

Increase Bandwidth in Medical & Industrial Applications With FPGA Co-Processors

Introduction

Programmable logic devices (PLDs) have long been used as primary and co-processors in telecommunications (see *Building Blocks for Rapid Communication System Development* white paper). Digital signal processing (DSP) in medical and industrial applications often has fundamental differences from the typical telecommunication application. In telecommunications, the input data are commonly audio data rates with strict timing constraints such as completing the calculations between successive input data samples. With a DSP processor, this only allows for a few tens of thousands of instructions for the entire calculation. This instruction bandwidth limitation can be minimized by taking advantage of the multiple processing units in some DSP processors. However, creating the specialized pipe-lined code to take true advantage of this parallelism requires hand optimization of assembly language routines. Maintenance, re-usability, and implementation of this type of code can be troublesome and expensive at best. Additionally, the degree of parallelism (simultaneous executions) is relatively low.

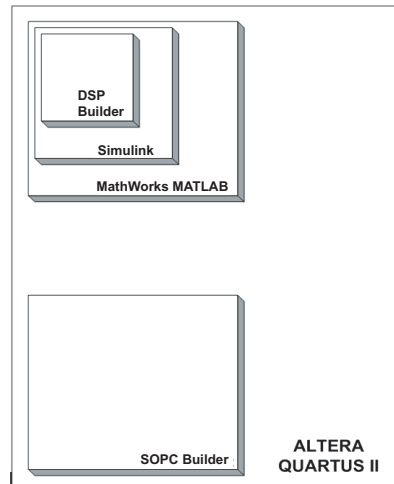
A better alternative for high-bandwidth computations is to use a PLD as a co-processor that integrates the repetitive, speed-critical portions of an algorithm into the PLD. With a PLD and Altera's new automated design software, the engineer has the ability to optimize system performance in ways not possible with a traditional DSP. The purpose of this white paper is to discuss the general issues of moving part, or all, of a DSP application onto a PLD using Altera's DSP Builder and SOPC Builder system software design tools.

Altera's Automated Design Suite

The Altera® design software consists of three main components: Quartus® II, SOPC Builder, and DSP Builder development tools. Their relationship is shown in Figure 1. Collectively, these tools comprise an automated system development platform that provides a high level of design integration and flexibility. Now engineers can concentrate on their target design at the system level rather than at the level of HDL and logic construction.

The Quartus II development tool is the foundation of the design suite. It is the logic-level design tool that supports embedded processor software development, FPGA and CPLD design, synthesis, place-and-route, verification, and device programming. It performs the lower-level functions of producing a programmed PLD from the set of design files passed to it by SOPC Builder and DSP Builder. In a broad analogy to general software development, Quartus II is the assembler and linker.

Figure 1. Altera Software Design Tools



Sitting on top of the Quartus II development tool are two packages that can be used separately or in conjunction, depending on the task at hand. These tools are equivalent to higher-level language compilers where HDL (Verilog or VHDL) is the underlying language. SOPC Builder and DSP Builder are analogous to the visual designer environments like MS Visual C++ providing a large suite of automated design assistance. In fact, with SOPC Builder and DSP Builder, the designer does not need to be a VHDL or Verilog programmer. These packages are automated system generation tools where the components of a hardware system are defined, inter-connected, simulated, and verified, all without resorting to the underlying HDL. With a true point-and-click design method, the system architect can generate an entire system, simulate and verify it, and download it into a PLD all from the PC desktop.

SOPC Builder is an automated system development tool that dramatically simplifies the task of creating high-performance system-on-a-programmable-chip (SOPC) designs allowing the engineer the complete flexibility to pick and choose peripherals and their performance based on the needs of the design. The tool automates the system definition and integration phases of development. Using SOPC Builder, system designers can define a complete system, from hardware to software, within one tool and in a fraction of the time of traditional system-on-a-chip (SOC) designs.

To provide enhanced signal processing capabilities, DSP Builder uses The MathWorks' MATLAB and Simulink tools for signal processing system generation. Creating a DSP application in a PLD requires both high-level algorithm and HDL development tools. The DSP Builder integrates these two functions by combining the algorithm development and the software simulation and verification capabilities of MATLAB/Simulink with the hardware synthesis and simulation of the Altera design software. DSP Builder allows system, algorithm, and hardware designers to share a common development platform that uses a drag-and-drop architecture. Components are selected from a large menu of options and are placed on the Simulink workspace where they are connected with the mouse. Parameters for a given component (for example, an analog-to-digital converter) are controlled by drop-down menus.

Together, these three design tools deliver an exceptional level of design integration and drastically decrease time-to-market for complex designs.

Medical & Industrial Applications

In medical imaging applications, such as computed tomography (CT) and magnetic resonance imaging (MRI), the input data is not a real-time data stream of successive bytes, but rather a large data block (an image) residing in memory or on disk. The task of the image-processing chain is typically to tile the image into sub-blocks, perform a series of linear algebraic transforms on those blocks, and place the resultant data back on the disk. In addition, very large computational bandwidths are required when multiple images are compiled into 3D or coronal/sagittal views.

In many industrial applications, such as ultrasonic flaw detection for example, the data rate from the sensors vastly exceeds audio and may reach as high as 20 MSamp/sec. In other industrial applications, the sensor data rates can be much smaller (e.g., 100 kSamp/sec), but there may be multiple sensors. This drives the effective data rate into the radio-frequency regime. If the processing chain is complex, DSP processors often do not have the bandwidth necessary to meet real-time deadlines and processing must be done in an "offline" manner.

The core of the difficulty with DSP processors in these applications comes in two parts. First, they are essentially serial machines, processing one element of the signal chain at a time. In the case of some high-end DSP processors, a small number of instructions can be processed simultaneously giving a small degree of parallelization. However, the cost of these DSP processors can be ten times that of a non-parallel version.

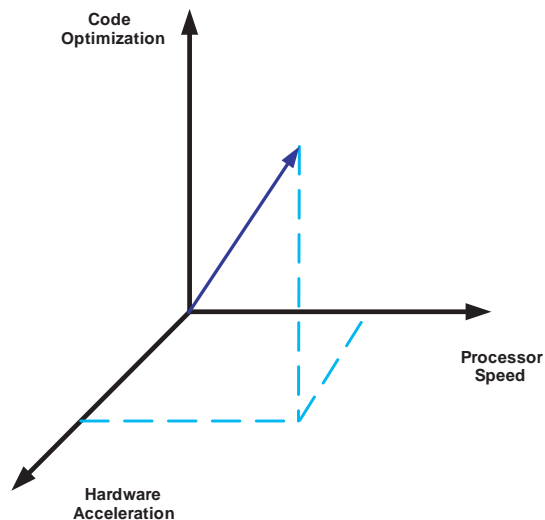
In a number of applications, multiple DSP processors can be utilized to obtain true parallel processing but the costs for hardware rise very quickly as the process is made parallel. Software for these systems becomes ever more complex and expensive as well as requiring a real-time operating system (RTOS), or very careful inter-processor communications schemes. However, the degree of true parallelism is relatively low and comes at a significant price in terms of design time, cost of goods, and time to market.

Gaining Speed With PLDs as Parallel Co-Processors

These issues have led many medical and industrial designers to use PLDs, such as Altera's Stratix® series FPGAs, to take advantage of the ability to convert a cascaded set of operations into a parallel structure that operates in several clock cycles at 200+ MHz. This is the central advantage of the PLD technology—the ability to speed up an algorithm by making the process truly parallel.

In a DSP application, engineers have few options to increase performance beyond writing pipelined assembly language (if their DSP processor supports it), or replacing the DSP processor with a higher frequency model. With the Altera approach, the hardware, as well as the software, can be simultaneously optimized. Now the designer has a three-dimensional optimization space available (see Figure 2), whereas code optimization and processor speed were their only previous choices. With the nearly exponential price versus performance curves for many hardware processors, designers were left with typically only one option—code optimization. As anyone who has had to optimize code knows, you can only go so far before you exhaust that avenue. The broad flexibility provided by Altera's programmable solutions can help create systems that were previously impossible to design either because of time and cost-of-goods, or because traditional DSPs could not handle the calculation load. Now the engineer has one more degree of freedom in the design process—hardware acceleration. This acceleration is accomplished by taking the parallelization process one step further than DSP processors and making the algorithm massively parallel.

Figure 2. The New 3D Optimization Space Opened by Altera's Design Software



In addition to the obvious calculation-speed increase, using PLDs as co-processors gives the designer increased flexibility in four important ways:

- Speed gain in the core algorithm relaxes deadline pressure on the remaining parts of the algorithm, reducing the overall timing criticality of the system.
- High-performance DSP applications can quickly fill the resources of available DSP processors, leaving no room for expansion. With the PLD approach, additional algorithms or filters can be added to the existing system without resorting to extra hardware or increasing the timing pressure on the system.
- Control hardware that traditionally exists external to the DSP processor can be easily integrated into the PLD and combined with the algorithm in a single hardware chip. This can save time and money in PC board development, and has a great impact on the flexibility of the system.
- The effect of specifications, performance, and availability changes made by other manufacturers is reduced because control is in the system architect's hands rather than in a third party's.

To illustrate these issues, Figure 3 shows a C code fragment for a convolution operation such as might be used in image processing or in a signal filter. This nested loop structure accounts for the vast majority of image and signal processing tasks.

With the Harvard architecture of DSP processors, the multiply-accumulate (MAC) can be performed efficiently in a single instruction cycle. However, multiple MACs must be performed to complete the inner loop in Figure 3, and the branching operation of the *for* loops consumes a larger number of instruction cycles as compared to the MAC operations themselves.

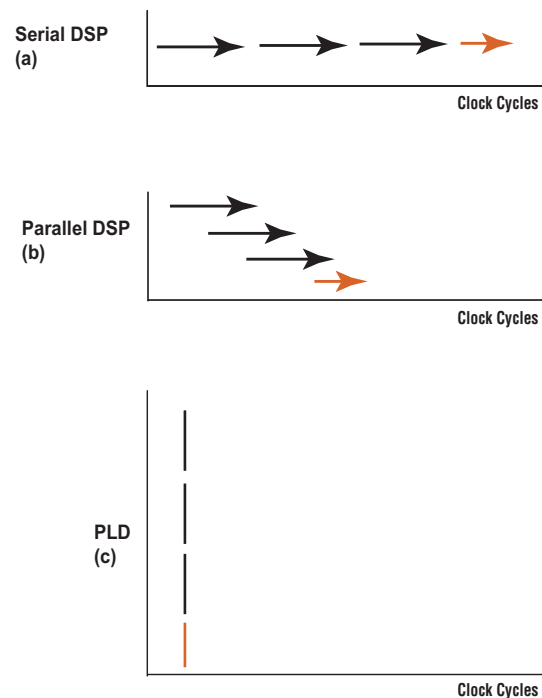
Figure 3. C Code Segment Illustrating Repetitive Nature of Typical Signal Processing Task

```

for (i=0; i < 2*nx; i +=2)
{
    real = 0.0;
    imag = 0.0;
    for (k= 0; k < 2*ny; j += 2)
    {
        real += h[k]*x[i-k] - h[k+1]*x[i+1-k];
        imag += h[k]*x[i+1-k] + h[j+1]*x[i-k];
    }
    res[i] = (real >> 15);
    res[i+1] = (imag >> 15);
}
    
```

For illustration purposes, we will display the clock cycle usage of a traditional DSP processor with this convolution operation and compare it to the same calculation that has been implemented on a Stratix series device using DSP Builder. Imagine un-rolling a single iteration of the outer *for* loop (represented by the colored arrows in Figure 3), and placing them in a serial DSP program. If we plot these calculations versus the clock of the system (calculation time), they are represented by the sequential line of arrows in Figure 4.

Figure 4. Comparison: Execution of a Convolution Algorithm in a DSP Processor & a PLD




In some DSP processors, the software designer can write optimized assembly code to pipeline instructions and data to parallel logical units. This reduces the clock cycle usage as represented in Figure 4(b). Speed gains of up to five times can be obtained by carrying out the optimization, either by hand or in the C compiler. The designer should be aware however that the C compiler can perform some of the parallelization, but the highest speed gains are found in optimized assembly code. Writing this type of code is a specialized skill requiring the detailed analysis of the algorithm to determine the manner in which the instruction and data must be interleaved into the instruction and data pipelines. If timing considerations change, then the code may have to be modified or re-written. A distinct drawback to this scenario is that the coding time and maintenance cost of this type of software is significant.

Using the DSP Builder software and implementing the algorithm on a PLD, the convolution filter can be constructed so that the filter operates in 1-2 clock cycles as shown in Figure 4(c). In many applications, this is the primary consideration—speed. In both systems, we have ignored the initial clock cycles required to load the data (fill the taps) before the calculation can be made.

When designing with a PLD, it is important to understand a key trade-off that requires a change of perspective from traditional DSP design. In order to acquire more speed in a DSP system, the designer can make several modifications:

- Write more optimized code (usually assembly)
- Upgrade the DSP processor to a faster, more expensive model
- Add more DSP processors

In a PLD design, the trade-offs are different. If the designer wants more speed (smaller execution times), then more logic elements (LEs) must be utilized to make the operation more parallel. Since each LE occupies space on the chip, the traditional way to express this trade-off is space vs. speed—more speed requires more space on the chip. Now the primary system trade-off is chip size (number of LEs) versus costs. Altera provides a wide range of chip sizes and silicon technologies that allows the designer to select a performance/price point that works for their system.

 *LEs are the basic units in the FPGA architecture. Altera provides a wide range of chips with different LEs and support features.*

The advantages of the PLD approach are four-fold:

- The speed gains over optimized assembly code in the DSP processor are a factor of 20 or more.
- The Altera design software, such as DSP Builder, is a graphical design tool that uses a drag-and-drop architecture. Thus, development time is significantly less than the writing of pipelined DSP code.
- If the filter requires a hardware control interface to external hardware, it can be easily implemented directly on the same PLD with the Altera software rather than writing code to interact with the interrupts or buses on a DSP processor.
- Modifications to the design and architecture are fast and easy to implement.

Design Decisions: PLD Co-Processor or DSP Processor

The speed advantage of using PLDs is evident from Figure 4. The process can be made parallel so that the ratio of computational throughput to the number of clock cycles used is quite high. In making the decision to take a project in the direction of PLD co-processors, there are a number of factors the designer must consider. We will attempt to illuminate the major concerns here.

The first design job is to segment the design into those tasks that will be placed in the co-processor, and those left in a DSP processor or other system microprocessor (the master processor). As we shall see in the design example, even the master processor can sometimes be eliminated from the hardware design with the use of the Nios[®] II soft processor.

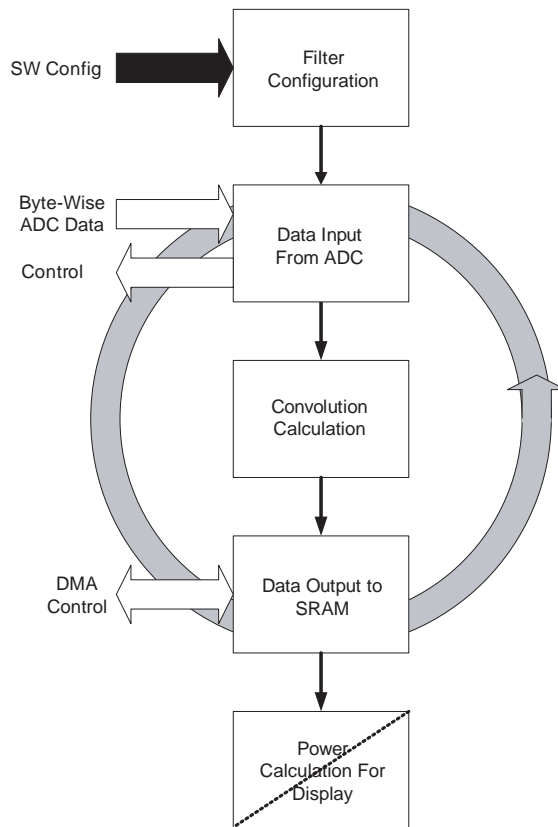
When faced with the segmentation task, the easiest and best way to look at the segmenting process is to attempt to divide the problem into two independent but related components: 1) the computational algorithm itself, and

2) hardware control of that algorithm. While these are interdependent, they can be separated easily with the use of a simple flow chart.

Figure 5 shows a simplified example that illustrates the way to look at the segmentation process. In the flow chart, we see the implementation of a simple FIR that filters a high data rate, real-time signal (for example, bandpass filtering a noisy sensor signal), and displays a derived parameter (in this case the signal power) on a CRT or LCD.

The upper block, labeled Filter Configuration, is where the tap coefficients are calculated based on the configuration of the system. For example, the user may choose filter type (Butterworth, Chebyshev, and so on), and cut-off frequencies. The crosshatched arrow leading into the Filter Configuration block is the software control provided by the master processor.

Figure 5. Segmentation of a Real-Time FIR Filter Processor



In the second block, the system manages the data input, such as framing incoming data into manageable segments for the filter. Additionally, this data framing operation must also send a few hardware signals to the ADC such as *enable* and *acknowledge*. The white arrows in the figure represent these hardware control points. In the Data Output block, hardware control is again required to format the output data and facilitate its final placement, such as a DMA operation formatted and placed in external SRAM. In the final block, the power in the filtered signal is calculated by squaring the signal elements and summing. The power parameter is then displayed on a CRT or LCD.

Now let's segment the design into parts that will reside in the PLD and parts that remain with the master processor. The grey circular arrow in the background of Figure 5 represents the high-speed, repetitive portion of the algorithm. This is the segment of the algorithm that changes little in form throughout the system's usage. For example, it does not become an infinite impulse response (IIR) filter based on simple selections of the user. It is also the most

speed-critical portion of the algorithm. This is the best candidate for transfer to a PLD based on speed requirements and its static configuration.

The final block has a dashed line diagonally across its face. This represents a possible division of the task into separate parts. Since we are simply displaying the power parameter, the timing pressure on this part of the algorithm is very low, on the order of a few tens of milliseconds. However, the calculation of the power can be computationally intense because it involves the squaring of an array element by element and then summing the elements together. Whether this remains with the master processor or is placed in a PLD depends on a number of questions:

- What is the load on the master processor? Can moving task(s) to the PLD free other master processor resources?
- How much FPGA real estate (LEs) does the filter operation take? How many LEs will the power calculation take?
- Are you using floating or integer arithmetic in the master processor? If integer math is used, then a scaling operation to prevent possible overflow will be needed because of the squaring operation.

When segmenting the design, designers will typically find that there are obvious portions of the algorithm that should be moved to the PLD while others are dependent on a number of system issues. For more complex code, the best way to make this determination is to develop a higher-level language (such as C/C++ or MATLAB) model of the code. The compiler profile functions can be utilized to determine execution time, and to find which portions of the algorithm are using the most CPU resources. Since MATLAB is required for the DSP Builder, it is very convenient to use MATLAB for the algorithm simulation and employ its tools such as the graphical profiler.

For those unfamiliar with MATLAB, it is an interpretive language with many common features of C/C++ and can be learned in a short period of time. See The MathWorks website at www.mathworks.com for more information on MATLAB and Simulink.

Design Decisions: Computational Control—The Nios Soft Processor

Often after the design has been


segmented and the repetitive elements have been configured in an Altera PLD, there still exists a control function that must be implemented as we saw in the previous section. In that case, we left the filter coefficients and the final display algorithm in the master processor. A general-purpose processor is better suited to these types of calculations for the following reasons:

- They are usually based on input from a user interface (UI) that is typically slow and subject to change.
- Multiple decision trees must be implemented.

Altera offers an alternative to keeping a DSP processor in the system for control purposes with its Nios II embedded processor. The Nios II processor is a 32-bit microprocessor that can be custom designed with peripherals and control logic using the SOPC Builder software. The processor design is then converted into interconnected LEs by Quartus II software, and can be downloaded into the FPGA much like any other hardware design. Sometimes this concept takes a moment to take root because it is a radical departure from the norm; not only is the logic design downloadable to the PLD, but the entire microprocessor is also downloadable.

The Nios II processor also provides a set of unique features specifically designed for performance enhancement—custom instructions and custom peripherals. With these new software functions, designers can combine a complex sequence of hardware operations into a single Nios II function call. This hardware acceleration technology is a unique way to integrate system hardware directly into the software. Essentially, it adds custom-defined logic/hardware functionality to the processor. Custom instructions are simple operations that can be handled in the Nios II processor registers. Subsequently, a custom instruction is attached directly to the arithmetic logic unit (ALU) of the Nios II processor. A simple example of a custom DSP instruction might be a MAC that also scales the result based on its size. Rather than writing a segment of code that operates with all the CPU overhead to perform the MAC and the scaling, the operation can be implemented directly in hardware and called in software as though it is a member of the native instruction set. Custom peripherals are larger and more complex pieces of hardware that can be abstracted into a

single C-callable function. By using these hardware acceleration technologies, the timing control of any design is simplified enormously.

 For more information on custom instructions, see the *Nios II Custom Instruction User Guide*

The combination of a Nios II processor with the DSP Builder can yield great gains in terms of performance as well as cost-of-goods. A single PLD can replace several DSP processors plus their support hardware depending on the calculation and control tasks in the system.

Design Example: A Cross-Correlation Image Processor

In this section, we discuss a medical image processing application. Because the design is complex, we will supply only the highlights of the design. A traditional image processor might consist of one or more DSP processors running a set of pipelined assembly and C code. The DSP processor(s) would be responsible for the algorithm calculation as well as the overall system control functions—an area where DSP processors typically perform poorly.

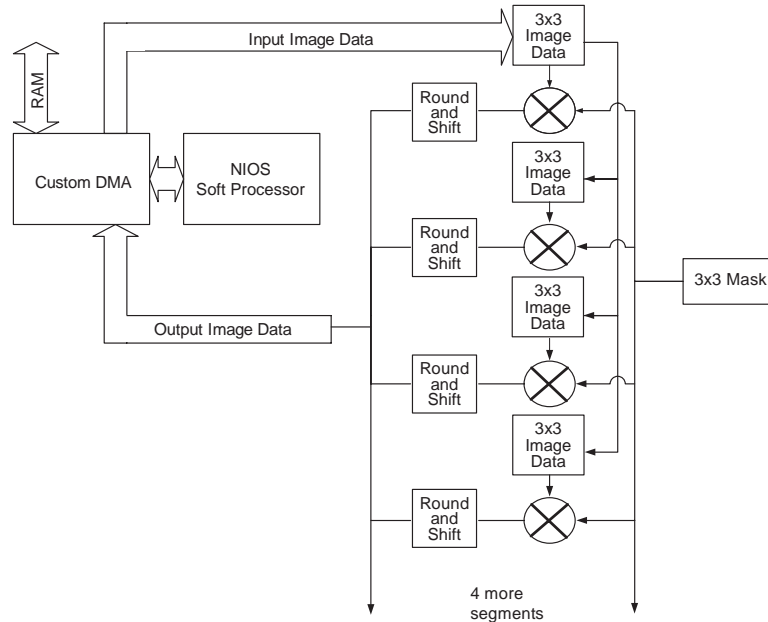
Registration of medical images can be aided by the use of fiducial (reference) markers purposefully placed in the image. One technique that can be used to automatically locate these marks is to run a cross-correlation between the image of the fiducial mark and the image under study. The core algorithm for this processing is a small mask cross-correlation, which is processed at each point in the image.

We will design and implement a simple 3×3 cross-correlation image processor. The algorithm multiplies the nine pixel values in the 3×3 array by the corresponding nine values stored in a 3×3 reference mask and then sums the nine products. As a final step, the algorithm rounds-off and then bit-shifts the bytes into the original data size. This circuit will be referred to as a multiplier segment.

The image that we will be processing is the 512×512 pixels typical of CT images. The 3×3 cross-correlation will be evaluated at each pixel in the original image. We will evaluate the Texas Instruments 3×3 correlator that uses 8-bit data, but we will devise our own 16-bit correlator that is more suited to the dynamic range of medical images.

In order to increase parallelism in our PLD design, we will process eight rows of the image simultaneously by stepping through the rows from left to right. The value of eight was a trade-off between speed and design complexity for this example. The design is extensible to as many parallel-row calculations that the chosen PLD can support. Altera has PLDs with up to 179,760 LEs. For simplicity in Figure 6, we show only one-half (four rows) of the correlation processor that we will utilize as a part of a larger system.

Operation of the system is straightforward (see Figure 6). The DMA is constructed as eight parallel sections each serving one of the multiplier segments. The first step is for the Nios II processor to load the parallel DMA processor with starting addresses for each of the segments. Each segment of the DMA extracts the image data for the 3×3 cross-correlation and loads into its respective segment a single byte per clock cycle. At each load cycle, the circuit calculates an output, but the only valid result is the one in which all nine values have been loaded. Therefore, DMA only returns values to memory that are the result of a completed calculation.

Figure 6. Four Segments of an Eight-Segment 3×3 Correlator

The design consists of the following four major components:

- The eight 3×3 multiplier segments designed in DSP Builder
- A Nios II processor designed in SOPC Builder
- A parallel DMA module that operates at high speed designed in DSP Builder
- The C code for the Nios II processor

An important point to note as we walk through this example is that the Altera software allows the design process to be object-oriented in much the same manner as C++. Once a design element is created and verified, we will encapsulate it and treat it as a black box (in analogy to a class in C++). It will appear as a block in the Quartus II integration, and at that stage, we will no longer concern ourselves about the lower-level details of the block. The advantages of this approach are the same as with C++: re-usability and standardized interfaces.

Multiplier Segments

Figure 7 shows the Simulink block diagram from DSP Builder used to create the a single multiplier/round-off segment depicted in Figure 6. The appeal of DSP Builder is that standard signal processing blocks can be used to assemble a high-speed signal processing system in a PLD without resorting to HDL programming. In the construction of the multiplier sections, the following four standard blocks from the DSP Builder Library were used:

- Two 9-tap FIR filter blocks
- Two multiply-add blocks
- One parallel adder block
- One product block

To understand the heart of the algorithm, we will step through the operation of a single multiplier/round-off section. First, the *data_reg_select* line is held low and the nine 16-bit bytes of the 3×3 reference mask are loaded into the lower FIR filter. The system is now ready to work its way through the image. Next, at the edge of each clock cycle (in this design $f = 300$ MHz), the DMA presents a 16-bit byte from the image on the *Data_In* port (far left-hand side of Figure 8), which is loaded into the lower byte of the upper FIR register. The calculation of the 9×9 product and sum is performed in a single clock cycle. We note that the result is presently invalid because we have not finished loading

the whole set of nine values into the upper FIR. However, the system does nothing with the result. At each clock cycle, another of the nine image bytes is loaded and the results are calculated. The DMA circuitry only transfers the cross-correlated byte to memory after the entire calculation is completed. All other results are ignored.

On the edge of the ninth clock cycle, we have loaded all the registers and the calculation has been concluded. Since the round-off operation must be presented with valid data from the cross-correlation sums, the system requires another clock cycle to perform the round-off and bit-shift. The results appearing at the *Conv_data* port (far right-hand side of Figure 7) is taken by the DMA circuitry and placed back in the resultant image memory in a single clock cycle. Our total time to calculate a single 3x3 cross-correlation is 11 clock cycles. In the full design, eight of these sections are aligned in parallel, so eight cross-correlations are performed in 11 clock cycles.

Figure 8 shows the Quartus II black boxes of the correlator and the round-off circuits. The circuit in Figure 7 is encapsulated into a single black box. This correlator can now be shared among designers and projects.

Figure 7. The DSP Builder Block Diagram of a Single Cross-Correlator Segment

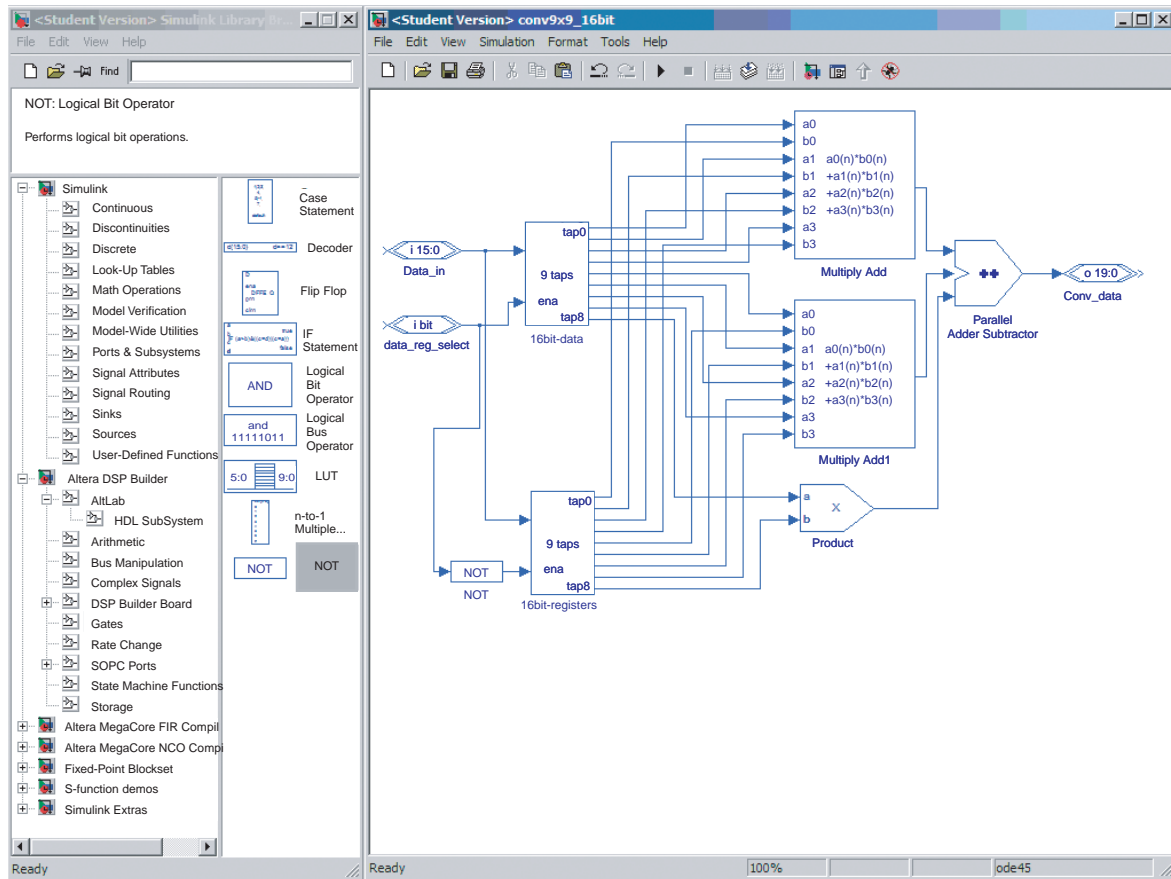
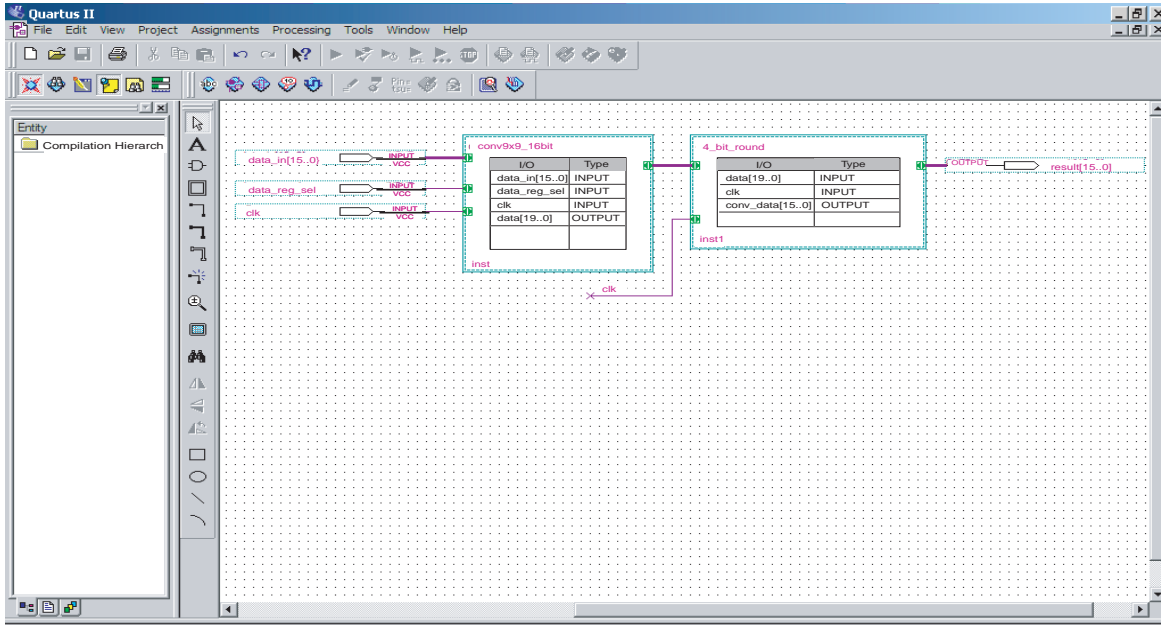



Figure 8. The Quartus II Block Diagram of a Single Cross-Correlator Segment



Nios II Processor Design

The Nios II processor serves as the on-chip controller by setting up the DMA transfers, loading static values such as the mask, and communicating to the host system.

 *SOPC Builder provides a number of standard communication protocols including serial, Ethernet, and PCI bridges. Third-party vendors can supply other protocols.*

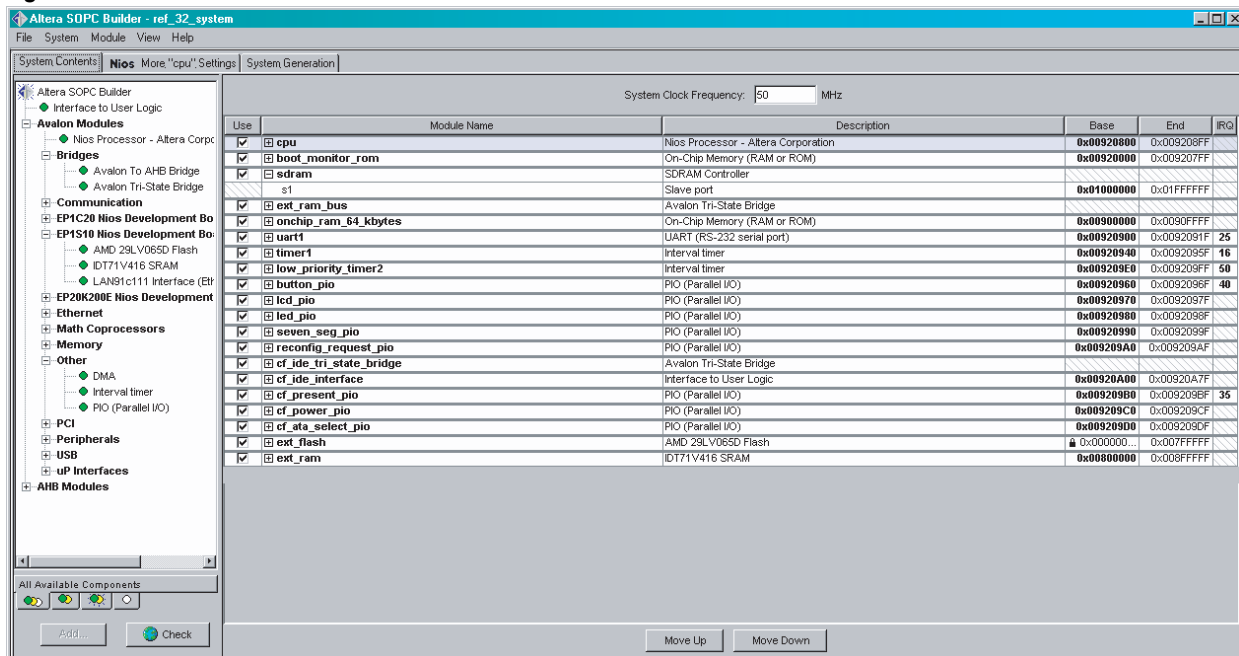
Construction of a complete Nios II processor consists of selecting the major processor components needed in the application. Within SOPC Builder, the engineer chooses the required components such as boot ROM, external memory, and peripherals. Figure 9 shows the Nios II design components for the correlator in the list format of SOPC Builder. Components are selected from the left-hand panel, and are automatically added to the table in the center. Parameters such as memory locations, interrupts, and operational modes are set with the Configuration Wizard of each component.

Upon generation and compilation of the Nios II processor, a custom software development kit (SDK) is automatically generated for use with the hardware. The SDK contains explicit instructions for the C compiler/linker such as variable definitions and memory maps to actual devices. The software for the correlator is then written within the context of this SDK and the low-level software details are managed by the SDK itself. In other words, connectivity between the hardware and Nios II processor software is automatically handled by the SDK.

In order to streamline the Nios II processor code, we will create two custom peripherals by grouping the hardware for the DMA and the hardware for the multiplier cells into separate custom peripherals. These complex circuits are then reduced to single function calls within the Nios II processor software.

After the complete circuit has been simulated, we accomplish this abstraction by collecting the two hardware sections into separate files. By following a few simple design rules, these custom logic blocks are imported into the Nios II design by filling out some simple menu items in the SOPC Builder software. We then instantiate these blocks as black box items, meaning that they will now appear as user blocks in the Quartus II final system integration. After this integration, these circuits are accessed and controlled in our Nios II C code as ordinary function calls.

Figure 9. The Nios II Soft Processor in SOPC Builder



DMA

As we saw in the multiplier/round-off discussion, the most time-consuming portion of the algorithm is in data movement. In order to get high throughput in a cross-correlator of this type, it is essential to have a DMA processor that moves the data from external memory to the inputs of the multiplier segments as quickly as possible. To accomplish this, the DMA processor loads the byte of the nine image values simultaneously on all eight segments.

The DMA processor is essentially a memory address/fetch engine that generates the address of the desired memory locations, and manipulates the required control lines to force the memory to present its data. In addition, it must do some simple shifting of addresses in order to acquire the data in a 3×3 block within the image and then index through the pixels of the original image on the clock edge. It also places the resultant cross-correlation into memory in image format—usually just rows placed end-to-end.

The actual structure of the DMA processor is system-architecture dependent, and depends explicitly on a number of decisions the designer may be forced to make for reasons other than speed. For example, the designer may choose SDRAM over SRAM because of costs. The precise construction will be sensitive to the following issues:

- Memory type (for example, SRAM or SDRAM)
- Memory access speed
- Bus speed, architecture, and arbitration

Software Design

Utilizing the Nios II processor custom peripheral capability, program and hardware control can be greatly simplified. The hardware shown in Figure 6 will be encapsulated into the three separate function calls found in Table 1.

Table 1. Nios Custom Peripherals for the 3x3 Cross-Correlator

Custom Peripheral Command	Functionality
nm_image_input()	DMA image data from original image
nm_correlate()	8-row cross-correlator
nm_image_output()	DMA results to memory

Once the proper interface discipline has been followed in the setup and definition of our custom peripherals, their use becomes that of a simple function call. The program that runs the cross-correlator is now abstracted into a few simple lines of C code. Since we will calculate eight rows at a time, we will run the cross-correlation processor 64 times to complete our 512×512 image.

Performance & Cost Comparison Between a PLD & a Traditional DSP

Finally, we compare the performance of our eight segment cross-correlator with that of a 200-Mhz Texas Instruments TMS320C6201 processor running the same algorithm from Texas Instruments' IMAGELIB Image Processing Library. Comparison of the two systems is found in Table 2.

Table 2. Stratix EP1S10 vs. TMS320C6201: Clock Cycles per Pixel(1)

	Calculation (Clock Cycles/Pixel)	Data Transfer (Clock Cycles/Pixel)	Total (Clock Cycles/Pixel)
TMS320C6201	5.26	3.41	8.67
Stratix EP1S10	0.25	1.13	1.38
Speed Increase	21.0	3.0	6.3

Note:

(1) The clock cycle values for the TMS are compiler dependent. We report the lowest values obtained in the Code Composer Studio profiler.

The speed increase in the calculation of the results in a PLD is a factor of 21 times greater than the TMS320C6201 DSP processor. The speed gain in the data transfer is much less and will depend upon the memory and bus configuration. There is some degree of uncertainty in the comparison of data transfer rates because the transfers are performed on two different systems. The message here is that the more computationally intense an algorithm, the more speed gains are possible with the Altera PLD approach. If data transfer dominates the algorithm, then the speed gain will be less.

Ease RoHS Transition With Altera Lead-Free Products

Altera maintains one of the most extensive lead-free product offerings in the industry, with over 1200 products in lead-free packages. As a preeminent supplier of environmentally friendly programmable logic solutions, Altera has shipped over 25 million lead-free products since 2002. Altera's lead-free devices comply with the maximum concentration restrictions, as required in the EU Directive on the Restriction of Hazardous Substances ("RoHS Directive") No.2002/95 with respect to lead (Pb), mercury, cadmium, hexavalent chromium, polybrominated biphenyls (PBB), and polybrominated diphenyl ethers (PBDE). Help ease your RoHS transition by integrating non-compliant ASSPs with Altera's PLDs.

Conclusion

Like every design decision in a complex system, moving all or part of an algorithm into a PLD co-processor depends on a wide range of issues. It really comes down to the analysis of the size and complexity of both the hardware and software components in the design. For example, a simple algorithm that presently runs at adequate speed in a low-cost DSP processor is probably not a good candidate for porting to a PLD unless the goal is to reduce PC board real estate or to have a higher degree of hardware integration. However, if your system is high performance, and generally pushes a DSP processor to its limits, then a PLD co-processor approach may be your only viable alternative.

Altera's design tools (Quartus II, DSP Builder, and SOPC Builder) are major steps forward in system development. The wide range of automated tools make the development of PLD systems faster and easier than ever before. We have shown here that the use of PLDs as co-processors in medical and industrial applications provides considerable speed gains over high-end DSPs. In addition, the system architect is given a much higher degree of control than is possible with DSPs.

Resources

For additional information, refer to the following resources on the Altera website.

- More System Integration Solutions
www.altera.com/technology/integration/int-index.html
- Altera DSP Solutions
www.altera.com/technology/dsp/dsp-index.jsp
- Information on Nios Embedded Processors
www.altera.com/products/ip/processors/nios2/ni2-index.html
- Customer Applications of Programmable Logic
www.altera.com/corporate/cust_successes/customer_showcase/csh-index.html
- Information on Altera Programmable Logic Devices
www.altera.com/products/devices/dev-index.jsp

Special Acknowledgement

Altera acknowledges Dan Grolemond, Ph.D., and Jose Gorostegui of Glacier Point Research, Inc., for their contributions to this project.



101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
<http://www.altera.com>

Copyright © 2006 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.