ECU and DSP Function

Application Note 82

Abstract

The use of Field Programmable Gate Arrays (FPGAs) in Digital Signal Processing (DSP) applications is drastically increasing due to high performance, flexibility, cost effectiveness, and low power. Most DSP functions require high computational power and the capability to perform the experiment with algorithms. The FPGA is recommended to achieve these criteria. This document describes the use of Embedded Computational Units (ECUs) in QuickLogic FPGAs to implement DSP functions.

Introduction

Design engineers in diverse fields such as imaging, communications, multimedia, video applications, and mass storage are turning to DSP to enable or enhance a variety of advanced system features. Most engineers use specialized DSP processors for implementing DSP functions such as filters, correlators, sine/cosine building blocks, transforms, and math functions. They are increasingly using traditional FPGAs for DSP functions due to the capability to improve system performance, lower system power dissipation, and integrate more functionality.

Today's FPGA technology is advancing quickly in performance and density, which enables designers to perform billions of Multiply and Accumulate (MAC) operations per second. Unfortunately, most of the arithmetic functions use many logic cells, which requires bigger and faster FPGAs. With the larger device usage comes higher power consumption and the need for increased board area.

To overcome these limitations, some designers have begun using standard cell ASICs. These devices are very fast and efficient for arithmetic functions, but lack flexibility for quick design changes, tend to be more costly, and consume too much power, especially for portable applications such as cellular mobile communication and wireless LAN.

QuickLogic's Eclipse family of FPGAs combine configurable embedded functions with field programmable logic to provide the performance and efficiency of ASICs with the flexibility and short development cycles of programmable devices. This concept has been used to embed dedicated blocks of reconfiguarable ECUs into the silicon, specifically designed to perform high-speed arithmetic logic. By embedding dedicated logic and high-speed memory, the devices provide up to four times the computational bandwidth of similar-sized general-purpose logic devices without using any of the available logic cells. Algorithms and coefficients are loaded into memory to configure a versatile computational unit at consecutive stages for varying high-speed arithmetic operations.

Eclipse devices come with 10 to 18 ECUs. With the execution of a multiply and an addition in a MAC operation in a single clock cycle, Eclipse designs are capable of achieving an average of 2.6 billion MACs per second with the ECUs alone. Eclipse also comes with several thousand embedded logic cells that are capable of achieving 2 billion MACs per second to provide a combined total of 4.6 billion MACs per second.

Embedded Computational Units

The ECUs are intended to be used as building blocks for larger functions. However, they can also be used as stand-alone functional blocks. The key to the design flexibility of the ECUs is the easily accessible control signals. ECUs can be dynamically configured through their control signals S1, S2, and S3. These control signals are connected to the normal routing channels, which can also establish connectivity with the I/Os if the designer wants to drive the control signals externally. The corresponding functions of the ECU for each different configuration of S1, S2, and S3 are listed in **Table 1**.

Table 1: S1, S2 and S3 Versus ECU Configuration Modes

S/N	S1	S2	S3	Operations
1	0	0	0	Multiply
2	0	0	1	Multi-Add
3	0	1	0	Accumulate
4	0	1	1	Add
5	1	0	0	Multiply (registered)
6	1	0	1	Multiply –Add (registered)
7	1	1	0	Multiply-Accumulate
8	1	1	1	Add (registered)

Figure 1 illustrates the ECU functional block diagram and **Table 2** illustrates the available ECU count in the Eclipse family.

Figure 1: ECU Functional Block Diagram

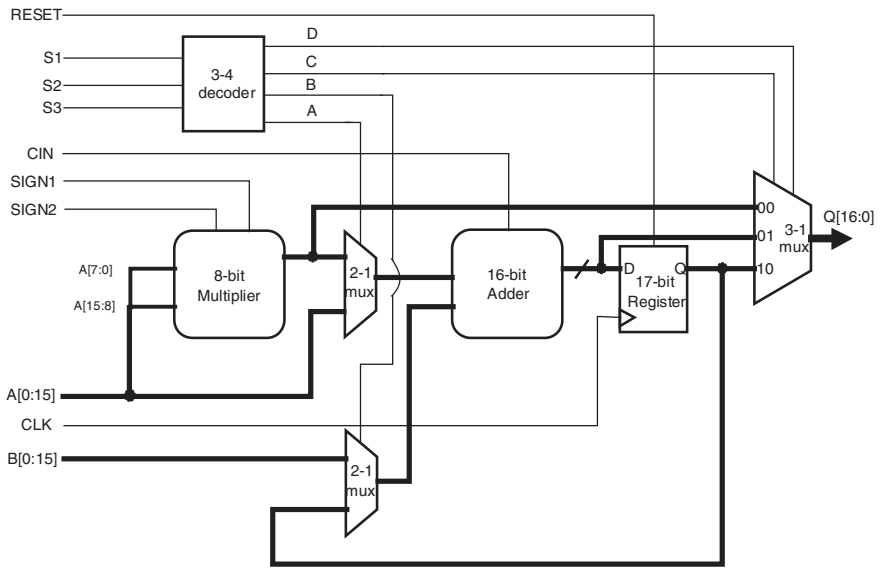


Table 2: ECU Count in the Eclipse Family of FPGAs

Part Number	Number of ECU
QL6250E	10
QL6325E	12
QL7100	10
QL7120	12
QL7160	16
QL7180	18
QL8250	10
QL8325	12

ECUs are placed in a row configuration in between the memory block and the logic cell array for maximum flexibility. This ensures fast and efficient memory/instruction fetch and addressing of DSP algorithmic implementations. After processing, data can either be routed back into memory or directly into the logic cells. The approach of embedding the ECU into the silicon guarantees performance of arithmetic functions such as single cycle, zero clock latency MACs (8-bit) at 144 MHz and Adds (16-bit) at 396 MHz.

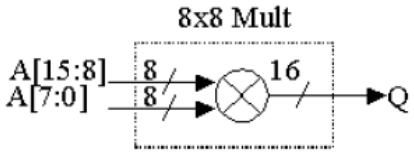
ECU Configurations

The following are the different ECU configurations based on **Table 1**.

Multiplier

When S1, S2, and S3 = “000”, the ECU behaves as an 8x8 multiplier. The output is a 17-bit result, however only 16-bits are valid in this configuration. Therefore, to output only 16-bits through the I/Os, declare 16-bit ports at the top-level VHDL entity or Verilog HDL module. The sign bits (SIGN1 and SIGN2) are explained in **Adder (Registered)** on page 6. **Figure 2** illustrates the 8x8 multiply function.

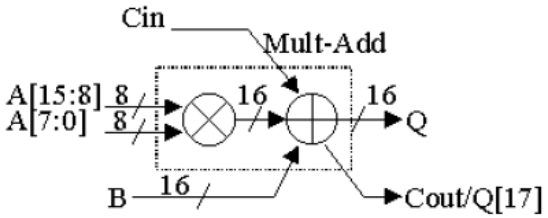
Figure 2: 8x8 Multiplier



Multiplier and Adder

As depicted in the **Figure 3** this configuration allows an 8x8 Multiply operation followed by a 16-bit adder operation. The 16-bit adder also takes care of a 1-bit Carry-in bit (Cin). In this configuration the output is 17 bits which includes the Carry-out bit (Cout)/Q [17].

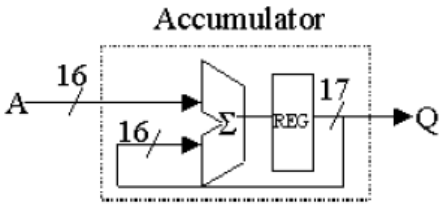
Figure 3: Mult-Add



Accumulator

The ECU can be transformed into a 16-bit-accumulator with one overflow bit. The input is 16 bits and the output is 17 bits as shown in **Figure 4**.

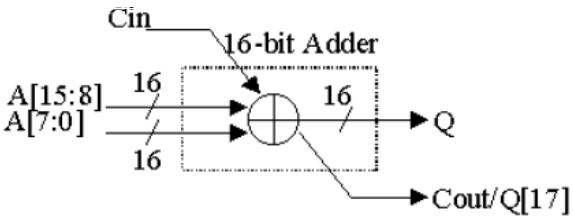
Figure 4: Accumulator



Adder

Figure 5 shows a 16-bit adder that accepts 1-bit Carry-in (Cin) and generates a 17-bit output. The MSB output is the Carry-out bit (Cout).

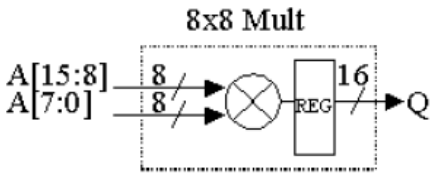
Figure 5: 16-bit Adder



Multiplier (Registered)

This 8x8 multiplier function is exactly the same as mentioned in **Multiplier** on page 3. The only difference here is the presence of the output registers. **Figure 6** shows the registered multiplier function.

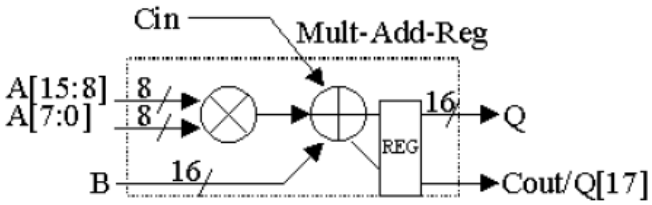
Figure 6: 8x8 Multiplier (Registered)



Multiplier and Adder (Registered)

The basic function of the registered multiplier and adder configuration shown in **Figure 7** is the same as the **Multiplier and Adder** on page 4. The only additional feature is the registered output.

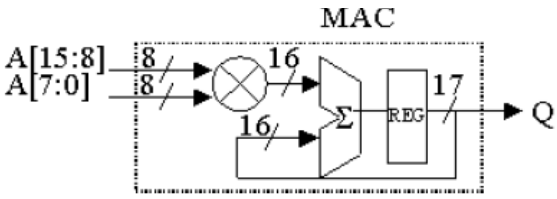
Figure 7: Mult-Add (Registered)



Multiply Accumulate (MAC)

This MAC function consists of an 8x8 multiplier and a 16-bit accumulator as described in **Accumulator** on page 4. The output is a registered 17-bit as shown in **Figure 8**.

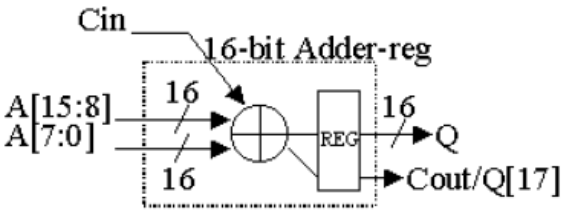
Figure 8: MAC Function



Adder (Registered)

The root function is a 16-bit adder that is the same as the basic **Adder** on page 4. The only difference is the registered output as shown in **Figure 9**.

Figure 9: Adder (Registered)



The 8x8 multiplier in the ECU has two sign bits. They are named SIGN1 and SIGN2 in the VHDL and Verilog HDL models of the ECU. These sign bits control how the 8x8 multiplier in the ECU interprets the values of the multiplicand and the multiplier. SIGN1 affects the interpretation of the 8-bit multiplier input to the ECU while SIGN2 affects the interpretation of the 8-bit multiplicand input to the ECU. **Table 3** illustrates the exact interpretation of the values of the multiplicand and the multiplier.

Table 3: Sign-bit

Sign-bit	Multiplier	Sign-bit	Multiplicand
SIGN1	A [7:0]	SIGN2	A [8:15]
0	(x00 – x7F) => (0 – 127) (x80 – xFF) => (128 – 255)	0	(x00 – x7F) => (0 – 127) (x80 – xFF) => (128 – 255)
1	(x00 – x7F) => (0 – 127) (x80 – xFF) => (-128 – -1)	1	(x00 – x7F) => (0 – 127) (x80 – xFF) => (-128 – -1)
0	(x00 – x7F) => (0 – 127) (x80 – xFF) => (128 – 255)	1	(x00 – x7F) => (0 – 127) (x80 – xFF) => (-128 – -1)
1	(x00 – x7F) => (0 – 127) (x80 – xFF) => (-128 – -1)	0	(x00 – x7F) => (0 – 127) (x80 – xFF) => (128 – 255)

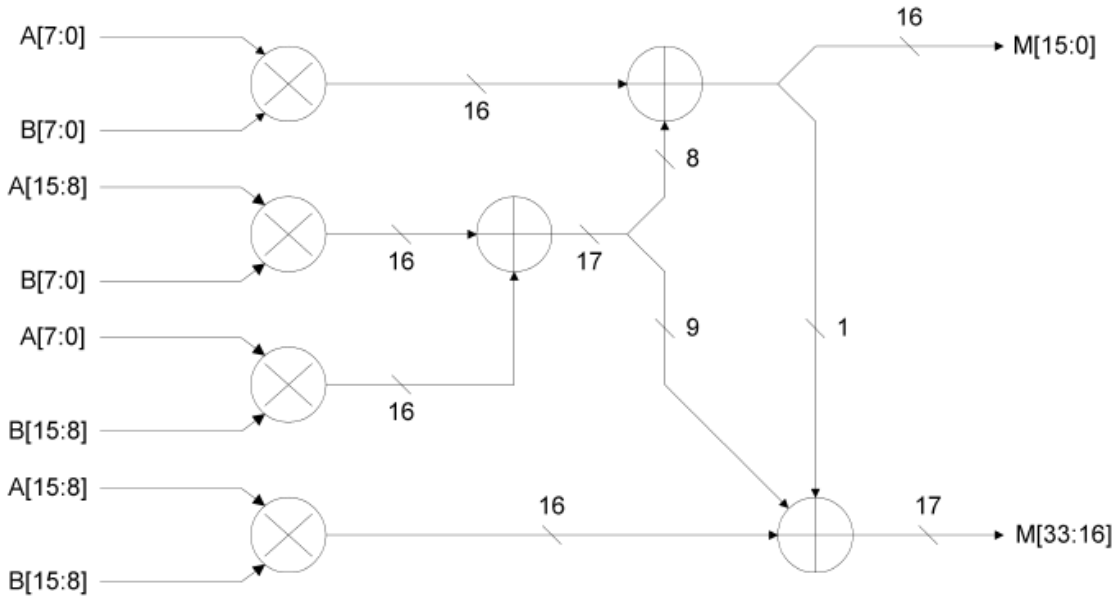
Therefore, when SIGN1 and SIGN2 are equal to 1, the multiplicand and multiplier are treated as signed binary. In other words, they must be compiled to the 2's complement format. The resulting output should be interpreted as 2's complement binary number. On the other hand, when SIGN1 and SIGN2 are equal to 0, the multiplicand and multiplier are treated as an unsigned binary. In the same manner, the resulting output should be interpreted as an unsigned binary (the whole 16-bit output is treated as the magnitude). Sign-bit is not intended to be interpreted as a sign-bit for a sign magnitude binary. Any sign-magnitude binary multiplicand and multiplier should be converted to 2's complement format before they can be input to the 8x8 multiplier for signed multiplying.

The sign-bit introduces flexibility in terms of sign-extending the multiplicand and the mulitplier independently using SIGN1 and SIGN2. This allows the building of large 16x16, 24x24, and 32x32 multipliers. The HDL behavioral model of the ECU block is located in the default directory `C:\pasic\spde\data\ecu.vhd` or `C:\pasic\spde\data\ecu.v`. The ECU entity or module must be instantiated in the design code when it is used to build larger functions as appropriate according to the designer's application.

Building Larger Arithmetic Structures

Larger multipliers can be constructed by using multiple ECUs. For instance, a zero-latency 16-bit multiplier can be constructed using four ECUs as shown in **Figure 10**.

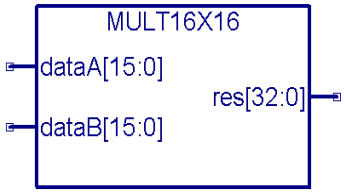
Figure 10: 16x16 bit Multiplier



Wider adders can be constructed using multiple ECUs. For example, a 32-bit adder can be built using two ECUs. One ECU implements the addition for the lower 16 bits and the other ECU implements the upper 16 bits. The carry-out of the lower ECU is connected to the carry-in of the upper ECU to provide the carry bit. Much larger adders are built in a similar fashion.

The following are some of the large arithmetic functions generated using multiple ECUs in QuickWorks. More of these types of functions are located in the default directory `C:\pasic\LIB\ECU`.

16X16 Unsigned Multiplier

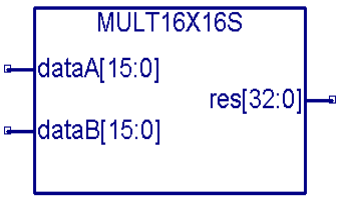


Inputs: dataA [15:0], dataB [15:0]

Outputs: res [32:0]

Multiplier	Multiplicand	Result
dataA	dataB	res [31:0] = dataA x dataB

16X16 Signed Multiplier



Inputs: dataA [15:0], dataB [15:0]

Outputs: res [32:0]

Multiplier	Multiplicand	Result
dataA	dataB	res[31:0] = dataA x dataB

24x24 Unsigned Multiplier

Inputs: dataA [23:0], dataB [23:0]

Outputs: res [47:0]

Multiplier	Multiplicand	Result
dataA	dataB	res[47:0] = dataA x dataB

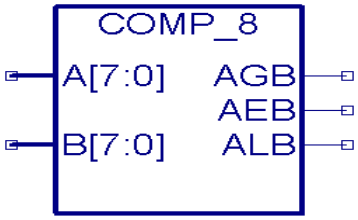
24x24 Signed Multiplier

Inputs: dataA [23:0], dataB [23:0]

Outputs: res [47:0]

Multiplier	Multiplicand	Result
dataA	dataB	res[47:0] = dataA x dataB

8-Bit Comparator

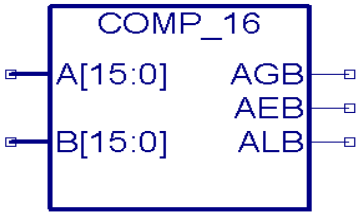


Inputs: A [7:0], B[7:0]

Outputs: AGB,AEB,ALB

Input Conditions	AGB	AEB	ALB
A > B	1	0	0
A = B	0	1	0
A < B	0	0	1

16-Bit Comparator

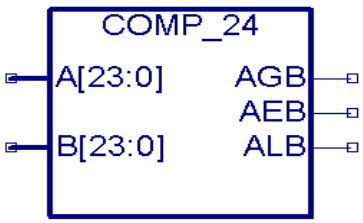


Inputs: A [15:0], B[15:0]

Outputs: AGB,AEB,ALB

Input Conditions	AGB	AEB	ALB
A > B	1	0	0
A = B	0	1	0
A < B	0	0	1

24-Bit Comparator

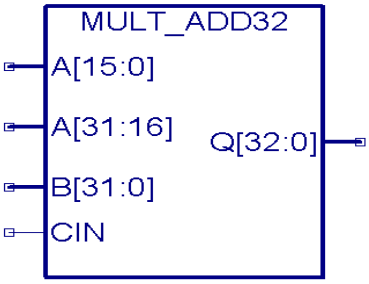


Inputs: A [23:0], B [23:0]

Outputs: AGB, AEB, ALB

Input Conditions	AGB	AEB	ALB
A > B	1	0	0
A = B	0	1	0
A < B	0	0	1

16x16-Bit Unsigned Multiplication and 32-Bit Addition

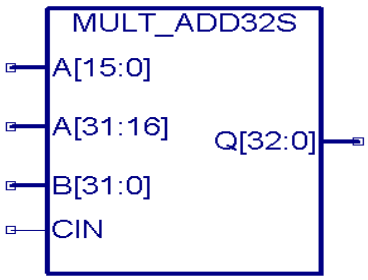


Inputs: A [15:0], A [31:16], B [31;0], CIN

Outputs: Q[32:0]

CIN	A[15:0]	A[31:16]	B[31:0]	Q[32:0]
L	A[15:0]	A[31:16]	B[31:0]	A[15:0] x A[31:16] + B[31:0]
H	A[15:0]	A[31:16]	B[31:0]	A[15:0] x A[31:16] + B[31:0] + 1

16x16-Bit Signed Multiplication and 32-Bit Addition



Inputs: A [15:0], A[31:16], B[31;0], CIN

Outputs: Q[32:0]

CIN	A[15:0]	A[31:16]	B[31:0]	Q[32:0]
L	A[15:0]	A[31:16]	B[31:0]	A[15:0] x A[31:16] + B[31:0]
H	A[15:0]	A[31:16]	B[31:0]	A[15:0] x A[31:16] + B[31:0] + 1

Digital Signal Processing Function

DSP applications utilizing multipliers in signal and data processing include filtering, image processing, video compression, etc. High computational power is needed to achieve real-time performance while processing the signal and data. Therefore, designers should keep their options open for hardware implementation of arithmetic intensive functions such as multipliers and adders since implementation of these functions in an ECU can yield much higher speed performance. Furthermore, the ECUs are easily accessible via a sea of high performance FPGA gates and dual-port embedded RAM blocks. Such a device can deliver performance, flexibility, parallel processing, and multi-channel processing capabilities.

The following section will explain the various commonly used DSP applications in which a designer can effectively utilize the ECUs for the building blocks of multipliers, adders comparators, MACs, etc. To reduce the time to market, designers can also use the built-in arithmetic functions (explained in **Building Larger Arithmetic Structures** on page 7) located in the default directory `C:\pasic\LIB\ECU`.

Finite Impulse Response Filter

In many of today's DSP applications, there is a need to be able to adapt to unknown conditions. The use of a fixed coefficient Finite Impulse Response (FIR) filters in many cases is not sufficient. DSP applications such as echo cancellation, estimation, and detection must use filters that behave differently depending on the conditions of the channel they are in. Therefore, an adaptive filter is the natural choice for these types of applications.

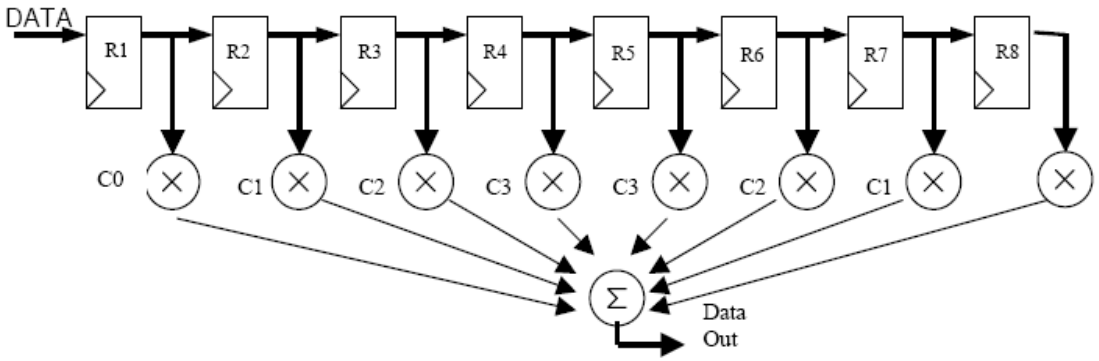
An adaptive FIR filter is an FIR filter with coefficients that can be dynamically configured by an adaptive algorithm. There are many different adaptive algorithms available that are tailored to different applications. One of these algorithms is the Least Mean Square (LMS) algorithm, which is discussed in **Implementation** on page 13.

FIR Filter Architecture

Figure 11 is an example of the 8-tap FIR filter architecture. The input is shifted through 8 registers (taps). Each output stage of a particular register is multiplied by a coefficient.

The resulting outputs of the multipliers are then summed to create the filter output.

Figure 11: FIR Filter Traditional Architecture, 8 Tap



The equation for a FIR filter is:

$$y(n) = \sum_{n=1}^{N} x(n)w(n)$$

Where:

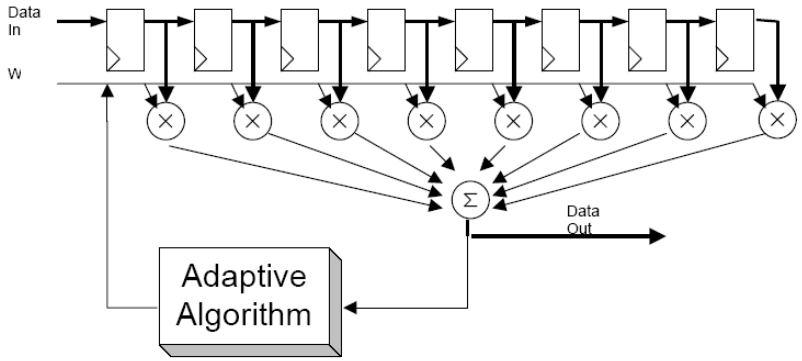
x(n) = input data sequence
w(n) = coefficients of the weight
y(n) = output sequence of the filter

In many cases, the coefficients of the filter are known beforehand. If this is the case, the implementation of the FIR filter is less complex. In many new applications the coefficients are not known and must be updated constantly. In this case, the filter must allow the designer to program the coefficients, which can be achieved by using a programmable coefficient FIR filter or an adaptive FIR filter.

The architecture of the programmable coefficient FIR filter allows the designer to use it as an adaptive filter with any type of algorithm that the designer chooses to implement. This algorithm can be implemented in the programmable logic portion of the Eclipse family of devices. **Figure 12** shows a block diagram of the architecture of an adaptive FIR filter with programmable coefficients or weights (W).

To make a conventional FIR filter into an adaptive one, add an extra block to the design that will contain the adaptive algorithm that will change the coefficients or weights on the fly. There are many different adaptive algorithms; each one of these algorithms has its own tradeoffs, such as convergence rate versus the complexity of the design.

Figure 12: Adaptive FIR Filter Traditional Architecture, 8 Tap



Implementation

FIR filters can be implemented in schematics or in a hardware description language. Eclipse devices contain up to 18 ECU blocks and up to 662,208 system gates, which allows great flexibility for the design. For example, a FIR filter can be implemented using the ECUs and an adaptive algorithm can be implemented using the logic cells. Another option is to use the ECUs to perform the multiplications and to use the logic cells for the addition and the remaining control logic. There are many other combinations that allow the designer to make optimal use of the device for their own specifications.

In the sample design, the multiplications and the additions are implemented by using the ECUs and the shifting procedure, as well as the adaptive algorithm by using the programmable logic portion.

The adaptive algorithm used is the LMS algorithm. The equations for the LMS algorithm are:

$$e(n) = d(n) - y(n)$$

$$w(n + 1) = w(n) + 2^{*}\beta^{*}e(n)^{*}x(n)$$

Where:

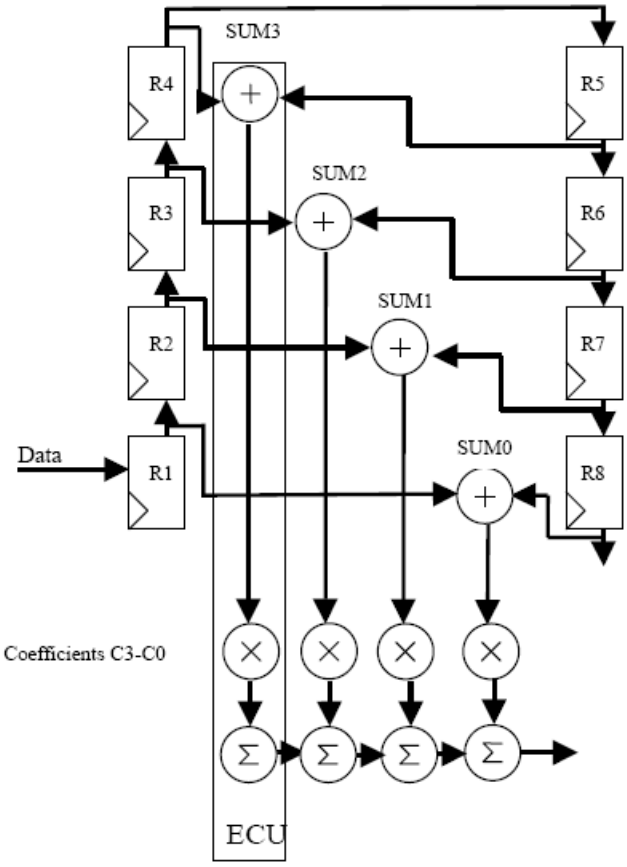
$e(n)$ = error signal
$d(n)$ = desired output signal
β = constant that determines the rate of convergence

The first equation determines the error in the output by subtracting the output signal of the filter from the desired output signal. This desired output signal is also called a training sequence for the adaptive algorithm. Once the error is determined, the second equation will use this error and calculate the new coefficients or weights that will correct or minimize the error. Each time the weights are updated the error is minimized until it converges.

Linear Response Filters

For linear phase response FIR filters, the coefficients are symmetric about the center taps. Thus, the filter can be "folded" in half thereby reducing the number of MACs required as shown in **Figure 13**.

Figure 13: Folding the Filter



Conclusion

FPGAs are being used in a wide variety of signal processing applications due to their high performance, low cost, flexibility, and low power consumption. DSP applications can be implemented efficiently in QuickLogic's Eclipse family of devices due to the flexibility of logic cells and ECUs. The approach of embedding computational units and memory blocks into silicon allows DSP design engineers to efficiently implement complex algorithms and multiple-sample processing across single or multiple data paths without sacrificing performance.

References

- Parallel Processing Via QuickDSP Devices, QuickLogic Application Note 33.
- Medical Imaging Using QuickDSP Devices, QuickLogic Application Note 34.
- Creating Adaptive FIR Filter Using QuickDSP, QuickLogic Application Note 49.
- Comparison Between ECU and LUT Approach to Build Large Multiplier, QuickLogic Application Note 51.
- ECU in QuickDSP, QuickLogic Application Note 52.
- QuickDSP Complex Multiplier, QuickLogic Application Note 53.
- FIR Filter Implementation, QuickLogic Application Note 23.

Contact Information

Telephone: (408) 990 4000 (US)
(416) 497 8884 (Canada)
+(44) 1932 57 9011 (Rest of Europe)
+(49) 89 930 86 170 (Germany & Benelux)
+(8621) 6867 0273 (Asia)
+(81) 45 470 5525 (Japan)

E-mail: info@quicklogic.com

Support: http://www.quicklogic.com/support

Web site: http://www.quicklogic.com/

Revision History

Revision	Date	Originator and Comments
Rev. A	June 2004	Haris Akkol and Kathleen Murchek

Copyright and Trademark Information