
Implementing Bit-Serial Digital Filters in AT6000 FPGAs

Introduction

This application note describes the implementation of digital filters in the Atmel AT6000-series FPGAs. Bit-serial digital signal processing is used to construct efficient Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filter macros that can be cascaded to create higher-order functions. We will briefly review the techniques first introduced in the application note: FPGA-based FIR Filter using Bit-Serial Digital Signal Processing. Several reference designs are included that demonstrate specific filters and how they are constructed. These filters are available as hard macros that can be used in DSP designs. The implementation details of these filters along with performance, sample-rates, numeric precision, and device utilization are presented. The digital filter macros are variable coefficient filters, where the characteristics of the filter can be modified in real-time via partial dynamic reconfiguration of the FPGA.

FIR and IIR filters are used in many digital signal processing (DSP) systems to perform a variety of signal filtering and conditioning functions. The IIR filter is capable of emulating the transfer functions of analog continuous-time filters, such as low-pass, band-pass, high-pass, and all-pass (phase-shifting) types of filtering. IIR filters exhibit similar phase characteristics as their analog counterparts. For arbitrary transfer functions with linear-phase response, FIR filters are utilized and have no equivalent in the analog domain.

Applications such as telecommunications, instrumentation, digital-audio, multimedia, video and image processing can

be supported with AT6000 FPGAs. The architecture of the Atmel AT6000-series FPGAs has been designed to support compute-intensive types of designs. For example, AT6000 FPGAs can be used to construct adaptive filters that a DSP or microprocessor can control through partial reconfiguration. This kind of scheme exploits the natural synergy that exists between software programmable microprocessors and the reconfigurable hardware of FPGAs. Algorithms for adaptive control are usually complex but operate at much lower data-rates than the signal processing channels they control. Hence, the adaptive control can reside in the DSP processor and the actual signal processing can occur in the FPGA.

Fixed versus Variable Coefficient Design Issues

Different FPGA architectures lend themselves to certain types of compute-oriented functions. Therefore, each FPGA type has its respective strengths and weaknesses within a given corresponding application. Basically, there are two kinds of SRAM-based FPGA architectures in use for DSP functions, logic-based core cells and LUT-based core cells.

Logic-based cells are simple and straightforward. The number of data bits required to configure the cell is small, yielding faster reconfiguration times. Furthermore, a one-to-one correspondence between the logic design and the physical realization exists. This leads to direct implementation of basic mathematical functions in hardware, such as adders and multipliers. Arithmetic arguments for DSP operations (e.g., filter coefficients) are presented to the computational units as numeric constant



Digital Filters in AT6000 FPGAs

Application Note



cells or storage registers. Direct manipulation of these values through partial dynamic reconfiguration of the FPGA is as easy as writing to a register. DSP functions targeted for this type of FPGA architecture, such as filters and convolvers, can employ variable coefficient schemes that are device efficient and powerful - they can be modified in real-time. The downside is, in order to achieve the parallelism that FPGAs can provide to DSP functions, discrete multipliers must be built and they are typically the most expensive compute resources in terms of device area.

Look-up table (LUT)-based FPGA architectures are generally more flexible from a breadth of application point-of-view. They permit any boolean function of their inputs to be realized and also allow for classic look-up table techniques to be exploited. A vector multiplier consists of an array of LUTs in which the inputs are the data values to be operated upon and the outputs are partial products, which are then summed using either an accumulator or a tree of adders. This technique permits the equivalent of many simultaneous multiplications in a single clock cycle.

However, the downside of this scheme is that the contents of the LUTs contain pre-computed values based upon coefficient values chosen at design compilation time. Hence, the coefficients are fixed. High sample-rates can be achieved at the expense of real-time modification.

Obviously, each architecture has value. In many situations, fixed coefficients are suitable and there is no compromise. For adaptive applications, LUT-based schemes can be restrictive or purely unacceptable. For these applications, logic-based FPGA architectures have decided advantages. In this application note, we will focus on digital filters realized in AT6000-series FPGAs wherein all variables are capable of modification in real-time.

Digital Filter Architectures

Definition

A digital filter is simply a discrete-time, discrete-amplitude convolver. Basic Fourier transform theory states that the linear convolution of two sequences in the time domain is the same as multiplication of two corresponding spectral sequences in the frequency domain. Filtering is in essence the multiplication of the signal spectrum by the frequency domain impulse response of the filter. For an ideal lowpass filter the pass band part of the signal spectrum is multiplied by one and the stopband part of the signal by zero.

Digital Filter Types

The general form of the digital filter difference equation is:

$$y(n) = \sum_{i=0}^N a(i)x(n-i) - \sum_{i=1}^N b(i)y(n-i)$$

where $y(n)$ is the current filter output, the $y(n-i)$'s are previous filter outputs, the $x(n-i)$'s are current or previous filter inputs, the $a(i)$'s are the filter's feed forward coefficients corresponding to the zeros of the filter, the $b(i)$'s are the filter's feedback coefficients corresponding to the poles of the filter, and N is the filter's order. IIR filters have one or more non-zero feedback coefficients. That is, as a result of the feedback term, if the filter has one or more poles, once the filter has been excited with an impulse there is always an output. FIR filters have no feedback coefficients. That is, the filter has only zeros, and once it has been excited with an impulse the output is present for only a finite (N) number of computational cycles.

Strengths and Weaknesses

Because an IIR filter uses both a feedforward polynomial (zeros as the roots) and a feedback polynomial (poles as the roots), it has a much sharper transition characteristic for a given filter order. Like analog filters with poles, an IIR filter usually has nonlinear phase characteristics. Also, the feedback loop makes IIR filters difficult to use in adaptive filter applications.

Due to its all zero structure, the FIR filter has a linear phase response when the filter's coefficients are symmetric, as is the case in most standard filtering applications. The noise characteristics of an FIR implementation are easy to model especially if no intermediate truncation is used. In this common implementation, the noise floor is at $(-6.02 \cdot B + 6.02 \cdot \log_2 N)$ dB where B is the number of actual bits used in the filter's coefficient quantization and N is again the filter order. This is why many digital filter designs call for more coefficient bits than data bits.

An IIR filter's poles may be close to or outside the unit circle in the Z plane. This means an IIR filter may have stability problems, especially after quantization is applied. A FIR filter is always stable.

Standard FIR Filter Design

Figure 1 shows a flow diagram of a standard 8-tap FIR digital filter. The filter has eight data registers, the FIR is often termed a transversal filter since the input data transverses through the data registers in shift register fashion. The output of each register (D1-to-D8) is called a tap and is termed $x(n)$, where n is the tap number. Each tap is multiplied by a

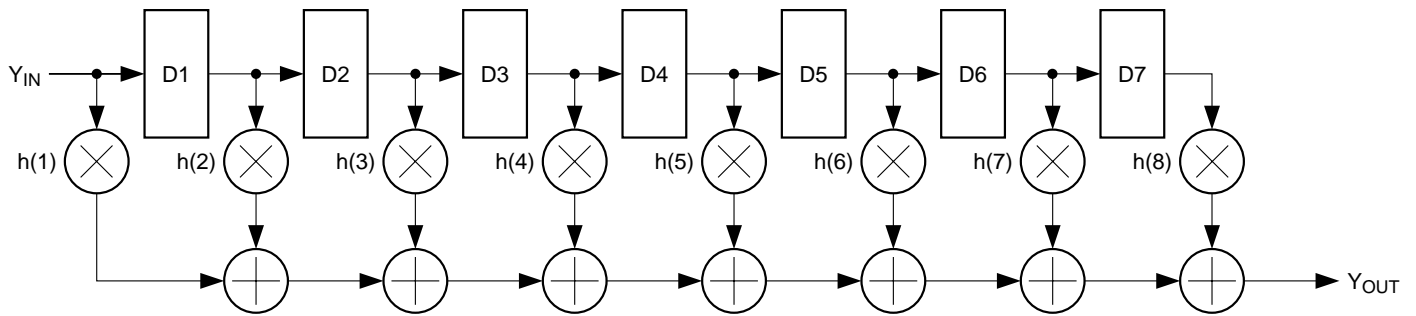


Figure 1. Standard 8-Tap

coefficient $h(n)$ and the resulting products are summed. The equation for the standard FIR filter is:

$$y(n) = \sum_{n=1}^8 x(n)h(n)$$

Although this form is the general one and has many uses, the symmetric FIR filter is often used because of inherent reduction in the number of multiplications required for processing the output.

Figure 2 shows a flow diagram for a symmetrical 8-tap FIR digital filter. This filter provides a linear phase response and the coefficients are symmetrical across the range of taps. This symmetry permits the symmetrical taps outputs to be

added together before they are multiplied by the coefficients. As seen in the diagram, the number of multiplies is reduced from eight to four although four additional add-operations are required. In the bit-serial domain, this is a easy compromise to make due to the small size of the bit-serial adders (essentially, there are n bit-serial adders for an n -bit serial-parallel multiplier — see section on the bit-serial multiplier). The equation for the 8-tap symmetrical FIR is:

$$y(n) = [\{x(1) + x(8)\}h(1)] + [\{x(2) + x(7)\}h(2)] + [\{x(3) + x(6)\}h(3)] + [\{x(4) + x(5)\}h(4)]$$

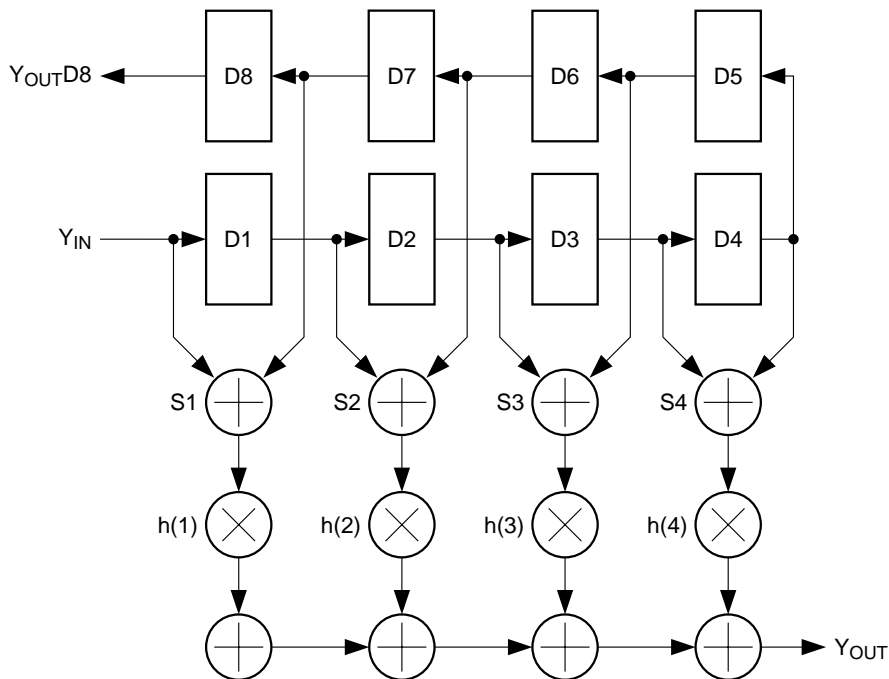


Figure 2. Symmetrical 8-Tap FIR Filter

Figure 3 shows a flow diagram for a second-order IIR digital filter. This filter is the standard canonical form of the IIR filter and is described mathematically as:

$$y(n) = \sum_{i=0}^N a(i)x(n-i) - \sum_{i=1}^N b(i)y(n-i)$$

The IIR filter can be cascaded to achieve higher-order filter functions as in analog filters. IIR filter can be paralleled or cascaded and most filter design software tools support the development of cascaded filters.

Compute-Intensive Design Strategies

We will now focus on digital filter design blocks using bit-serial arithmetic digital signal processing. Bit-parallel implementations are the subject of a subsequent application note. Bit-serial is a digital processing technique that processes data one bit-at-a-time. Although processing functions such as multiplication and addition take many clock cycles to complete, the size of the logic resource needed to perform these operations is reduced by at least 1/n over the equivalent bit-parallel logic resource. Furthermore, since routing and logic levels within the function (adder/multiplier, etc.) are also reduced accordingly, much higher clock speeds are possible with a bit-serial approach. Consequently, for certain classes of applications, FPGA utilization is high, performance goals are attained while using economically attractive FPGA devices. For applications requiring high-speed performance, bit-parallel yields the highest performance figures.

Bit-Serial Merits

Parallel arithmetic structures, such as single-cycle multipliers and wide adders, are often inefficient in smaller (cost effective) FPGAs. In cases where the element fits, the routing resources are overtaxed or the resulting design oper-

ates too slowly to meet the application's requirements. Parallel structures process all the bits simultaneously at a significant hardware cost. Bit-serial, by comparison, process the input one bit at a time. The advantage is that all the bits pass through the same logic, resulting in a huge reduction in the required hardware. Typically, the bit-serial approach requires 1/nth of the hardware required for the equivalent n-bit parallel design. The price of this logic reduction is that the serial hardware takes n clock-cycles to execute, while the equivalent parallel structure executes in one clock-cycle.

However, the speed/logic-cost product is usually better than the equivalent parallel designs because the logic delays between registers are generally significantly smaller. This means that the serial machine can operate at much higher clock frequencies. In FPGAs, signal routing contributes sizable propagation delays and consumes resources. Serial structures tend to have very local routing, often to one destination. In contrast, parallel machines require many signals to span the width of the processing element. In some cases, the overall throughput for a serial design implemented in a FPGA can actually exceed that of an equivalent parallel design in the same device.

The bit-serial DSP elements can operate at clock speeds in excess of 100MHz in AT6000 FPGAs. For 8-bit operations, (8-bit input-data and 8-bit coefficient word-lengths) Bit-serial functions such as accumulators and multipliers can achieve a raw processing rate of 5M samples per second. For 16-bit operations, (16-bit input-data and 16-bit coefficient word-lengths) the raw rate is 2.5M samples per second. By utilizing "function folding techniques", the elements can be used to perform many DSP operations per sample period for applications, such as, telephony, multimedia audio, industrial signal conditioning and analysis. In this class of applications, signal sample rates range from 4-8K samples/sec. to over 50K samples/sec.

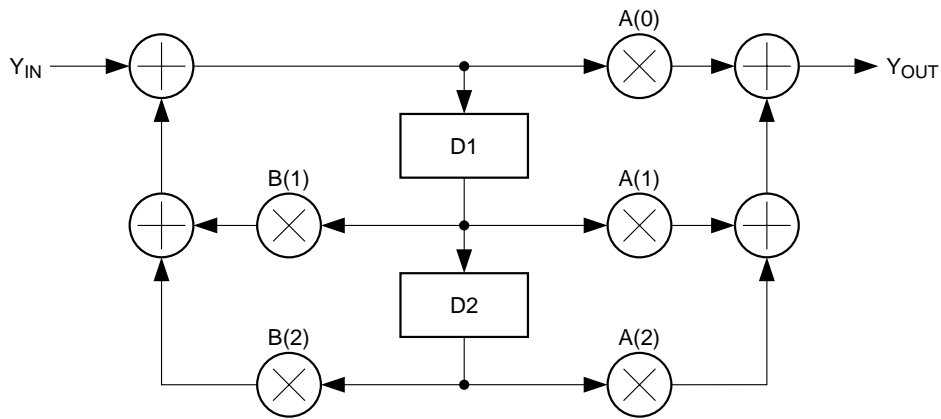


Figure 3. Second-Order IIR Filter

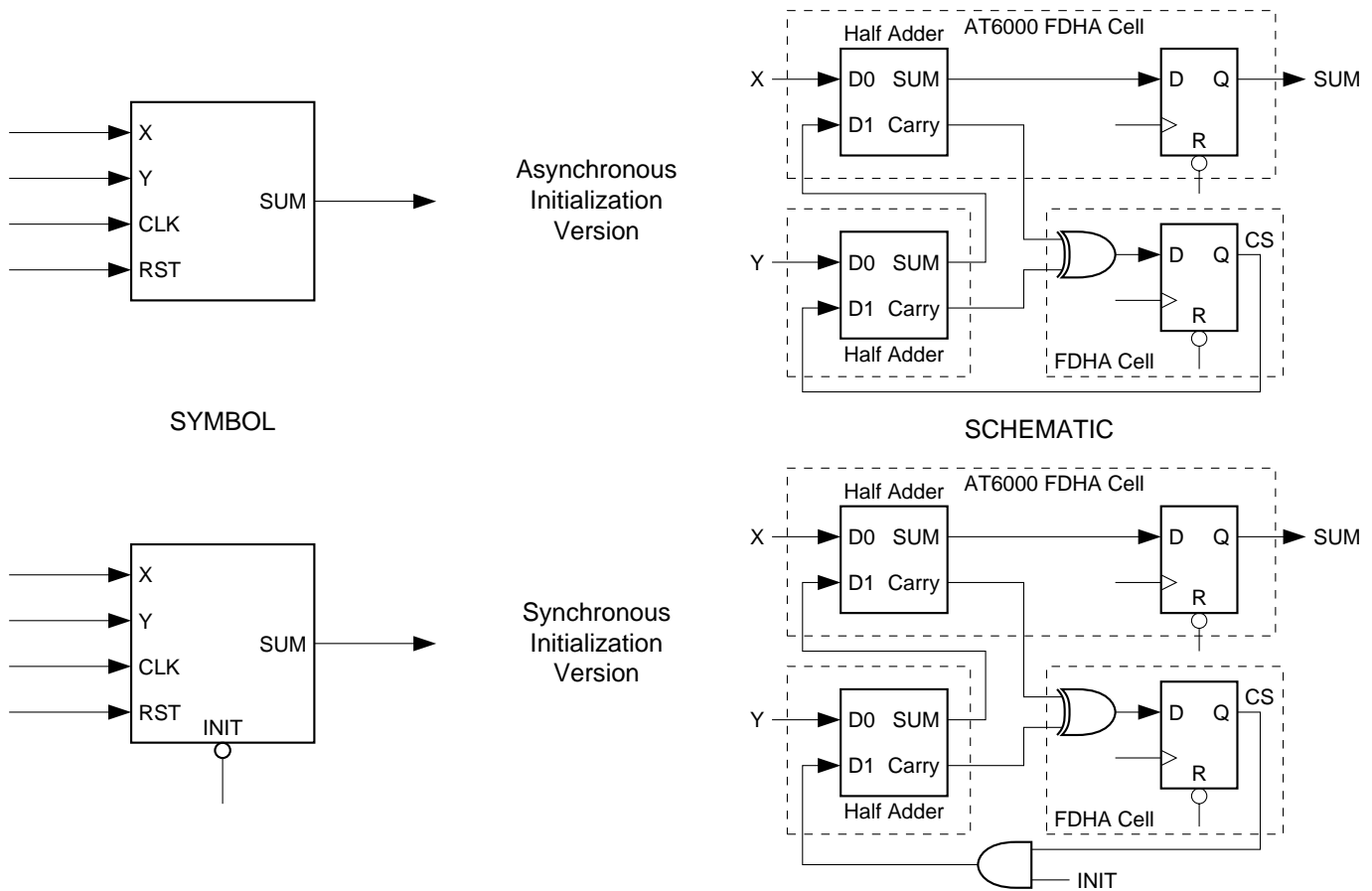


Figure 4. Bit-Serial Adder

Bit-Parallel Merits

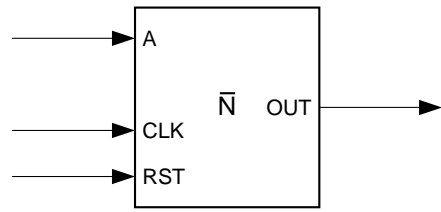
Bit-parallel DSP elements can operate at speeds in excess of 50MHz. For highly pipelined adders and comparators, that rate can be much higher. By using multiple arithmetic elements operating in bit-parallel mode, significant acceleration of algorithms is possible. Since parallel forms of arithmetic elements use many logic blocks and routing resources, their applications can consume much of the available logic inside a given FPGA. Consequently, external resources, such as high speed memory, are used to pump data into and out-of the FPGA. Multiple FPGAs are used to construct higher-level system functions. Final product cost as well as ergonomic and power considerations generally pose the upper limit to these kinds of multi-chip systems. However, the continuing improvement of the price and performance of FPGAs is attracting designers wishing to push the limits of performance while maintaining a highly adaptive and flexible hardware environment. In general, construction of bit-parallel elements is automated through the use of the Atmel Component Generator. With this tool, a large number of compute-intensive functions can be created in minutes. These building blocks enable rapid development of bit-parallel DSP-oriented designs.

Basic Bit-Serial Building Blocks

The basic functions required for most any signal processor include addition, negation, and storage registers. These basic functions can then be used to construct the more complex structures such as accumulators and multipliers. In most cases, using a bit-serial architecture simplifies the hardware required since all of the data bits pass through a single bit-wide element. For bit-serial arithmetic, the two fundamental building blocks are the bit-serial adder and the two's complement circuit.

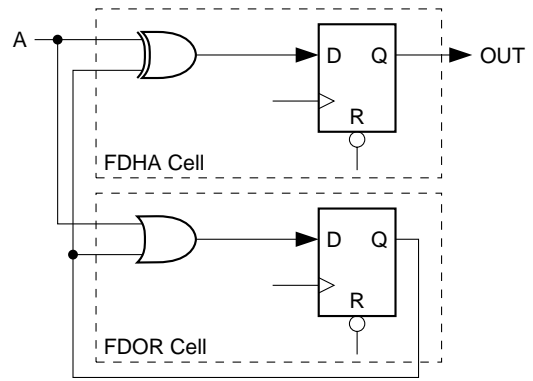
Bit-Serial Adder

A bit-serial adder is sometimes referred to as a carry-save adder because of the nature of its operation. Refer to the bit-serial adder schematic in Figure 4. It is constructed using a full-adder with registers on both its carry and sum outputs. The registered carry output is fed back to the carry input of the full-adder. In operation, the two words to be added are simultaneously shifted, least significant bit first, into the main inputs of the full-adder. The carry output from the addition of each bit is stored and used as the carry-in for the addition of the next bit. Essentially, the carries remain stationary as the inputs are shifted through the adder. The output of the circuit is registered, performing bit

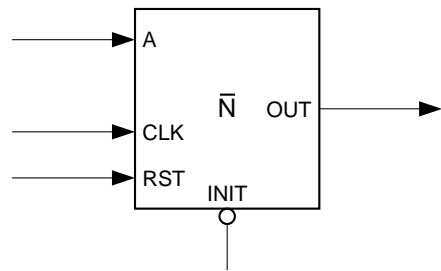


SYMBOL

Asynchronous Initialization Version



SCHEMATIC



Synchronous Initialization Version

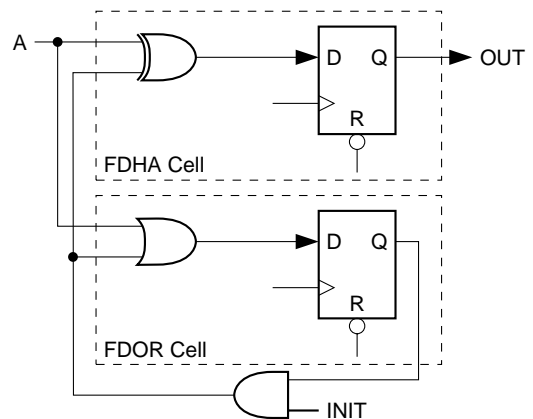


Figure 5. Two's Complementor

pipelining. The latency is one bit-time. The bit-serial adder must be initialized before each new word by clearing the carry save register or adding an AND gate to the carry feedback path. This is demonstrated in the two sets of symbols and schematics shown in the figure. The asynchronous version uses less area but can impact device utilization by consuming the reset terms for each column of D flip-flops comprising the adder. This compromise is eliminated by the use of the synchronous initialization scheme. Initialization insures that the first (LSB) addition is performed properly. The input words must be of the same length. Three AT6000 core logic cells are used to implement the bit-serial adder. The FDHA cell illustrates the AT6000 compute-orientation, a registered half-adder in a single cell, provides a fundamental building block for addition, accumulation, and counting.

Bit-Serial Two's Complementor

The second fundamental function is a circuit which computes the two's complement of the input word, shown in Figure 5. The serial algorithm for two's complement generation starts with the least significant bit, copies each bit until the first one is encountered and then all remaining bits are

inverted. The circuit uses two flip/flops, an XOR gate and an OR gate. The detection flip-flop must be reset before each input word, since it causes the XOR to invert the input continuously after the first logic one is detected. The output is registered, so the function has a one bit-time latency. As in the bit-serial adder, there are two means in which LSB initialization is achieved; the asynchronous and synchronous versions.

Delays

The last elementary function is the bit delay (a bit time is one clock cycle). The delay is required to align words, produce word delays needed for many algorithms, and to propagate control signals that must also be aligned. Word delays are usually implemented as shift-registers of the proper length. As will be shown later, depending on the input word length, appropriate delays are required between arithmetic operations. The Atmel Component Generator can automatically produce shift-registers of any length, reducing the design task to mere specification.

Higher-Level Functions

Most of the higher level math functions are constructed from the above elements. It can be shown that many cus-

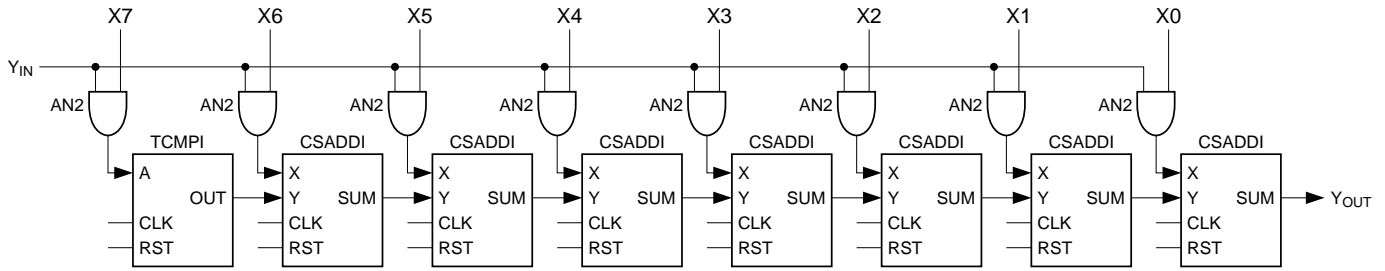


Figure 6. 8-Bit Serial-Parallel Multiplier

Other math functions can be constructed using the basic building blocks of bit-serial arithmetic, such as accumulators, deductors, squaring circuits, etc. The FIR filter requires multipliers and a serial column adder, which adds the tap values simultaneously. Their operation is discussed below.

Serial-Parallel Multiplier

Multipliers are essential to most digital signal processing algorithms. The simple serial-parallel multiplier (SPM) is particularly well suited for FPGA implementation because all of its routing is to nearest neighbor cells with the exception of the input. This multiplier has one serial input, one parallel input (used for coefficients), and a serial output.

The number of building blocks is a function of the width of the parallel input. An 8-bit SPM is shown in Figure 6. This multiplier performs the familiar shift-add algorithm: the parallel input is multiplied by each bit of the serial input as it is shifted in, and each partial product is added to the shifted accumulation of the previous products. The bitwise products are the logical AND of the input bit with each bit of the parallel input. The shifting accumulator is constructed by chaining a series of bit-serial adders together so that the inputs are bit parallel and the sum is downshifted on each clock cycle. The serial output is taken from the output of the least-significant-bit adder. The output has the same weight as the previous serial input bit with a latency of one bit-time. The number of bits in the output is equal to the sum of

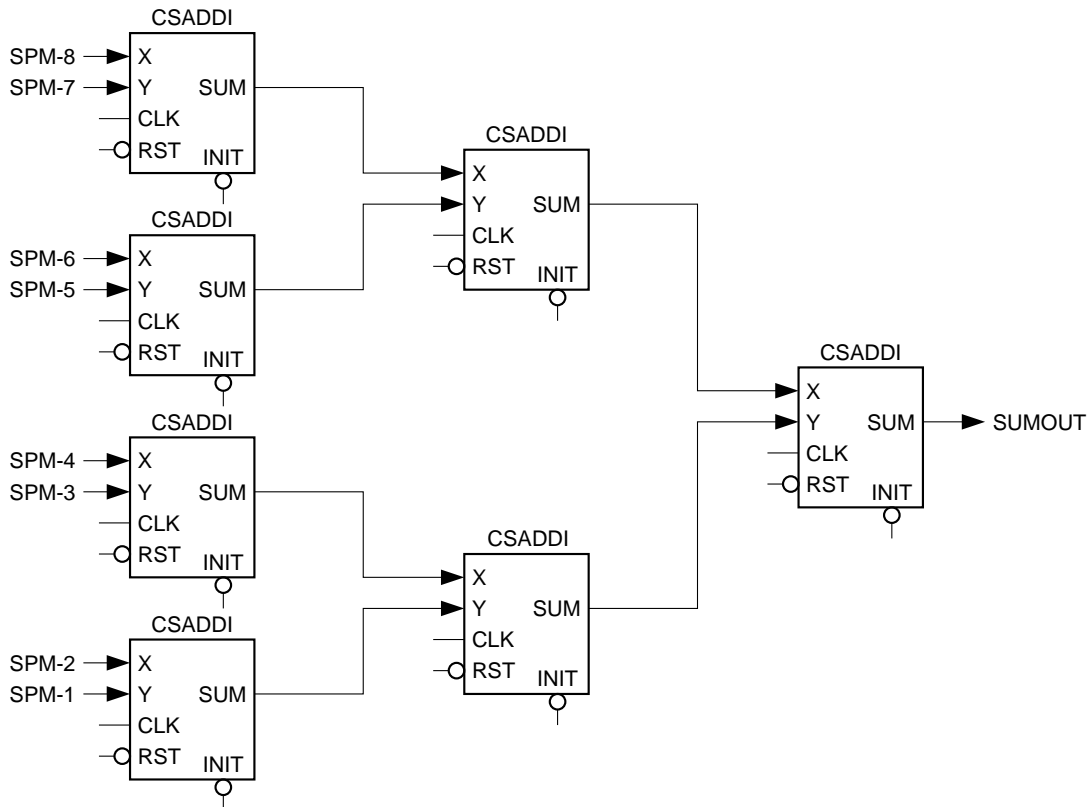


Figure 7. 8-Input Serial-Column Adder

the number of serial input bits and the parallel input bits. Since the serial input has to be of the same length as the output, it is sign-extended.

The SPM must be cleared before a new word is input to prevent errors. This is especially true if the X (parallel) input is negative since the two's complement circuit cannot self clear. No other controls are required. The serial input has to be sign-extended by the number of bits in the parallel input (i.e., to make the number of bits in the serial input equal to the number of bits in the output). The output is always a full precision output.

Serial-Column-Adder

An adder structure capable of simultaneously adding more than two inputs is a desirable function. This is easily accomplished by a tree of serial adders. Each serial adder (carry-save adder) combines two input streams into one output, hence each level of the tree structure reduces the number of serial streams by half, adding one bit-time of latency in the process. A serial column adder (SCADD) designed in this way allows an arbitrarily large number of inputs to be summed together without a sacrifice in the bit rate. If an odd number of inputs exist in a level, the odd input can be passed on to the next level via a delay register to maintain the bit-alignment. If overflow is to be avoided, one bit of growth must be allowed for each level in the adder. Since the input and output must have the same number of bits, the input must be sign-extended (guard bits) to prevent overflow. The number of levels and hence the latency and number of guard bits for an n-input serial column adder is equal to $\text{Log}(n)$ rounded up to an integer. As with single bit-serial adders, the inputs and outputs run least-significant-bit first. The 8-input SCADD used in this design appears in Figure 7. The initialization signal resets the bit-serial adders. The control signal, INIT, is delayed and applied to the successive stages insuring that they are timed properly.

Bit-Serial Digital Filter Reference Designs

The table in Figure 8 shows the performance and characteristics of the digital filter reference designs that are available as hard macros. The device utilizations summaries for each hard macro follows below.

The specifications and schematics shown below denote 8-bit versions of generic digital filters that are comprised of generated components produced using tools in the current release of the ATMEL IDS4.0 software. The files on the enclosed disk include a library of bit-serial functions and building blocks such as the carry-save adders and the two's complementors. Combined with serial-parallel multipliers created by the Component Generator, a wide variety of bit-serial signal generation and processing algorithms can be implemented.

As can be seen from these examples, the "wiring" concentration is minimal; point-to-point in most cases. Routing capacity is usually never a problem for bit-serial designs. Efficient macros and their placement is more important. The library elements and the Component Generator supply very compact macros. Thus, hardware parallelism can be easily exploited to compensate for the slower processing speed. However, the performance is fairly impressive; consider that eight (8-bit x 8-bit) multiply operations, seven 16-bit additions, seven 20-bit data-shift operations, and two I/O-transfer operations at 5M samples/sec. equals 120-Mops (Mega-operations per second).

The bit-serial system processes data LSB first. At the beginning of each process cycle, an initialization pulse is produced by the control logic. This pulse resets the saved-carry in the arithmetic elements. Each level in the tree of adders contains 1-bit-time of latency, hence, the initialization control pulse is delayed for each level. The width parameter of the SPM (serial-parallel multiplier) component generator corresponds to the coefficient width. The input data stream can be of any width - controlled by the length of the data shift-registers, which act as temporary data stor-

Filter	Taps	Type	Data Width	Coefficient Width	Internal Precision	Output Precision	Sample Rate (max)	System Clock
FIR8	8	Std	8 Bits	8 Bits	18 Bits	8-18 Bits	5 MS/sec	100 MHz
FIR8S	8	Sym	8 Bits	8 Bits	18 Bits	8-18 Bits	5 MS/sec	100 MHz
FIR16S	16	Sym	8 Bits	8 Bits	20 Bits	8-18 Bits	5 MS/sec	100 MHz
FIR24S	16	Sym	8 Bits	8 Bits	22 Bits	8-18 Bits	5 MS/sec	100 MHz
FIR32S	32	Sym	8 Bits	8 Bits	22 Bits	8-18 Bits	5 MS/sec	100 MHz
FIR64S	64	Sym	8 Bits	8 Bits	24 Bits	8-18 Bits	5 MS/sec	100 MHz
IIR20	N/A	2nd Order	8 Bits	8 Bits	18 Bits	8-18 Bits	5 MS/sec	100 MHz

Figure 8. Bit-Serial Digital Filter Specifications

age, and the control system timing that initializes the datapath components and maintains the I/O transfers.

The filters can be scaled for any number of parameter variations, including number of taps, input data-width, coefficient width, internal precision, and output precision. The FIR "S" type filters are symmetrical and designed to be cascaded, e.g., the FIR16S, FIR24S, FIR32S, and FIR64S are all built from the FIR8S filter block. Additional CS-adders are used to accumulate the data from all the blocks into the final output. The IIR Filter is a 2nd-order, direct form II type and can be cascaded or paralleled.

Bit-Serial Design Tips

The serial-parallel multiplier (SPM) Component Generator is designed so that one input is serial and can be of any length. The other input (usually for the coefficients) is specified to the generator. The product is shifted into a shift register, the length of which will determine how much of the product is used or stored. For example, if the input data is 8-bits and the coefficient is 12-bits, generate a SPM with a width of 12. Since the SPM always produces a full product, a 24-bit shift-register needs to be generated for the product. The input needs to be sign extended to support the extra shifts into the SPM (see the FIR application note for details). The basic system needs 24 clocks minimum to support the process (actually, if extra precision is required in the accumulate phase, then 2 or 3 extra clocks are needed as well as 2-3 three more SR stages). If internal sample storage is required as in filter designs, then the storage SR needs to be the same length.) Therefore, a 24-7 times 3Mhz (72-84MHz) bit-serial process clock is needed. It's best to use a PLL clock frequency synthesizer to generate the process clock from the word sample-clock.

Using the CSADDI (Carry-Save ADDer with Initialization) from the library with a shift-register will produce the accumulator. A simple counter or shift-register based controller will allow the generation of the timing pulses required to initialize the SPM and CSADDI circuits at the LSB time since the system processes the data LSB first.

General Applications

Traditionally, most digital filter applications have been limited to audio and high-end image processing. With advances in process technologies and digital signal processing methodologies, digital filters are now cost-effective in the IF range and in almost all video markets.

Digital filters are commonly used for audio frequencies for two reasons. First, digital filters for audio are superior in price and performance to the analog alternative. Second, audio analog-to-digital converters (A/Ds) and digital-to-analog converters (DACs) can be manufactured with high accuracy and are available at low cost. Thus, the combined cost of filtering and conversion (if necessary) is low.

Digital signal processing technology can now produce cost-effective digital filters for IF. New data conversion products have been introduced for communications systems. Using these data conversion devices, many digital radio systems are using digital filters for signal band-limiting and decimation. In these cases, the design engineer must not only know digital filters but also understand the effects of narrow-band-filtering processing-gain on A/D requirements. Additionally, power dissipation must be considered. Currently, digital IF filter solutions are excluded from low-power applications such as personal communication devices. However, for base stations, repeaters, and head-end equipment, FPGA-based signal processing can be a good choice.