTAX-DSP
CoreDDS	• Fast switching • Fine frequency resolution • Broad frequency operation • Generates complex or real-valued sine or cosine waveforms • Three configurable hardware architectures
CoreCORDIC	• Vector rotation—conversion of polar coordinates to rectangular coordinates • Vector translation—conversion of rectangular coordinates to polar coordinates • 8-bit to 48-bit configurable word size • 8 to 48 configurable number of iterations • Parallel pipelined architecture for the fastest calculation • Bit-serial architecture for the smallest area

Thus, when the hardware engineers encounter a "standard" function, or core in the algorithm, they simply parameterize (configures) the one and instantiates it in the overall design. Needless to say, the IP cores have already been tested and verified by the vendors. In addition, these are often supplied with appropriate test benches that may serve as foundation for the larger test bench covering the entire design.

IP also provides other valuable benefits:

- An IP core is a hardware module designed to be used by a hardware engineer. Because a majority of the FPGA designers favor IP cores.
- The IP based FPGA design makes the algorithm handoff much easier since the FPGA designer does not have to get to the bottom of the DSP function. This allows for some detachment of the algorithm development and its FPGA implementation. From the business organization standpoint that might be beneficial.

- IP usually offers “fine tuned” implementation options, for example, fixed-coefficient FIR filter, distributed arithmetic (DA) FIR filter implementation, MAC-based filter, etc. These differ significantly by the amount, and their type of resources utilized and performance. For example, the MAC-based FIR filter utilizes multipliers and is good for multiplier-rich parts. The DA FIR filter, to the contrary, is a clever “multiplierless” structure, primarily utilizing RAM blocks; therefore, the designer can select a solution that fits the design needs the best.
- IP developed for a particular platform takes advantage of the FPGA type architecture.

MATLAB Interface

Looking at Figure 2 on page 5, a keen mind may figure out the fact that Step 4 yields a comprehensive algorithm description in some form, for example, C-program, MATLAB code, or a state diagram. Is there a way to convert that description into HDL automatically? Or, is it possible to bypass the design capture stage of the FPGA design flow (Figure 3 on page 6), and bridge the latter with the DSP development flow. The answer is yes in many cases. Further, the test bench can also be converted into the HDL description or another form the simulation tool can accept.

It is quite common for today’s FPGA industry to provide one or another form of the MATLAB – FPGA interface [2]. These can differ significantly by the implementation and even philosophy but from your standpoint they look somewhat similar. The design flow follows the steps 1 to 4 shown on Figure 2 on page 5. The DSP architect evaluates the algorithm at a high level of abstraction using the MATLAB-Simulink package. The environment offers advanced features, such as a smart and convenient visualization facility, extended DSP libraries, automated test bench facilities, and a nice user interface.

In many cases, the DSP architects are not familiar with the RTL-based design flow. It’s a challenge for FPGA vendors to conceive a flow that enables the DSP architects to leverage familiar MATLAB and Simulink environments, while transparently exercising the FPGA design flow. There has been significant progress made and the new Simulink-Synphony-Libero IDE flow allows transparent design flow. In this new generation of DSP-FPGA tools, significant advantages are offered to you through a what-if scenario analysis capability. You can analyze multiple options and make tradeoffs at every stage of the design flow. Figure 4 shows the four stages of iterative design flow, enabling you to evaluate the different possibilities on hand.

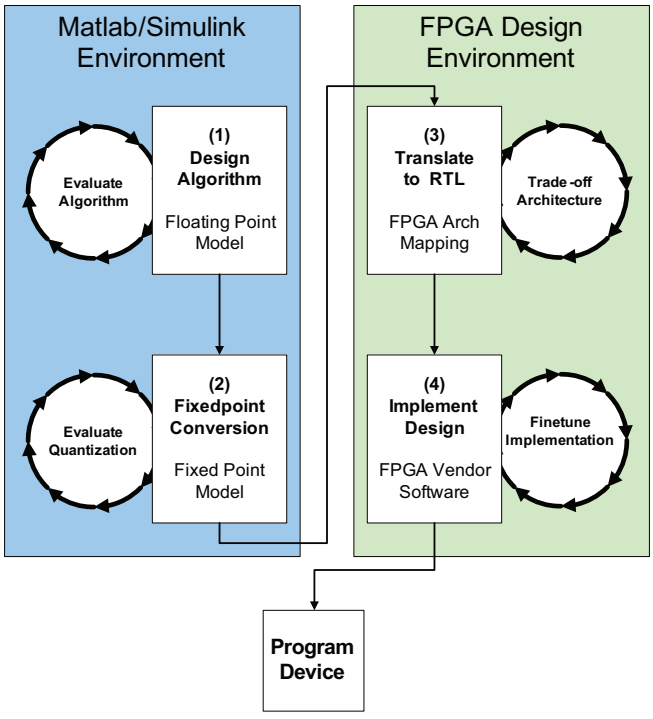

***Figure 4* • MATLAB/Simulink Integration in FPGA Design Flow**

Algorithm Design

From a concept, a DSP architect translates the ideas into a design in the MATLAB and Simulink environment. At this stage, DSP blocks are supplied by the Synphony Model Compiler. Refer to the Synphony Model Compiler User's Guide for the available block set and creating customer block set at www.microsemi.com/soc/documents/SynphonyModelCompilerAE_UG.pdf.

Figure 5 shows an example of how to create a design in Simulink and use the Filter Design Analysis tool from MATLAB. The design capture can start by dragging in the desired blocks and connecting them up to realize the desired function. Once the design capture is done, the filter tool provides a convenient facility to analyze the functional behavior of the filter. One of key point is that you need to create the whole design or atleast the desired blocks using Synphony Model Compiler blockset so that the HDL code can be automatically generated.

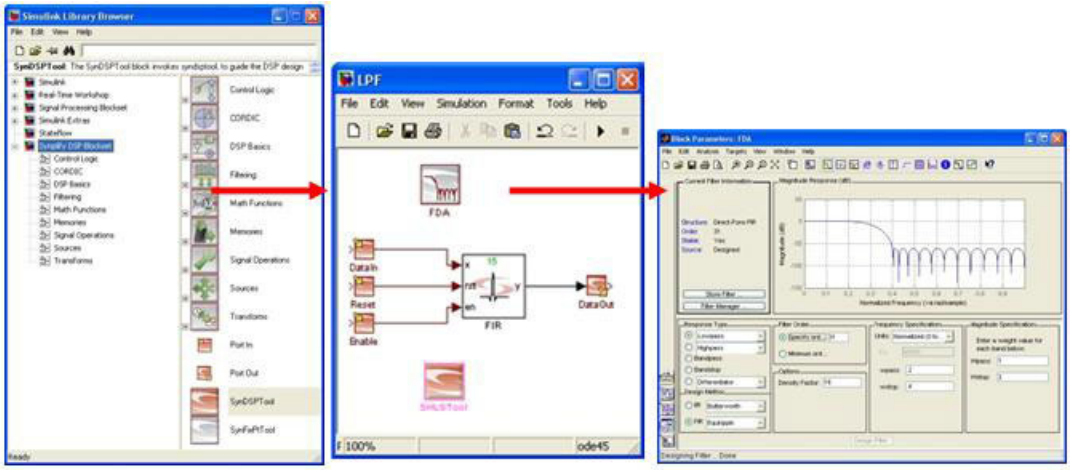

***Figure 5* • Algorithm Development in MATLAB and Simulink Environment**

Conversion to Fixed Point

Simulink provides an environment to simulate the design and analyze its behavior using floating point and fixed point accuracies. Simulation can be performed using the built-in stimuli and scope block sets in Simulink. The floating point format provides a baseline performance of the algorithm that helps in the analysis of the fixed point behavior of the design. As shown in Figure 6, the fixed point tool in Synphony helps to automatically change the accuracy of the data and the effects of the tradeoffs can be easily viewed in the scope.

Note: This conversion is done in a very short amount of time, that is the HDL code does not have to be changed or modified to re-simulate different tradeoffs.

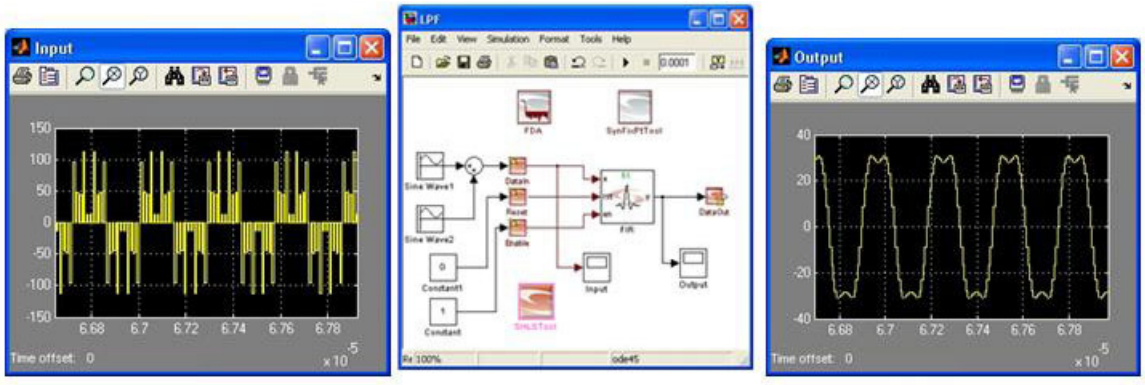

***Figure 6* • Conversion to Fixed Point and Tradeoff Analysis**

RTL Design Generation

The Synphony Model Compiler from Synopsys offers various optimization strategies as shown in the Figure 7. You can try different design tradeoffs to meet system performance easily by selecting the right options.

Folding: This optimization strategy helps in reducing the area utilization by reusing the same area hardware components (like multipliers) for multiple streams of data. This results in a very compact design. This optimization is a tradeoff between the area utilization of the hardware and the higher clock rate to maintain similar data rates.

Retiming: Retiming optimization is similar to the register balancing optimization done at the RTL Synthesis level; in this case, the optimization is done at the system level.

Multi-Channelization: Once a DSP algorithm has been developed and verified in Simulink, the design can then be replicated over multiple channels. This optimization automatically creates the mux logic required for passing the input data stream over the corresponding channels.

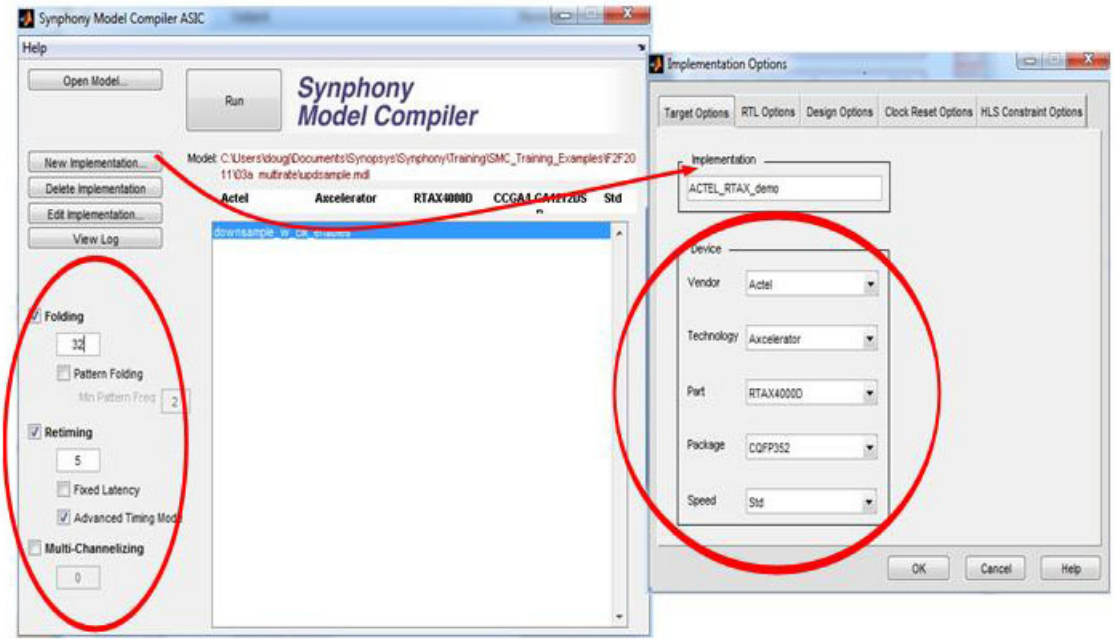

***Figure 7* • RTL Design Generation Using Synphony Model Compiler**

Physical Design Implementation

Figure 8 shows the FPGA implementation environment where the RTL design generated from Synphony is synthesized, simulated, and mapped to the FPGA device. The design flow has been made easy for you through the flowchart shown in the tool (Figure 8). You must click the relevant buttons in the flow to complete the task.

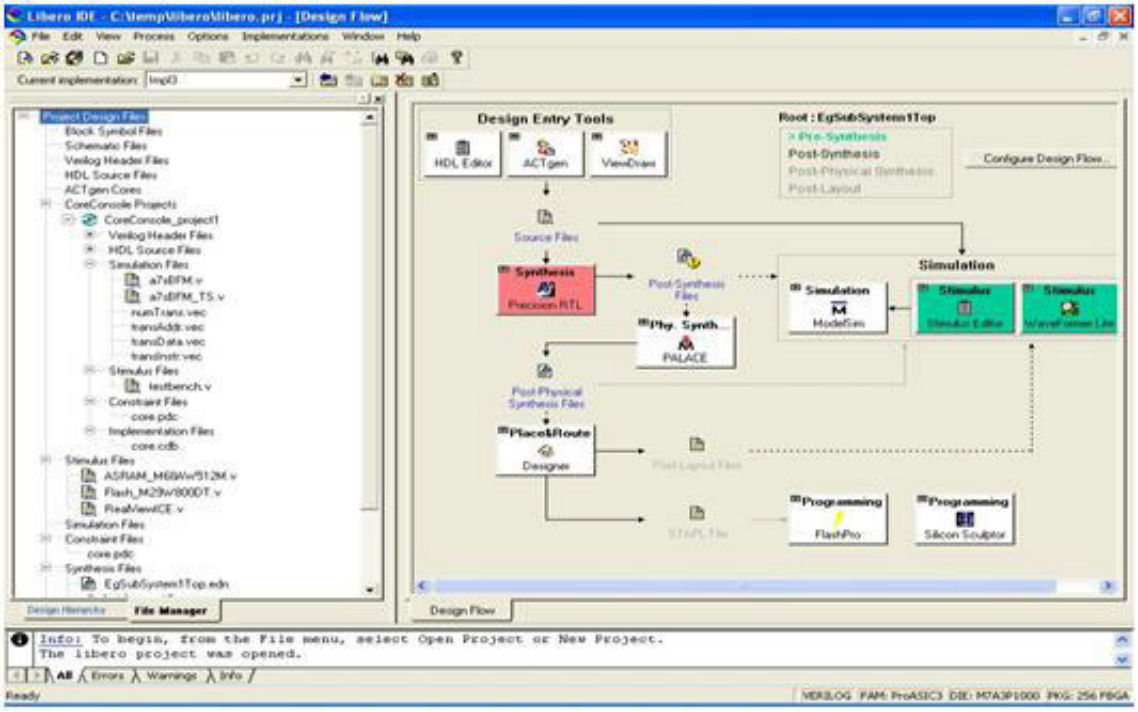

***Figure 8* • FPGA Design Implementation in Microsemi Libero**

DSP-Centric FPGA Design Flow

The Microsemi Libero IDE tool is not only limited to performing the physical design only but also supports the overall standard FPGA design flow. In conjunction with vendor optimized DSP tools and IP libraries, it creates a unified flow, as depicted in Figure 9 on page 12.

Following the flow, the DSP architect can create a DSP design in the MATLAB and Simulink environment, then convert the floating-point representation into the fixed-point, optimize the latter, and verify the result. To evaluate the filter, the architect can create a test bench using the available pre-developed test modules, signal generators, scopes, signal analyzers, etc. Once the filter has been verified, you are given an opportunity to enter implementation specific configuration parameters, such as the desired clock rate, format of the input and output data, and precision of the results. Finally, after pushing a button, the RTL model gets generated.

Thus, the systems architect who primarily develops the DSP algorithm can implement one on the FPGA with no or little help from an FPGA designer. If this is not the final implementation, then a physical model that can be tested in real time gets delivered much sooner.

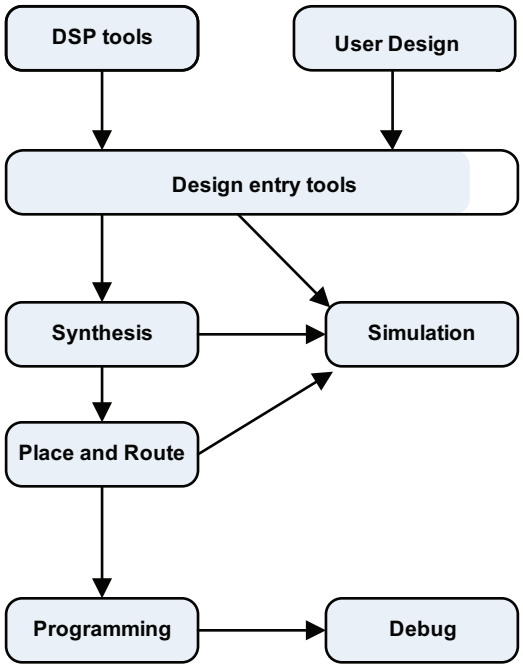

***Figure 9* • DSP-centric FPGA Design Flow**

This works because at the backplane of the MATLAB interface there are two libraries: one used by the MATLAB-Simulink (called a “block set”), and another containing parameterized hardware IP for all the components of the first library. By clicking the button, you can transfer algorithm configuration and parameters to the matching IP.

Note: The final implementation is only as good as the pre-designed IP cores.

Design Flow for Mixed Signal FPGAs

FPGA vendors have been adding SoC blocks on the FPGA fabric to deliver better value to their customers. The emergence of mixed signal FPGAs, such as the SmartFusion® cSoC device or Fusion Programmable FPGAs from Microsemi, has opened new doors for new design methods. The versatility offered by on-chip flash memory and a large number of analog input channels leverage new techniques in DSP system design. Some of the methods are outlined below.

Non-Volatile On-Chip Storage

Majority of common DSP applications make some use of fixed data sets. For example, the folded and polyphase FIR filters store filter coefficients and FFT utilizes sine/cosine tables. Some sophisticated DSP algorithms can be efficiently implemented as the look-up tables with relatively simple logic around them. For a single-chip DSP solution on the FPGA, such applications require that data sets can be loaded from a predetermined location. The SmartFusion cSoC and Fusion offer a large amount of on-chip flash memory to store the data. Each set can be considered as a context based on the condition in which the application or the core is used. The appropriate context containing the necessary data set can then be loaded. Multiple tables can also be loaded from a single flash memory block.

The IP block to load information from the table is supplied by the vendor. The entire design can be realized with a single chip, thus saving board space and overall design cost.

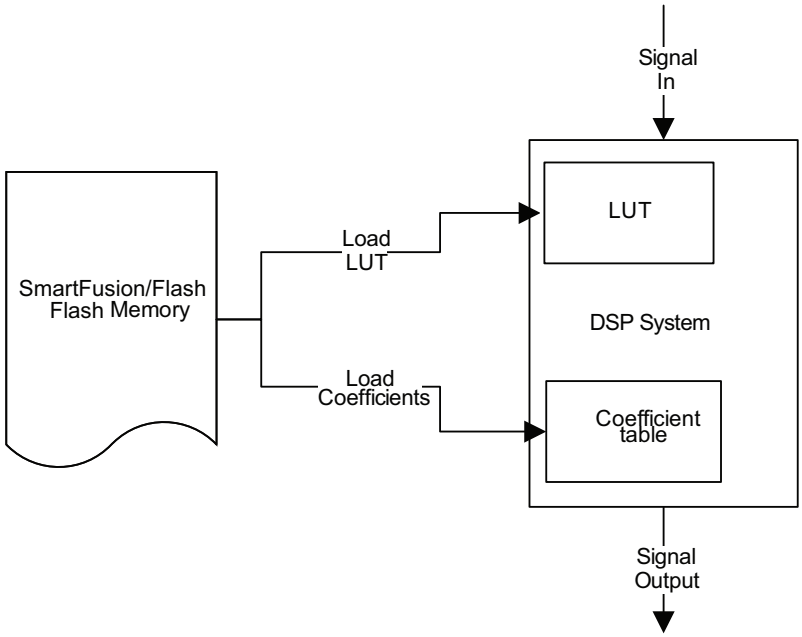

***Figure 10* • Loading look-up Tables From Flash Memory**

Handling Multiple Channels of Data Stream

Designs that involve processing data from multiple data streams can create architectures on the fly leveraging multi-channelization tools. You can create a filter, and once satisfied by its performance, can quickly invoke the special option to create a design that handles multiple channels.

Figure 11 shows how the data streams coming in from multiple on-chip analog channels can be processed through a filter, which has been multi-channelized in the Synopsys Synphony tool.

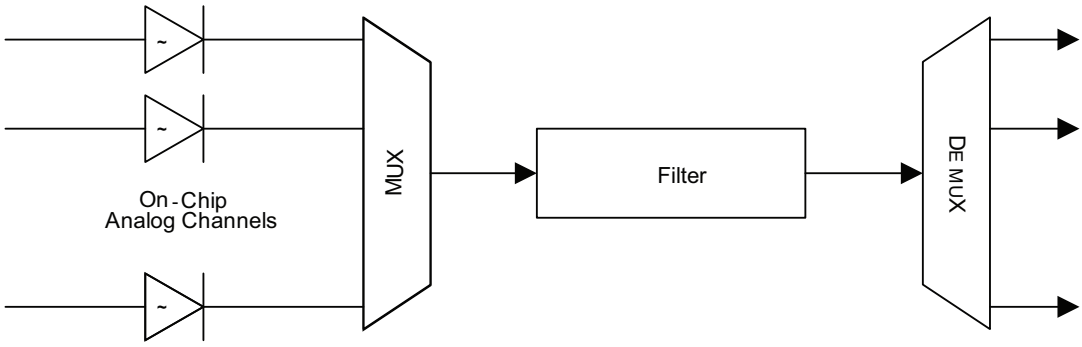

***Figure 11* • Multi-channelization of Analog Channels**

Context Management for Adaptive Coefficients

In adaptive filters, the coefficients that are tuned towards specific scenarios can be safely stored and retrieved from the on-chip flash memory available. Figure 12 shows one such case where the signals are subjected to different scenarios and the corresponding adapted coefficients are stored for later use. Extending this concept with clever use of multiple contexts adds significant value for this one-chip DSP solution.

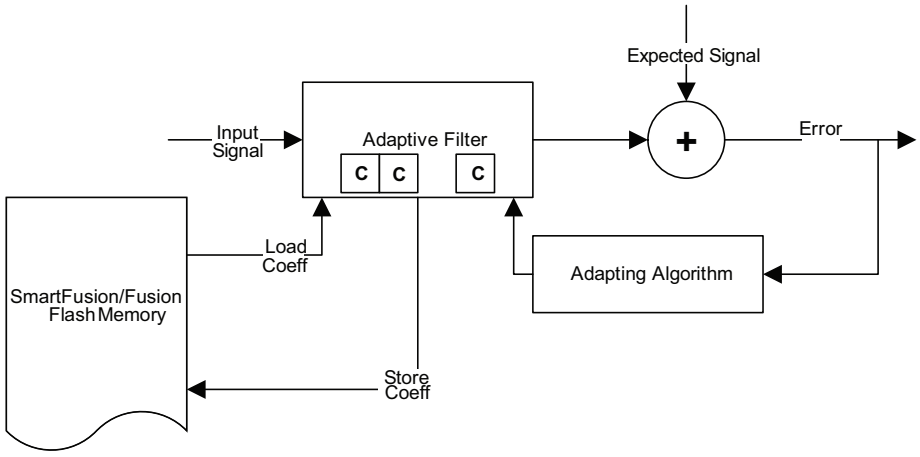

***Figure 12* • Load and Store of Adaptive Coefficients**

It's clear that the on-chip flash memory and other features in SmartFusion/Fusion FPGA features can be leveraged to design new techniques in DSP processing as demonstrated by the above examples. These device features, when supported with tools that are friendly towards DSP architects, make the design space exploration an easy task. Even DSP architects, who are not skilled in RTL design methods, can create compact and high performance DSP designs on FPGAs through these new generation of DSP design generators.

Types of Core Generated

It is likely that engineer who struggle with the density of a design may want to cut off some unnecessary stuff from a generated core. They may also want to access a few core internal signals to use them elsewhere in the entire design. But the core generated by the IP, either a direct one or using MATLAB interface, often does not provide a designer with full control over its contents. Driven by the desire to protect their IP, some (but not all) vendors limit the designer's access to the internal workings of the core.

In general, a core is delivered in three different ways; netlist or encrypted netlist, automatic RTL code, or "Human" RTL code.

The first category prohibits any kind of access to the core. The core itself is not portable, i.e., it applies to the specific vendor products and most likely to some particular parts. The second category does not expect you to look inside the core because the automatically generated code is really hard to read. The code generated though is portable. Finally, the third category generates hand made readable RTL code which gives you full access to the internal workings of the core.

Consider this example. Many, if not all, popular FFT cores utilize sine/cosine LUTs. Upon power-on, the sine/cosine table gets loaded in the on-chip RAM. Obviously, prior to that, a pre-calculated table sits on an external PROM. With Microsemi SmartFusion/Fusion one-chip solutions, the table has to be initialized internally (i.e., the design carries an internal sine/cosine table generator). The speed is not an issue here since the initialization takes place upon power-on. The CORDIC algorithm provides a good solution for a small and slow (in hardware terms) table generator. If the designer have access to both cores, FFT and CORDIC, they can easily combine the two if both cores are generated in 'Human' RTL.

There is always a trade-off between size/speed and time-to-market. Clearly, the direct IP and MATLAB interface aim at shortening the design cycle. Such rapid prototypes may serve as a final solution in many cases. But, in other cases, more or less significant improvement might prove to be necessary.

The worst case scenario is that you need to redesign some or all the “black box” cores, meaning you need to start designing from scratch. However, if the cores are open to a qualified user, they can be used as a platform for further improvement rather than dumping the whole prototype.

More Development Flows

High-level Flows

The MATLAB-Simulink interface is not the only direction EDA vendors are looking at. Another approach, called electronic system-level (ESL) design, is said to specify a system in an implementation-neutral language with the push of a button, and out would emerge full, detailed hardware design and corresponding software (http://www.us.design-reuse.com).

SystemC also includes both software and hardware concepts. SystemC supports designing both the hardware and software components together as these components would exist on the final system, but at a high level of abstraction. System Verilog extends the original Verilog towards system-level description and modeling.

‘Manual’ Implementation or Back to FPGA Design

Manual DSP design faces all the traditional design challenges as well as a few specific to DSPs. DSP-rich designs are often much larger than regular ones. Obviously, it is much easier to create designs out of a variety of building blocks. To help designers meet these challenges, all kinds of highly efficient, well-optimized libraries should be at their disposal from large IP-supporting DSP functions, peripherals and soft processors, to relatively small components, like counters or adders.

DSP designs require plenty of multipliers. Some FPGA parts like RTAXS-DSP, in addition to regular fabric and RAM blocks, carry multiple hard multipliers with more sophisticated MAC units. The issue here is that the multiplier instantiation is not always straightforward. For example, not every multiplication sign on a Verilog code has to call the hard multiplier instance. Instead, it may call for a constant coefficient multiplier, which implements nicely as one or a couple of adders. Another example is a barrel shifter. The one implemented on a regular fabric may cause a speed bottleneck.

Being built out of a spare hard multiplier it exhibits excellent speed characteristics and saves a lot of fabric resources. Synplify synthesis tool available in Libero IDE allows mapping the multiplier to DSP block and thus allow mitigating the problem.

Sometimes design density is less important or a designer is ready to trade it for the time-to-market reduction. In other cases it presents a non-trivial, elaborate, and time consuming problem. Imagine a wireless base station design that uses only off-the-shelf IP. A given FPGA can accommodate say, 10 of those communications channels. After some manual optimization as a result of an efficient resource sharing, the same part can accommodate 20 channels. Now, after manual intrusion in place-and-route, or floor planning, 40 channels can fit in the part. The final optimized design, though a time consuming one, reduces the overall number of FPGA parts required, the power consumption of the whole station, its weight and size, and sometimes eliminates the need for a cooling subsystem, etc.

Summary

Among many existent DSP platforms, FPGAs provide unmatched computational power, flexibility and high reliability like Microsemi RTAX-DSP FPGAs. Microsemi offers several design flows that speed up the FPGA implementation and make it less error prone. A variety of well designed and tested DSP IPs and libraries has significantly improved productivity of the Microsemi FPGA design flow.

References

[1] True DSP Synthesis for Fast, Efficient, High-Performance FPGA Implementation, Synplicity, Inc. White paper, January 2005.

[2] Clive “Max” Maxfield, “The Design Warrior’s Guide to FPGAs”, Newnes, 2004.

[3] Uwe Meyer-Baese, “Digital Signal Processing with Field Programmable Gate Arrays (Signals and Communication Technology)”, 2nd edition. Springer- Verlag, 2004.

[4] Keshab K. Parhi, “VLSI Digital Signal Processing Systems” John Wiley & Sons, Inc, 1999.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at **www.microsemi.com**.

© 2012 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

51900252-0/4.12