

## 2 Pipelining and DSP

1

In Section 1, I looked at various different approaches to “state variable assignment” and hinted at the strong relationship with different circuit architecture. In Section 2, I will look at the approach of data pipelining which is frequently used in Digital Signal Processing applications.

# Digital Signal Processing

- Convolution (filtering)
- Correlation
- Fourier Transform
- Etc.

2

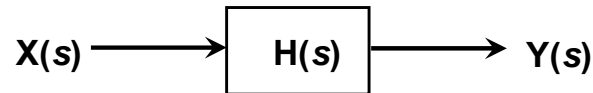
During your degree, you have come across a pile of mathematics for Signal Processing. You certainly know about the three above - write a list here of other techniques you know about. In the core course, you learnt to do these things in continuous time. Depending on which options you have taken, you may have learnt quite a bit also about the mathematics of discrete-time signal processing .

Looking back, it should not come a huge shock to you to be told that almost no-one actually performs signal processing in continuous time.

Digital Signal Processing is a whole lot easier to implement and is hugely important in many real industrial applications. I want in this section to look at some of the implications for electronics design.

# The analogue viewpoint - 1

- Analogue filters are most naturally designed in the frequency domain



$$Y(s) = H(s).X(s)$$

Note: for AC analysis we can set  $s = j\omega$

3

Let's think about filter design. For continuous-time signals, you have learnt the value of frequency responses. The concept of a Transfer Function is a very powerful one.

## The analogue viewpoint - 2

- That's because the time domain is hard work!

$$y(t) = h(t) * x(t) = \int_{-\infty}^{\infty} h(t - \tau) \cdot x(\tau) d\tau$$

**The impulse response**

**Convolution**

If  $x(t) = \delta(t)$ , the "unit impulse" (zero except at  $t = 0$  where  $\int_{-\epsilon}^{\epsilon} \delta(t) \cdot dt = 1$ ), then  $y(t) = h(t)$ , the "impulse response"

**Causality:**

usually  $h(t) = 0$  for  $t < 0$

so  $y(t) = \int_{-\infty}^t h(t - \tau) \cdot x(\tau) \cdot d\tau$

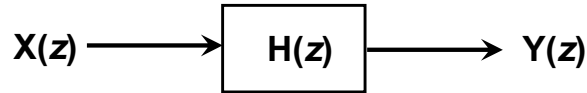
4

In contrast, dealing with the time domain is very messy and hard and involves performing convolution integrals.

The concept of "Causality" is simply that for real-time processing we can't expect a response until after we have put something in .

Of course it can be that we are not doing our processing in real time, e.g. we might capture a complete frame of video before we start processing it. In this case causality is no longer relevant and the upper limit of our integral can indeed be greater than  $t$ . However, in practice we can't usually go as far as infinity!

# The digital viewpoint - 1



$$Y(z) = H(z).X(z)$$

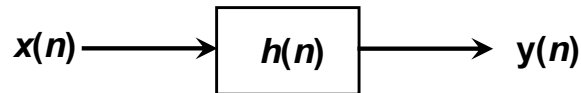
- Just like the analogue domain!  
( $z = e^{Ts}$  – we'll come back to that later)
- BUT, in the computer, we are usually dealing with samples  $x(n)$ ,  $y(n)$ , *etc.* .....

5

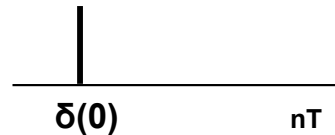
The concepts of the **frequency domain** and transfer functions is equally applicable to the digital world as it was in the analogue world. We just use a frequency variable  $z$  instead of  $s$ . Mathematically  $z = e^{Ts}$ . This is the basis of the  $z$ -transform, but don't worry, I'm not going to test you on that!

However, it turns out that the **time domain** is a good deal easier to handle in the digital world than it was in the analogue world ....

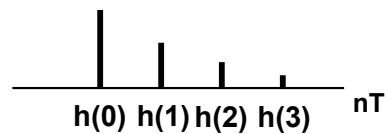
## The digital viewpoint - 2



Consider when  $x(0)$   
is a unit impulse  
 $\delta(0)$ :



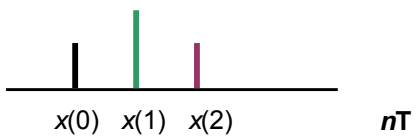
Then  $y(n) = h(n)$  is the  
impulse response,  
say:

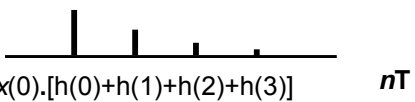


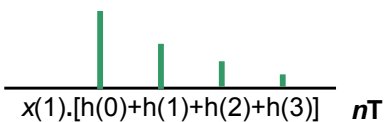
6

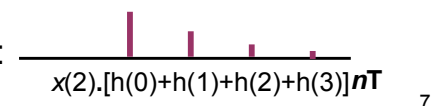
An impulse in the digital world is just the value at one sample time. If we apply an impulse to the filter, we get a series of values out at successive time steps and these constitute its “impulse response”.

## The digital viewpoint - 3

Suppose  $x(n)$  is the signal: 

The response to  $x(0)$  is: 

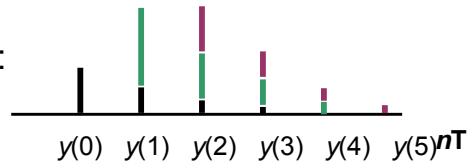
The response to  $x(1)$  is: 

The response to  $x(2)$  is: 

So if we apply a succession of impulses, each one will produce an impulse response at the corresponding time and corresponding amplitude ...

## The digital viewpoint - 4

... and the overall sum  
of these responses is:



where  $y(0) = h(0).x(0)$

$$y(1) = h(1).x(0) + h(0).x(1)$$

$$y(2) = h(2).x(0) + h(1).x(1) + h(0).x(2)$$

...

$$y(n) = \sum_{i=0}^n h(n-i).x(i)$$

← That's easy to do in a computer!  
Technically it's "CONVOLUTION"  
but we'll just call it a digital filter

8

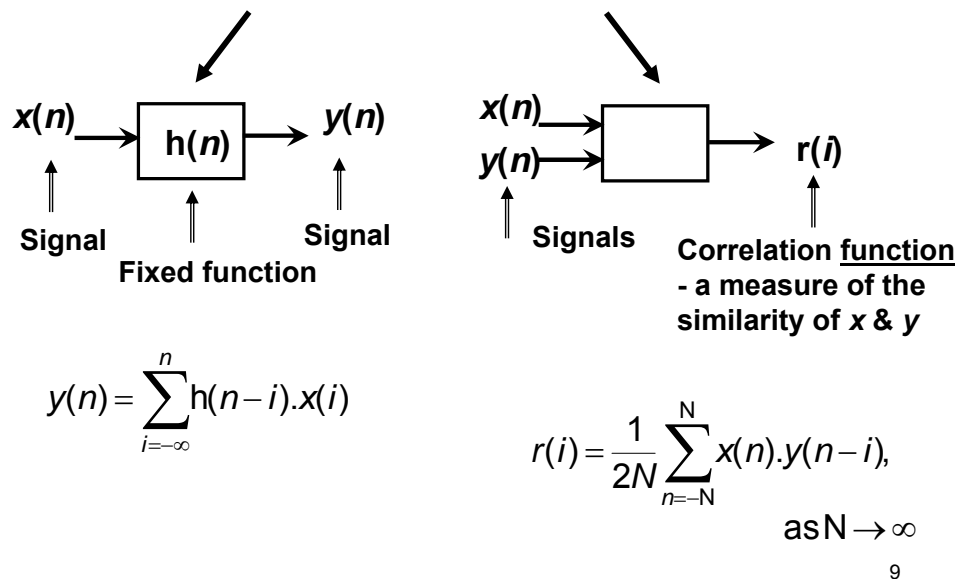
Since we deal almost all the time with Linear Systems, we can just add up the individual responses to produce the overall response.

That's "Convolution"!

You see that convolution in the digital domain is real simple stuff and so much easier that it was in the analogue world! Actually this multiplying and adding is just what a computer is so very good at and this bottom equation (the convolution summation) is exactly the way it would implement the digital filter.



# Convolution and Correlation



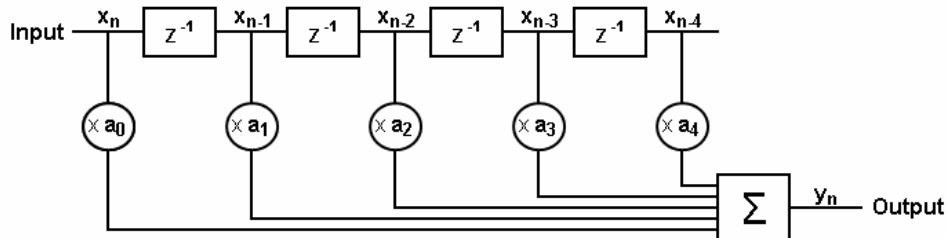
Some of you may well been a bit hazy on Convolution and Correlation so let's just recap.

In principle, convolution is involved with taking one signal and producing another. We have seen that it is just a digital filter, no more and no less. Usually the filter is fixed so that the coefficients **h** are constant.

In principle, correlation is about taking two signals and comparing them to produce a measure of their similarity. If the statistics of the signals remain constant (*i.e.* they are “stationary”), the correlation output will be a fixed function of the time offset between the signals (**i**).

In our applications, it is often case that the statistics are not stationary and we have to keep recomputing new sets of  $r(i)$ . Indeed, it is probably more common that one of the input signals is held constant (*e.g.* we send out repetitive radar pulses and then correlate the received radar reflection against the same template every time). That's really confusing because now  $y(n-i)$  is constant and  $r(i)$  is changing and the computer finishes up doing the same job as it did for convolution!

## Finite impulse response (FIR) or “non-recursive” filters



- $y(n) = \sum_{k=0}^N a_k x(n-k)$
- Causality
- Delay/latency

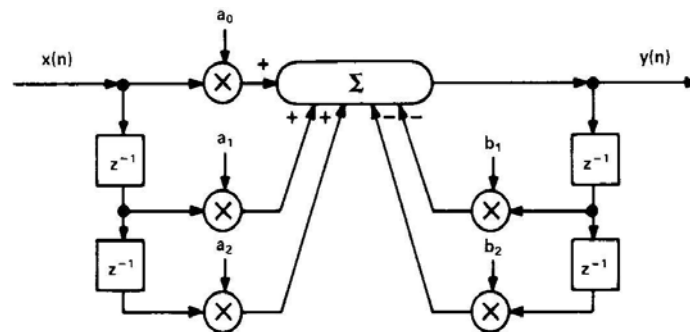
10

Here's a simple digital filter. The boxes marked as  $z^{-1}$  are sample period delays (e.g. registers). The notation arises because Laplace transform theory tells us that  $z^{-1} = e^{-Ts}$  is the transfer function of a delay by  $T$ .

Because the data flow is straight through with no feedback, it is a “non-recursive” filter and its impulse response will obviously consist of just five successive outputs. Because of that, we call it a “Finite Impulse Response” filter.

## Infinite impulse response (IIR) or “recursive” filters

- In a “recursive” filter the output depends on previous outputs (as well as previous inputs)  
$$y(n) = \sum_{k=0}^N a_k x(n-k) - \sum_{k=1}^M b_k y(n-k)$$
- *In principle* it has an “infinite” impulse response.



11

As you might expect, the recursive filter is more powerful but it turns out that it has more subtle problems of stability etc..

Mathematically, the recursive filter allows us control of both the poles and the zeros whereas the non-recursive filter only allows us to choose the zeros.

[Of course, when we implement this with a computer the impulse response will eventually get so small that it cannot be computed so it is not really infinite in the end.]

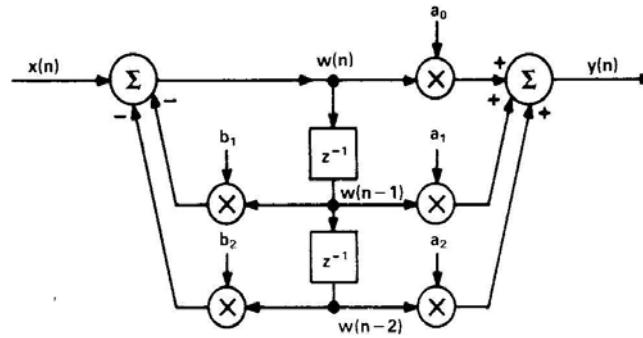
# Advantages of FIR filters

- Linear phase
- Unconditionally stable
- Accuracy / Insensitivity
- No limit cycles
- Easy to design

# Advantages of IIR filters

- Fewer coefficients
- Fewer variables to store
- Computationally efficient
- Lower delay
- Closer to analogue concepts

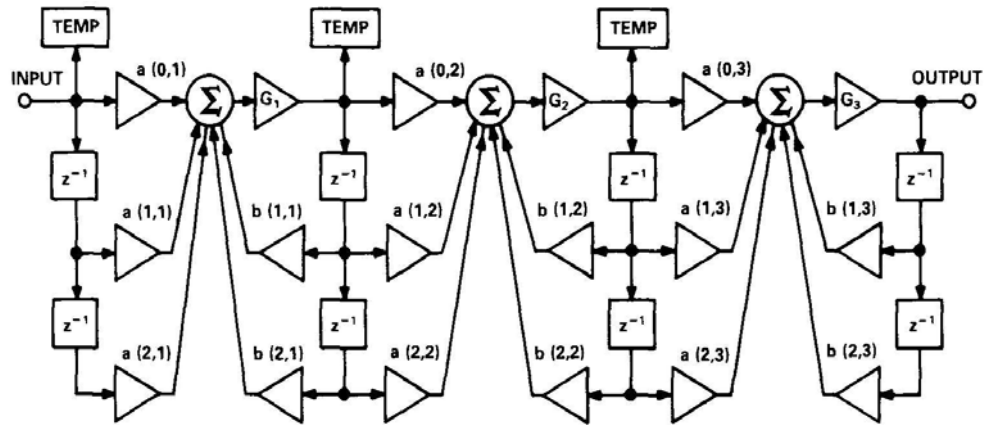
# Canonical Form



14

By unravelling the transfer function in a slightly different way with an intermediate variable  $W$ , we have a slightly more efficient realisation (it requires less memory/registers) which is called the canonical form.

## Cascade of “bi-quad” sections



15

Here we cascade three quadratic IIR filters to obtain a 6<sup>th</sup> order IIR filter.

# Key functions

- Add
- Multiply
- Add
- Multiply
- Add
- Multiply!
- ...
- Division, comparison, sorting

16

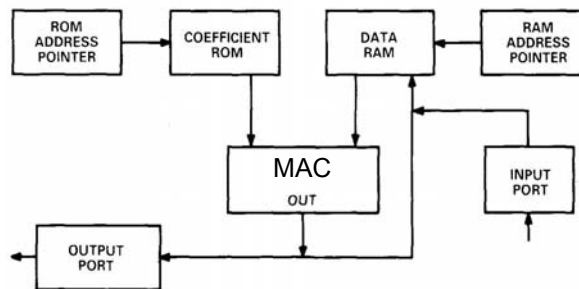
This is not a course on DSP; the point I want to make is that a huge number of DSP algorithms are based on adding and multiplying.

Because of this we will spend some time examining the options available for these functions.



# Implementation by a “DSP”

SECTION LOCATION	RAM			ROM	
	SECTION 1	SECTION 2	3	LOCATION	VALUE
1	X(1,1)			1	a(0,1)
2	X(2,1)			2	a(1,1)
3	Y(1,1)	X(1,2)		3	a(2,1)
4	Y(2,1)	X(2,2)		4	b(1,1)
5		Y(1,2)	X(1,3)	5	b(2,1)
6		Y(2,2)	X(2,3)	6	a(0,2)
7			Y(1,3)	7	a(1,2)
8			Y(2,3)	8	a(2,2)
9	TEMP	TEMP	TEMP	9	b(1,2)
				10	b(2,2)
				11	a(0,3)
				12	a(1,3)
				13	a(2,3)
				14	b(1,3)
				15	b(2,3)



17

Of course we could implement our signal processing with a **general purpose microprocessor**, and if the performance is satisfactory, that may well be the best way to do it.

If we want a bit more horsepower, we can buy a **Digital Signal Processor** from (e.g.) the likes of Analog Devices or Motorola. It comes with one or more very fast single-clock-cycle Multiply Accumulate units (MACs), with parallel data paths for data and coefficients and a specialist address generator to make sure that the right data and coefficients are loaded every clock cycle.

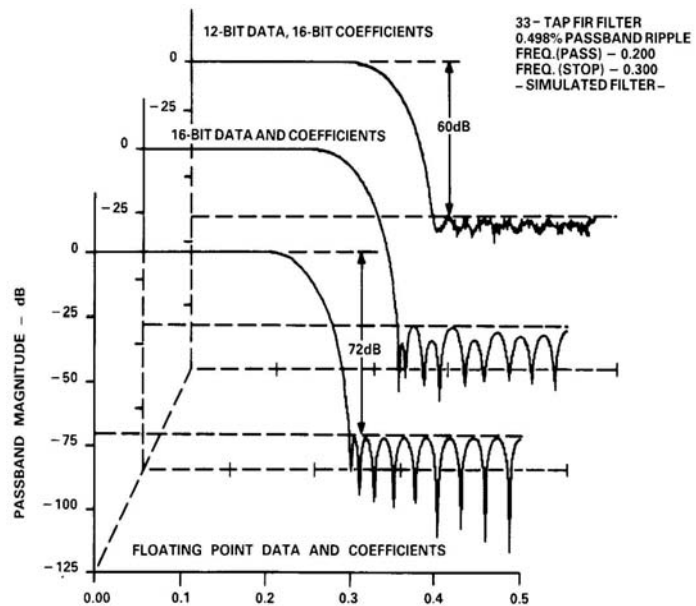
# Fixed-point arithmetic

- Simpler
- Faster
- Cheaper
- Smaller
- Lower Power

18

You can choose DSPs which use floating-point arithmetic or ones which use fixed-point arithmetic. The latter are simpler and therefore both cheaper, faster, *etc.*. They are therefore more popular, but you do have to simulate the design to be sure that you have scaled the signals properly and that you have enough precision (you can do that quite easily in Matlab).

# Precision



Finite precision arithmetic can have quite an impact on the desired result as this slide illustrates.

In practice we would be wise to simulate the result to see if it satisfies our requirements.

# Word-length

- Telephone: 12-bits
- HiFi: 16-bits
- BUT THINK:  
    noise!  
    scaling!  
    precision!
- 24-bits?

20

Typical word-lengths are given here but we need a bit of headroom on top of this as we are to avoid accumulating errors. In the case of fixed point arithmetic we also need to be rather careful about the scaling of the variables and/or to have further headroom to avoid saturation or loss of resolution.

# High Performance?

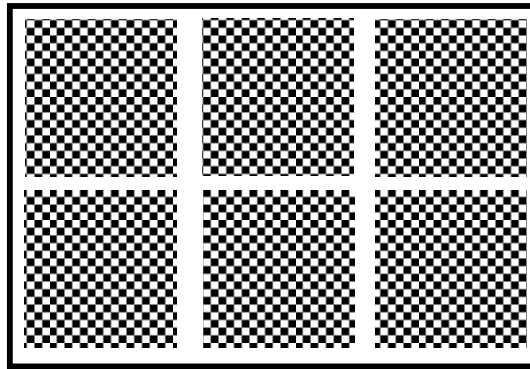
- Make only what you need
- Do things in parallel
- Pipeline
- Don't move data further/more often than you need

21

Whatever the technology we use, there are four key things we can do to get high performance from our electronics.

# Parallelism

- e.g. Image processing

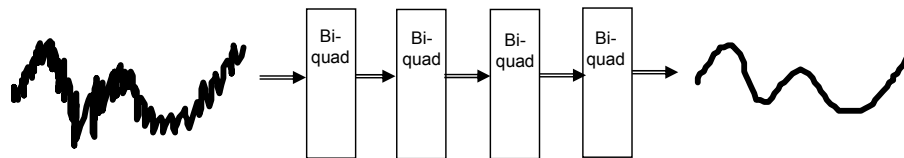


22

Many DSP functions can be split up into parts so that we can use multiple copies of our hardware to compute the parts in parallel. We say  $n$  parts, we hope to speed up the processing by a factor  $n$ . This is a hugely important concept for high performance DSP.

We do however have to be careful that overheads do not erode the hoped-for advantage. For example, if we split up an image into sections and process them separately, we will have to spend time to stitch the results together at the end.

# Pipelining



- e.g. speech processing with 8<sup>th</sup> order low pass filter

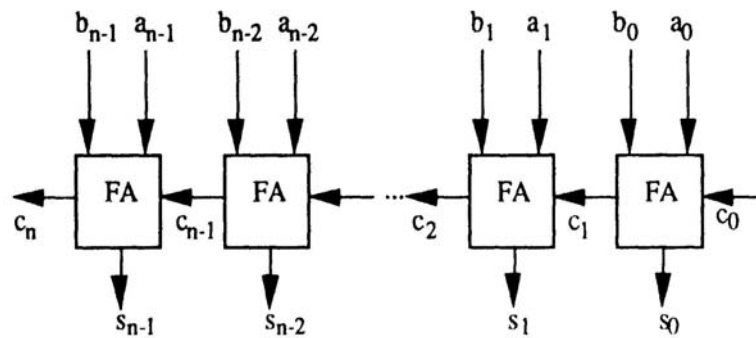
23

The other key concept to high performance DSP is the use of pipelining. For example if we want to implement a 8<sup>th</sup> order filter we can cascade four 2<sup>nd</sup> order sections together. Each 2<sup>nd</sup> order section works on a different data value and passes its results on to the next section. If (say) each section calculates its result in one clock cycle, the clock speed can be much faster than it would be if we had to calculate the whole 6<sup>th</sup> order filter in one go.

At the end of these notes I include a data sheet of a Lattice Semiconductor Corp “Serial FIR Filter” macro.

Of course the use of a pipeline will inevitably introduce a “latency” into the calculation. In the above example, the results will not start to be available until three clock periods after the relevant input.

# Carry-ripple adder



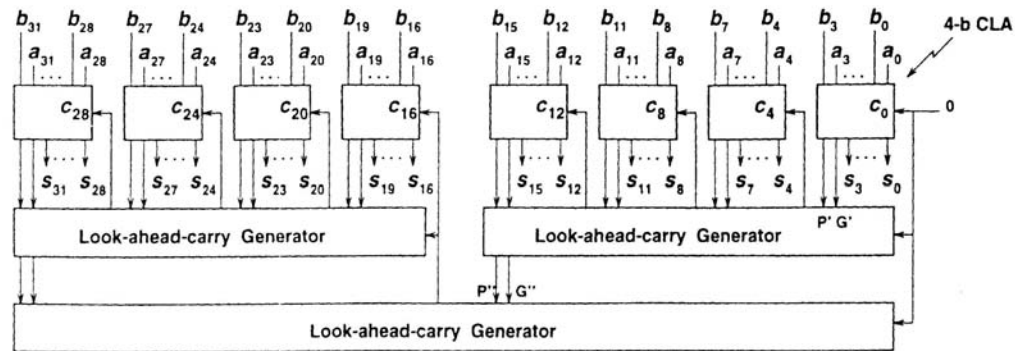
24

Let's look very briefly at some ways to implement the add & multiply functions.

To make an  $n$ -bit adder, you know that we can simply cascade a set of full-adders. The speed is determined by the time it takes for a carry to ripple from the l.s.b. to the m.s.b. so this will be slow.



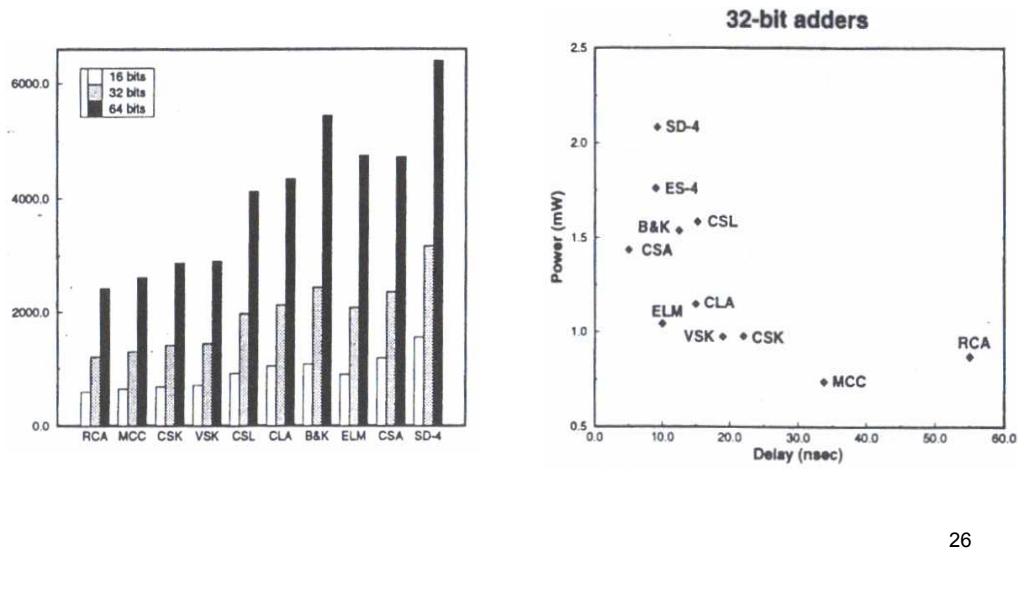
# Block carry look ahead



25

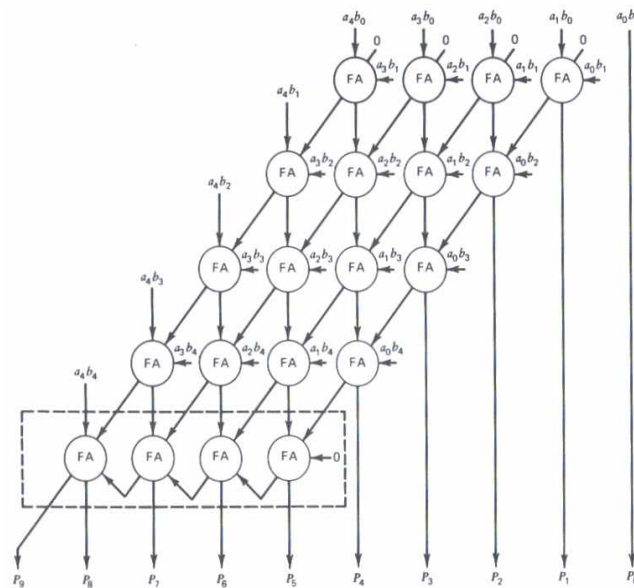
... so a common thing to do is to use “carry look ahead” – throwing more gates at the problem in order to get the carries out quicker. It’s a case of diminishing returns and if we need more than about 4 bits at a time we adopt a hierarchical approach as here.

# Adder performances



You would hardly believe how many different ways there are to add two numbers together. Of course some are more expensive than others (shown by the cost in terms of numbers of transistors) and each gives a different balance of power and speed. Therefore when we design our system, we have to choose which is best for our application. For example, the software that comes with FPGAs typically has a choice of several macros with different cost/benefits.

# Multiplication

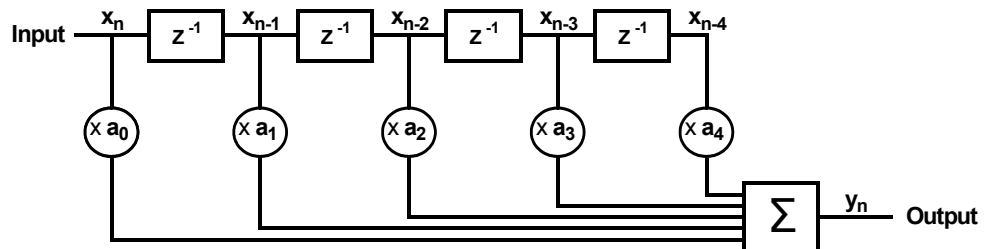


27

These days, the density of ICs is such that we can nearly always throw enough hardware at the problem that we can use array multipliers as in this diagram. The array corresponds to “long multiplication” so we can multiply in one cycle (albeit that the cycle has to be slow enough to allow the signals to propagate through the array). See that each row of the array is adding in A multiplied by one bit of B.

The part at the bottom inside the dotted line is a ripple carry adder and can be speeded up using one of the techniques previously discussed (e.g. using carry look ahead).

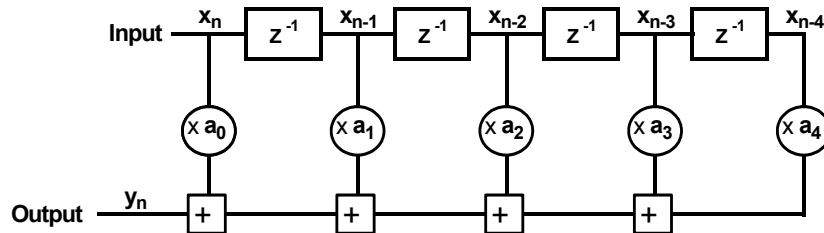
# FIR Filter



28

Lets look back at the FIR filter.

# FIR Filter - Implementation



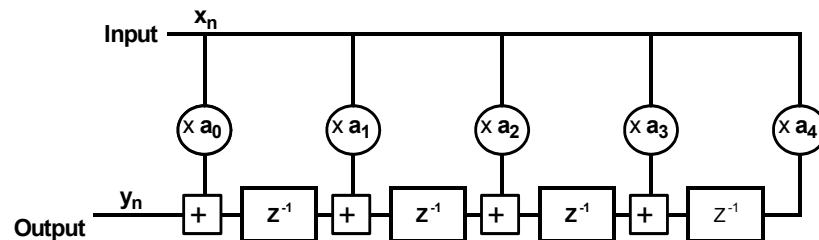
29

Here we break down the multiple summation with simpler adders.

Suppose we have just one multiply and add unit, then the five multiplications and four adds will take a minimum of five cycles (maybe more if we implement it with a microprocessor and have to store and recall the old values of  $x$ ).

On the other hand, if we have five multiply and add units, we can get a hardware implementation of the filter in just one cycle. However, the cycle length will need to be long enough for the completion of one multiply unit (all multiplications can be performed in parallel) and four addition units (which are in series).

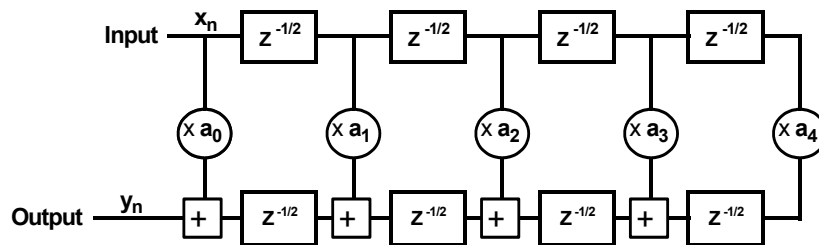
# FIR Filter - Pipelining



30

By moving the registers to the bottom, we get exactly the same function but now the clock cycle only needs to be long enough to complete one multiply and one add. That's pipelining!

## FIR Filter – “Systolic Array”



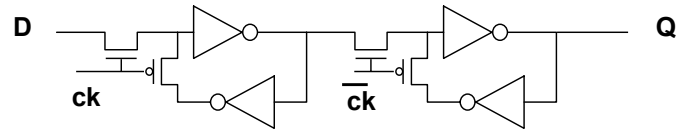
31

If the FIR filter has lots of taps, there will be a very high “fan-out” from the input  $x$  and this may well turn out to limit the speed.

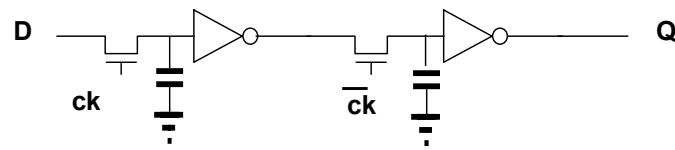
An alternative which gets around this is the so called “systolic array” which every path has register in it.

# CMOS Master-Slave Flip-Flop

**Static circuit**



**Dynamic Circuit**

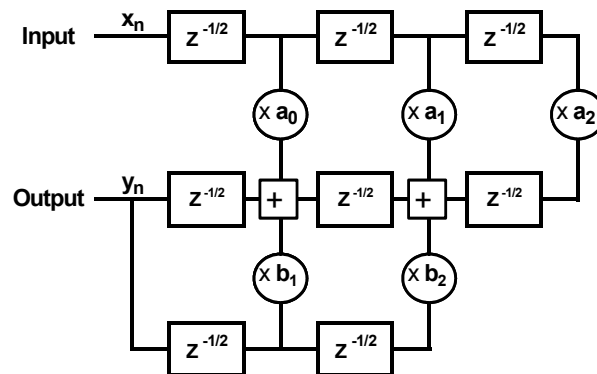


32

If a flip-flop is  $z^{-1}$ , then the master latch (and the slave latch) are each equivalent to  $z^{-1/2}$ .



# Bi-quad

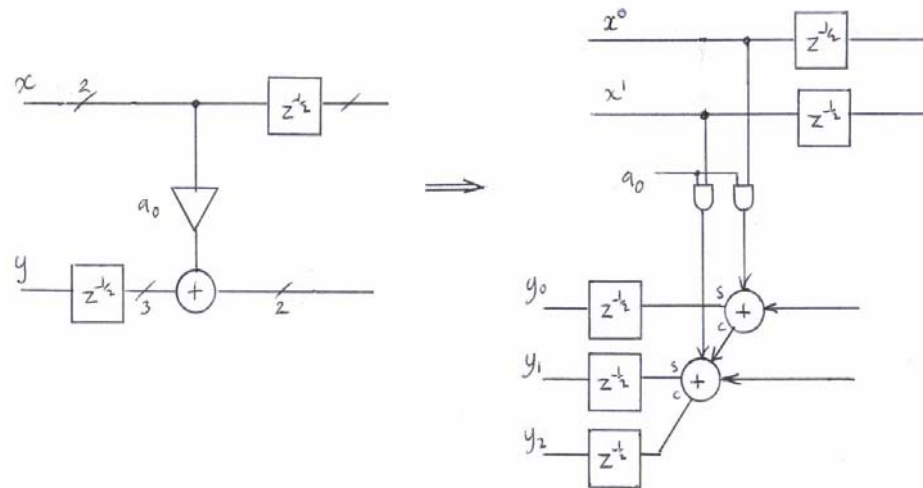


33

Here's the biquad section as a systolic array.

In CPLDs and FPGAs it is not usually possible to use  $z^{-1/2}$ . We can use  $z^{-1}$  in the diagram above, but then we can only feed in fresh data every other cycle. The notes at the rear from Xilinx show how to implement a systolic filter with their FPGA.

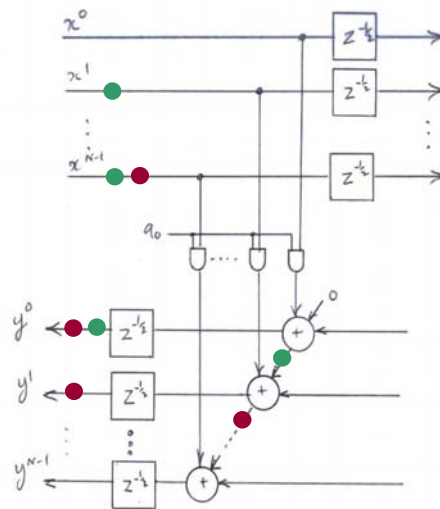
## Word/Bit-level array



34

Pipelining is usually done at the word level, but it turns out that for absolute maximum throughput, we can pipeline also at the bit level.

# Timing



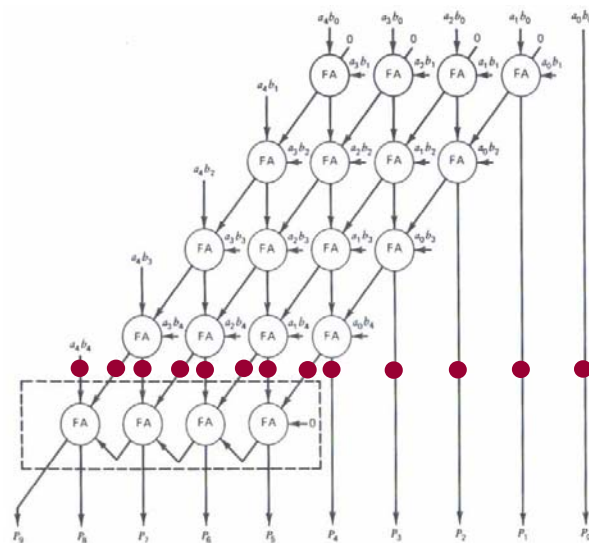
35

See here that the critical path will probably be the series of adders (effectively a ripple carry adder).

Pipelining at the bit level involves putting further flip-flops in the paths so that data has less to do in each clock cycle.

If we put flip-flops in one path, we have to put equivalent delays in all the other paths.

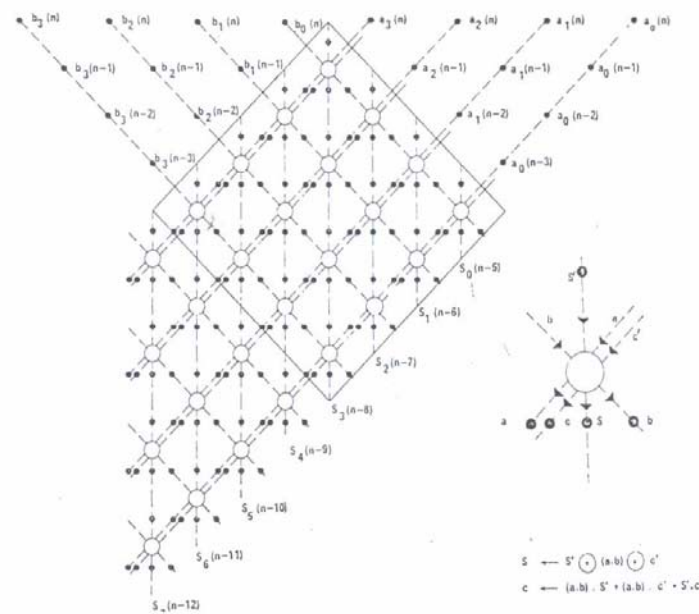
## Example - Multiplier



36

Let's look back again to the array multiplier. That's a great application of parallelism in hardware but we have a critical path of  $2n$  full adders in series. To get a faster clock cycle, we may therefore want to pipeline it too.

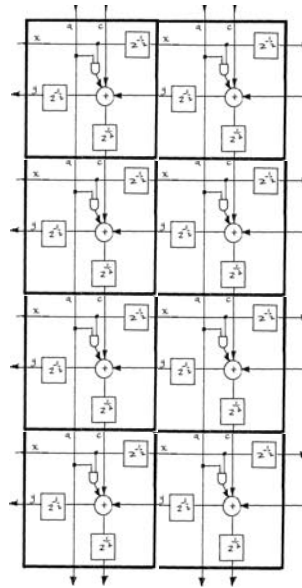
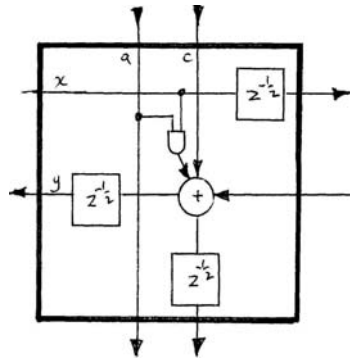
# Pipelined/Systolic Multiplier



37

In the extreme, we can pipeline so that the critical path is just one full adder.

# Regularity



38

Two other important aspects for efficient implementation of DSP functions are to extract the maximum regularity so that we have an efficient layout and to ensure that data only travels small distances in one clock cycle.

It may seem natural to implement an array multiplier in this way but in other cases it is much less obvious. Consider the bit-level FIR filter three slides back. With a bit of ingenuity we can see that the cell on the left can be used repeatedly to implement this filter.

# Choices

- Technology
- Arithmetic
- Algorithms
- Circuit style

39

Add your own notes.

# Circuit Choices

- Parallelism
- Pipelining
- Regularity
- Abutment
- Cell optimization
- Parameterization
- Embedded cores

40

Add your own notes



# **Additional Material on FPGAs**

**Will Moore**

[http://www.latticesemiconductor.com/products/fpga/xp/index.cfm?jsessionid=ba307d0f3e5b\\$0F\\$26\\$F](http://www.latticesemiconductor.com/products/fpga/xp/index.cfm?jsessionid=ba307d0f3e5b$0F$26$F)

## **Lattice Semiconductor Corp**

### **LatticeXP**

LatticeXP FPGA devices utilize a combination of non-volatile FLASH cells and SRAM technology to deliver a single-chip solution supporting "instant-on" start-up and infinite re-configurability. A non-volatile FLASH cell array distributed within the LatticeXP FPGA device stores the device configuration. At power-up the configuration is transferred from FLASH memory to configuration SRAM in less than 1mS providing an instant-on FPGA. In addition, LatticeXP FPGA devices provide security by eliminating the need for an external configuration bit-stream and by providing non-volatile security features.

Non-volatile, reprogrammable FPGAs are well suited for implementing system logic in a wide variety of end markets including communications, consumer, industrial, computing, military and automotive. These are especially well suited where there is a requirement for instant-on, reduced parts count, high-security or real time programming.

- Optimized FPGA Architecture for Low Cost Applications
  - Feature set optimized for high-volume, low cost applications
  - Low cost TQFP & PQFP Packaging
- Non-Volatile Reconfigurable FPGA
  - Instant On
  - High Security
  - No Boot Prom
  - SRAM and FLASH
- TransFR (TFR) Technology
  - Simplifies in-field logic updates
- Flexible sysIO Buffers & sysCLOCK
  - Supports LVCMOS, LVTTTL, PCI, LVDS, SSTL and HSTL
  - Up to 4 analog PLLs
- Dedicated sysDDR circuitry
  - Simplifies implementation of DDR memory interfaces
  - Operates up to 333Mbps
- ispLeverCORE Intellectual Property Support
  - Speeding up design cycle with ispLeverCORE Intellectual Property
- ispLEVER Design Tools
  - Easy to use SW package supports all Lattice FPGA and programmable logic devices
  - Evaluation boards available for easy review of your design implementations
- LatticeXP FPGA Applications

- LatticeXP FPGA devices are ideal for a variety applications in cost sensitive market segments such as Consumer, Automotive, Medical & Industrial, Networking, Computing

	<b>LFXP3</b>	<b>LFXP6</b>	<b>LFXP10</b>	<b>LFXP15</b>	<b>LFXP20</b>
<b>Vcc Voltage (V)</b>	1.2/1.8/ 2.5/3.3	1.2/1.8/ 2.5/3.3	1.2/1.8/ 2.5/3.3	1.2/1.8/ 2.5/3.3	1.2/1.8/ 2.5/3.3
<b>PFU Rows</b>	16	24	32	40	44
<b>PFU Columns</b>	24	30	38	46	56
<b># of PFU</b>	384	720	1216	1932	2464
<b>LUTs (K)</b>	3.1	5.8	9.7	15.4	19.7
<b>Dist. RAM(K bits)</b>	12	23	39	61	79
<b>EBR SRAM(K bits)</b>	54	72	216	324	396
<b># of EBR SRAM Blocks</b>	6	8	24	36	44
<b># of PLLs</b>	2	2	4	4	4
<b>Package</b>	100 TQFP 144 TQFP 208 PQFP	144 TQFP 208 PQFP 256 fpBGA	256 fpBGA 388 fpBGA	256 fpBGA 388 fpBGA 484 fpBGA	256 fpBGA 388 fpBGA 484 fpBGA
<b>Maximum User I/O</b>	136	188	244	300	340

## Features

- Serial Arithmetic for Reduced Resource Utilization
- Variable Number of Taps up to 64
- Data and Coefficients up to 32 Bits
- Output Size Consistent with Data Size
- Signed or Unsigned Data and Coefficients
- Full Arithmetic Precision
- Fixed or Loadable Coefficients
- Decimation and Interpolation
- Real or Complex Data
- Selectable Rounding
- Scaleable Outputs
- Multi-cycle Modes for Area/Time Tradeoffs
- Supports Symmetric or Anti-Symmetric Filters
- Optimization Based on Symmetry of Filter
- Fully Synchronous Design

## General Description

Many digital systems use filters to remove noise, provide spectral shaping, or perform signal detection. Two types of common filters that provide these functions are Infinite Impulse Response (IIR) and Finite Impulse Response (FIR) filters. IIR filters are used in systems that can tolerate phase distortion. FIR filters have an inherently stable structure, and are used in systems that require linear phase. This benefit makes FIR filters attractive enough that they are designed into a large number of systems. However, for a given frequency response specification, FIR filters are of higher order than IIR, making them computationally expensive.

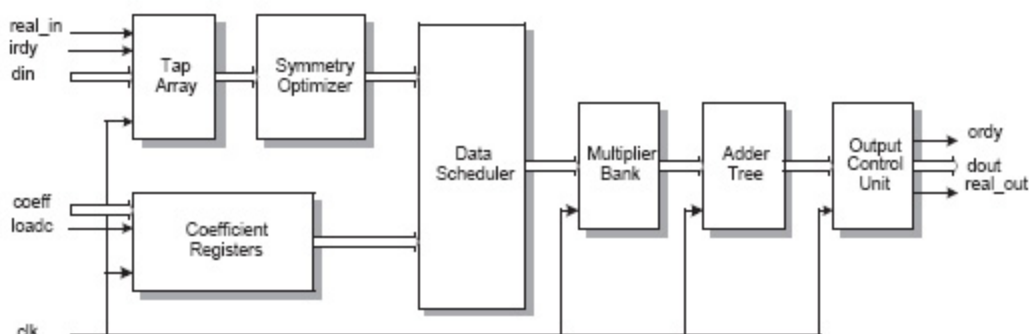
The Lattice Serial FIR filter uses serial arithmetic elements to achieve a compact size. Due to the serial nature of the arithmetic, the data rate is slower than the clock rate and dependant on the data width. The effective throughput is defined as:

$$\text{Data rate} = (f / (ofw + 1))$$

where *ofw* is the Output Full Width and *f* is the clock frequency.

## Block Diagram

Figure 1. Serial FIR Filter Core Block Diagram



The Lattice Serial FIR Filter IP core is based on a generic parameterizable model. This model generates a customized core with optimal architecture based on user parameters set at the time of core generation. The core supports a wide range of user-defined taps, data lengths and architectures.

The core supports a variable number of taps, from 4 to 64, which is sufficient for most applications. The core can also be used to implement decimation and interpolation filters. Additionally, the core can operate with complex or real data types. The coefficients can either be fixed at the time of core generation or dynamically loaded during runtime, thus allowing use in adaptive filtering. The architecture is optimized depending on the symmetry type of the filter. Symmetric odd and symmetric even types are recognized in addition to non-symmetric type.

## Functional Description

The Serial FIR Filter core is comprised of a number of interconnected functional blocks (Figure 1). Descriptions of these functional blocks are given below.

### Tap Array

The Tap Array module stores delayed versions or taps of input data. The number of taps of the FIR filter and the data width are user parameters and they are fixed at the time of core generation. The array consists of  $n$  taps each of width  $w$ , which are organized as shift registers. All data registers are reset when the `reset_n` input is asserted. At the end of each processing cycle, the data values are shifted into the next sequential shift register inside the Tap Array, with the first register getting the value from the input data port `din`.

### Symmetry Optimizer

The Symmetry Optimizer block is used for optimizing symmetric or anti-symmetric filter implementations. Symmetric filter implementations incorporate adders, while anti-symmetric filter implementations incorporate subtractors. For anti-symmetric filters, the output of this block has an additional latency of one clock cycle.

### Coefficient Registers

The Coefficient Registers module holds the coefficient data for the filter. The Serial FIR Filter coefficients can either be loaded at run time or set during IP core generation. This module supports real as well as complex coefficients. The serial FIR core can operate as fixed or loadable coefficients filter. For a fixed-coefficient filter, this module supplies the fixed coefficient value. For a loadable-coefficient filter, this module implements a series of word shift registers equal to the number of taps, (or equal to half the number of taps for symmetric or anti-symmetric filters).

### Data Scheduler

The Data Scheduler synchronizes the tap and coefficient data with the multiplier bank for multi-cycle filter implementations. In these implementations, the number of taps and the number of clock cycles determine the number of multipliers used. The Data Scheduler is also used to multiplex data for optimizing decimation and interpolation filters.

This block is not present in single-cycle filter implementations.

### Multiplier Bank

The Multiplier Bank receives data from the data scheduler and delivers output to the Adder Tree. The maximum delay through the Multiplier Bank is equal to the delay of a single multiplier. The number of multipliers is equal to the number of taps for single-cycle implementations.

### Adder Tree and Output Processor

The Adder Tree contains serial adders instantiated in a binary tree fashion. The Adder Tree receives input from the multiplier bank, and delivers output to the Output Processor. The Output Processor may contain an adder to support the following operations:

1. Accumulating the intermediate computation results in multi-cycle filter implementations.
2. Rounding to the nearest value, when output-scaling option is enabled.

In both cases, output latency increases by one clock cycle.

## Parameter Descriptions

Table 1 shows the parameter definitions for the Serial FIR Filter core.

**Table 1. Serial FIR Filter Parameter Definitions**

Parameter	Default Value	Value	Description
Filter Type	Single-Cycle	Single-Cycle, Multi-Cycle, Decimation or Interpolation	Type of filter selected by the user. This determines the rest of the parameter options.
Data Width	8 bits	Real: 4 to 32 bits Complex: 4 to 16 bits	Width of input data ( $W$ ) in bits. The width of the coefficients is also equal to this parameter. For complex data types, the Data Width is equal to the width of the real part and the range is from 4 to 16 bits.
Number of Taps	16	4 to 64	Number of taps ( $n$ ) in the filter.
Computational Cycles	2	2 to 32	Number of cycles ( $C$ ) for multi-cycle filters. Number of processing cycles (PP) to perform the filtering process. The output is computed once in $C$ PPs.
Decimation Ratio	2	2 to 32	Decimation is down sampling of the bit stream. In a decimation filter with decimation ratio ' $d$ ', the output data rate is $1/d$ of the input rate.
Interpolation Ratio	2	2 to 32	For interpolation filters. Interpolation is the reverse of decimation. The input rate is $1/U$ of the output rate.
Rounding Method	Nearest	Truncation or Nearest	Types of rounding available.
Arithmetic Type	Signed	Signed or Unsigned	Specifies the type of arithmetic modules for the core. If the symmetry of the core is even or odd, then the arithmetic type is always signed.
Data Type	Real	Real or Complex	Specifies the data type of the inputs ( $d1n$ and $coeff$ ) and the output ( $dout$ ) of the Serial FIR core. When complex data type mode is selected, the arithmetic type is always signed.
Complex I/O Mode	Parallel	Parallel or Serial	In the parallel I/O mode, real and imaginary parts are applied on the data bus in the same clock cycle. In the serial mode, real data is applied in the first PP cycles, followed by the imaginary data in the next PP cycles.
Output Width	Full Precision	4 to 96	Width of output data ( $W$ ) in bits. If the width is less than the maximum output width determined by the core generator, the outputs are scaled.
Coeffs Loadable	Run-time Loadable	Fixed or Run-time Loadable	Determines if the coefficients are run-time loadable. If the coefficients are run-time loadable, the core has two additional input ports, $coeff$ and $loadc$ , for loading purposes. If the coefficients are fixed during core configuration, no additional input ports are used.
Coefficients Format	Hexadecimal	Hexadecimal or Decimal	The coefficient values can be entered in either in hexadecimal or decimal format.
Symmetry	Even	None, Even, or Odd	Specifies the impulse response of the filter. Even Symmetry applies to symmetric impulse response, while Odd Symmetry applies to anti-symmetric impulse response. Decimation and Interpolation filters do not have Symmetry (the value None should be selected). If the Symmetry of the core is even or odd, then the arithmetic type is always signed.

## Signal Descriptions

Table 2 shows the definitions of the I/O ports.

Table 2. Input/Output Signal Descriptions

Name	Size	I/O	Description
clk	1	Input	<b>Clock.</b>
reset_n	1	Input	<b>Reset.</b>
din	4 to 32 bits	Input	<b>Data Input.</b> Data to be filtered.
irdy	1	Input	<b>Input Ready.</b> This signal indicates that the din port contains valid data to be filtered.
dout	4 to 128 bits	Output	<b>Data Output.</b> Data output after filtering.
ordy	1	Output	<b>Output Ready.</b> Filtered data is ready.
coeff	4 to 32 bits	Input	<b>Coefficient Data.</b> Coefficients loaded into the Serial FIR core for the filtering process. This signal is present only when the <b>Coeffs Loadable</b> parameter is set during core configuration.
loadc	1	Input	<b>Load Coefficient.</b> This signal indicates that the input coefficient data is valid. This signal is present only when the <b>Coeffs Loadable</b> parameter is set during core configuration.
real_in	1	Input	<b>Real Data Input.</b> This signal indicates that the real part of the input complex data is being applied. This is used only if the <b>Complex I/O mode</b> is serial.
real_out	1	Output	<b>Real Data Output.</b> This signal indicates that the real part of the output complex data is being output. This is used only if the <b>Complex I/O mode</b> is serial.

## Custom Core Configurations

For Serial FIR configurations not available in the Evaluation Package, please contact your Lattice sales office.

## Related Information

For more information regarding usage of the core, please refer to the Serial FIR User's Guide, available on the Lattice web site at [www.latticesemi.com](http://www.latticesemi.com).

## Appendix for ispXPGA™ FPGAs

Table 3. Performance and Resource Utilization

Parameter	LUT4s <sup>2</sup>	PFUs <sup>3</sup>	Registers	External Pins	System EBRs	f <sub>MAX</sub> <sup>1</sup>
fir_ser_xp_1_002.lpc <sup>4</sup>	260	115	382	41	None	185

1. Performance and utilization characteristics using LFX1200B-04FE680C in Lattice's ispLEVER™ v3.0 design tool. Synthesized using Synplify's Synplify Pro v.7.2.1. When using this IP core in a different density, package, speed, or grade within the ispXPGA family, performance may vary slightly.
2. Look-Up Table (LUT) is a standard logic block of Lattice devices. LUT4 is a 4-input LUT. For more information check the data sheet of the device.
3. Programmable Function Unit (PFU) is a standard logic block of some Lattice devices. For more information, check the data sheet of the device.
4. Configuration fir\_ser\_xp\_1\_002.lpc has the following settings: Number of Taps (n) = 16, Data Width(w) = 8, Coeff Width = 8, Output Width = 20, Signed, Single-Cycle, Real, Symmetric Coeff, Loadable Coeff (8 Coeffs).

## Supplied Netlist Configurations

The Ordering Part Number (OPN) for the Lattice Serial FIR Filter IP Core is FIR-SER-XP-N1. Table 4 lists the Lattice-specific netlists that are available in the Evaluation Package, which can be downloaded from the Lattice web site at [www.latticesemi.com](http://www.latticesemi.com).

Table 4. Parameter Values for Evaluation Configuration(s)

Parameter File Name	Input Data Width (Bits)	Number of Taps	FIR Type	Symmetry	Arithmetic Type	Data Type	Output Data Width (Full Data Width) <sup>2</sup>
fir_ser_xp_1_002.lpc <sup>1</sup>	8	16	Single cycle	Symmetric	Signed	Real	Full (20)

1. The latency for the fir\_ser\_xp\_1\_002 configuration is (6 + Output\_Full\_Width + 1) or 27.
2. The Output Data Width is the same as the Full Data Width.



## **Xilinx - XtremeDSP™ Slice**

---

Virtex-4™ FPGAs provide blazing DSP performance with unrivalled economy. The XtremeDSP slices available in all Virtex-4 family members facilitate new DSP algorithms and higher levels of DSP integration than previously available in FPGAs, while delivering low power consumption, very high performance, and efficient silicon utilization. With up to 512 XtremeDSP slices operating at 500 MHz, you can solve complex challenges, such as implementing:

- Hundreds of IF-to-baseband down-conversion channels
- 128X chip-rate processing for 3G spread spectrum systems
- High definition H.264 and MPEG-4 encode/decode algorithms

### **XtremeDSP Slices deliver high-performance, low power, versatility, and efficiency**

The XtremeDSP slice forms the basis of a versatile, coarse grain DSP architecture, enabling you to efficiently add powerful FPGA-based DSP functionality to your system.

- XtremeDSP Slices have been custom designed in silicon to achieve 500MHz performance independently or when combined together within a column to implement DSP functions.
- Each XtremeDSP Slice draws only 57μW/MHz, just 15% of the power consumption of previous FPGA DSP implementations.
- The XtremeDSP Slice supports over 40 dynamically controlled operating modes including; multiplier, multiplier-accumulator, multiplier-adder/subtractor, three input adder, barrel shifter, wide bus multiplexers, or wide counters.
- Cascade XtremeDSP Slices without using FPGA fabric or routing resources to perform wide math functions, DSP filters, and complex arithmetic.

### **Architectural highlights of the XtremeDSP slices:**

- 18-bit by 18-bit, two's complement multiplier with full precision 36-bit result, sign extended to 48 bits
- Three input, flexible 48-bit adder/subtractor with optional registered accumulation feedback
- Over 40 dynamic user-controller operating modes to adapt XtremeDSP slice functions from clock cycle to clock cycle.
- Cascading, 18-bit B bus, supporting input sample propagation
- Cascading, 48-bit P bus, supporting output propagation of partial results
- Multi-precision multiplier and arithmetic support with 17-bit operand right shift to align wide multiplier partial products (parallel or sequential multiplication)
- Symmetric intelligent rounding support for greater computational accuracy

- Performance-enhancing pipeline options for control and data signals are selectable by configuration bits
- Input port "C" typically used for multiply, add, large three-operand addition or flexible rounding mode
- Separate reset and clock enable for control and data registers



## Chapter 4

# Parallel FIR Filters

---

This chapter describes the implementation of high-performance, parallel, full-precision FIR filters using the DSP48 slice in a Virtex-4 device. Because the Virtex-4 architecture is flexible, it is practical to construct custom FIR filters to meet the requirements of a specific application. Creating optimized, parallel filters saves resources.

This chapter demonstrates two parallel filter architectures: the Transposed and Systolic Parallel FIR filters. The reference design files in VHDL and Verilog permit filter parameter changes, including coefficients and the number of taps.

This chapter contains the following sections:

- “Overview”
- “Parallel FIR Filters”
- “Transposed FIR Filter”
- “Systolic FIR Filter”
- “Symmetric Systolic FIR Filter”
- “Rounding”
- “Performance”
- “Conclusion”

## Overview

There are many filtering techniques available to signal processing engineers. A common filter implementation for high-performance applications is the fully parallel FIR filter. Implementing this structure in the Virtex-II architecture uses the embedded multipliers and slice based arithmetic logic. The Virtex-4 DSP48 slice introduces higher performance multiplication and arithmetic capabilities specifically designed to enhance the use of parallel FIR filters in FPGA-based DSP.

## Parallel FIR Filters

A wide variety of filter architectures are available to FPGA designers due to the flexible nature of FPGAs. The type of architecture chosen is typically determined by the amount of processing required in the available number of clock cycles. The two most important factors are:

- Sample Rate ( $F_s$ )
- Number of Coefficients ( $N$ )

In [Figure 4-1](#), as the sample rate and the number of coefficients increase, the architecture selected for a desired FIR filter becomes a more parallel structure involving more multiply

and add elements. Chapter 3, “MACC FIR Filters” addresses the details of the sequential processing FIR filters, including the single and dual MAC FIR filter. This chapter investigates the other extreme of the fully parallel FIR filter as required to filter the fastest data streams.

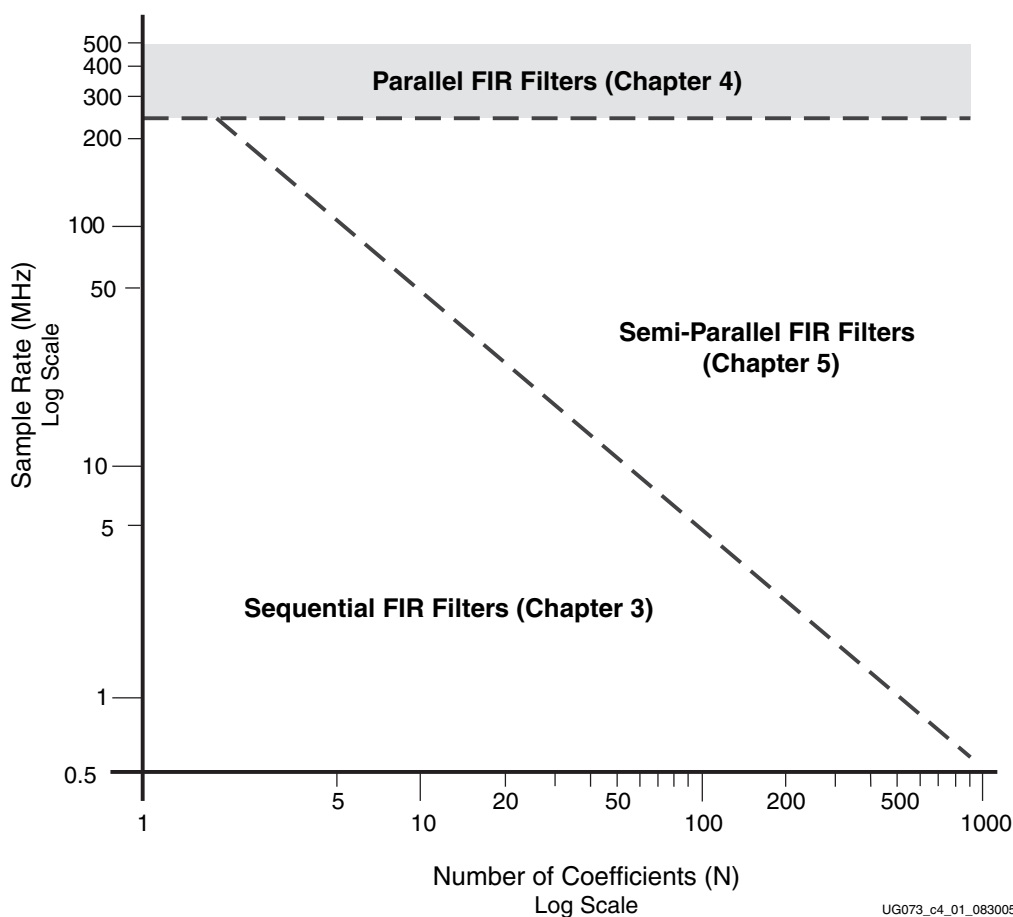


Figure 4-1: Selecting Filter Architectures

UG073\_c4\_01\_083005

The basic parallel architecture, shown in [Figure 4-2](#), is referred to as the Direct Form Type 1.

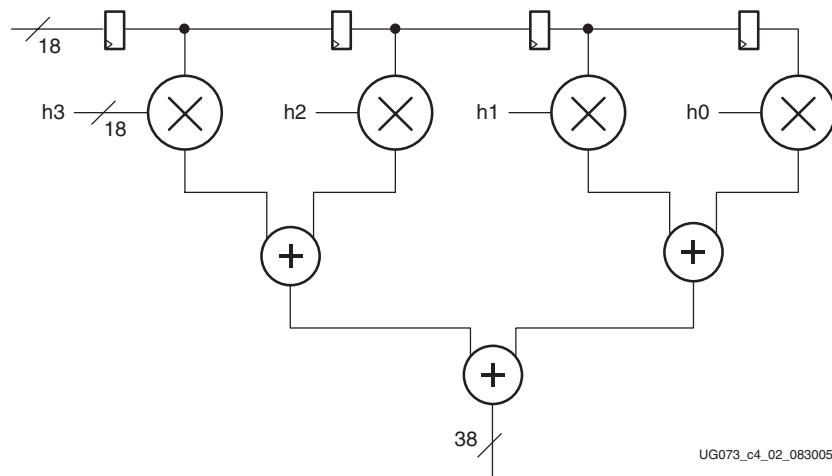


Figure 4-2: Direct Form Type 1 FIR Filter

This structure implements the general FIR filter equation of a summation of products as defined in [Equation 4-1](#).

$$y_n = \sum_{i=0}^{N-1} x_{n-i} h_i \quad \text{Equation 4-1}$$

In [Equation 4-1](#), a set of N coefficients is multiplied by N respective data samples. The results are summed together to form an individual result. The values of the coefficients determine the characteristics of the filter (e.g., a low-pass filter).

The history of data is stored in the individual registers chained together across the top of the architecture. Each clock cycle yields a new complete result, and all multiplication and arithmetic required occurs simultaneously. In sequential FIR filter architectures, the data buffer is created using Virtex-4 dedicated block RAMs or distributed RAMs. This demonstrates a trend; as algorithms become faster, the memory requirement is reduced. However, the memory bandwidth increases dramatically because all N coefficients must be processed at the same time.

The performance of the Parallel FIR filter is calculated in [Equation 4-2](#).

$$\text{Maximum Input Sample Rate} = \text{Clock Speed} \quad \text{Equation 4-2}$$

The bit growth through the filter is the same for all FIR filters and is explained in the section [“Bit Growth” in Chapter 3](#).

## Transposed FIR Filter

The DSP48 arithmetic units are designed to be easily and efficiently chained together using dedicated routing between slices. The Direct Form Type I uses an adder tree structure. This makes it difficult to chain the blocks together. The Transposed FIR filter structure (Figure 4-3) is more optimal for use with the DSP48 Slice.

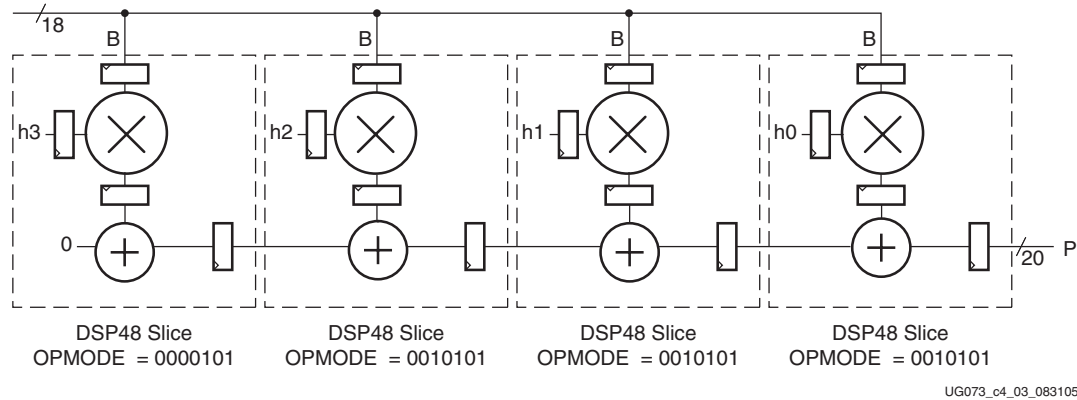


Figure 4-3: Transposed FIR Filter

The input data is broadcast across all the multipliers simultaneously, and the coefficients are ordered from right to left with the first coefficient,  $h_0$ , on the right. These results are fed into the pipelined adder chain acting as a data buffer to store previously calculated inner products in the adder chain. The rearranged structure yields identical results to the Direct Form structure but gains the use of an adder chain. This different structure is easily mapped to the DSP48 slice without additional external logic. If more coefficients are required, then more DSP48 slices must be added to the chain.

The configuration of the DSP48 slice for each segment of the Transposed FIR filter is shown in Figure 4-4. Apart from the very first segment, all processing elements must be configured as shown in Figure 4-4. OPMODE is set to multiply mode with the adder, combining the results from the multiplier and from the previous DSP48 slice through the dedicated cascade input (PCIN). OPMODE is set to binary 0010101.

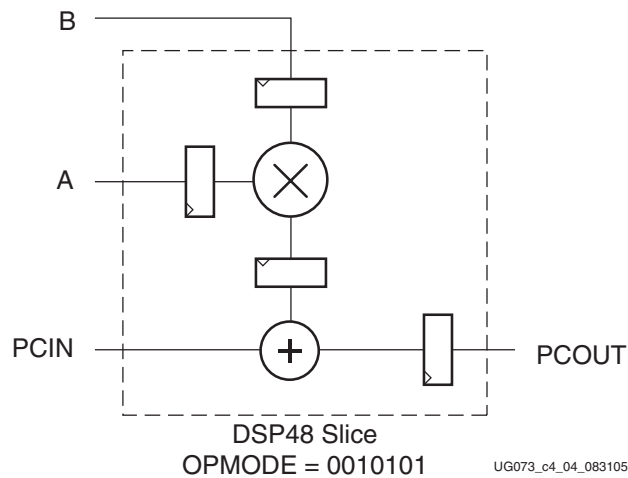


Figure 4-4: Transpose Multiply-Add Processing Element

## Advantages and Disadvantages

The advantages to using the Transposed FIR filter are:

- **Low Latency:** The maximum latency never exceeds the pipelining time through the slice containing the first coefficient. Typically, this is three clock cycles between the data input and the result appearing.
- **Efficient Mapping to the DSP48 Slice:** Mapping is enabled by the adder chain structure of the Transposed FIR filter. This extendable structure supports both large and small FIR filters.
- **No External Logic:** No external FPGA logic is required, enabling the highest possible performance to be achieved.

The disadvantage to using the Transposed FIR filter is:

- **Limited performance:** Performance might be limited by a high fanout input signal if there are a large number of taps.

## Resource Utilization

An  $N$  coefficient filter uses  $N$  DSP48 slices. A design cannot use symmetry to reduce the number of DSP48 slices when using the Transposed FIR filter structure.

## Systolic FIR Filter

The systolic FIR filter is considered an optimal solution for parallel filter architectures. The systolic FIR filter also uses adder chains to fully utilize the DSP48 slice architecture (Figure 4-5).

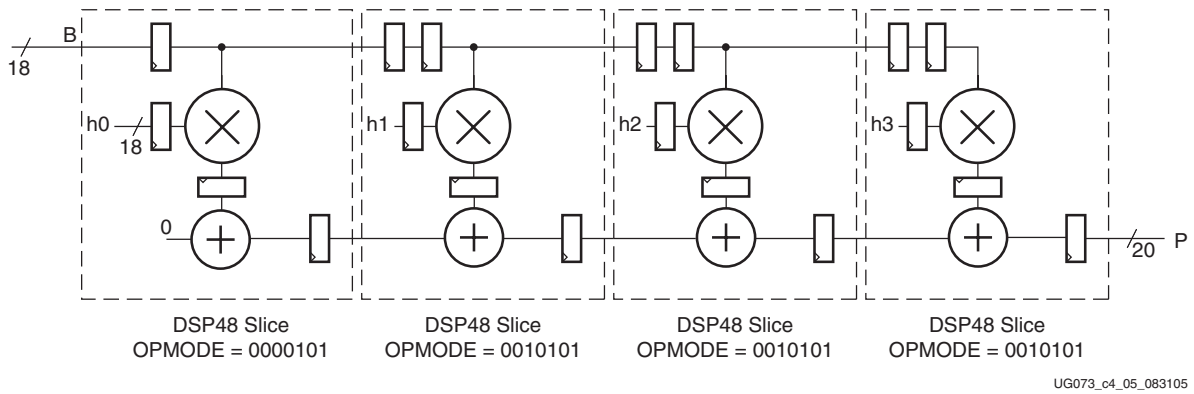


Figure 4-5: Systolic FIR Filter

The input data is fed into a cascade of registers acting as a data buffer. Each register delivers a sample to a multiplier where it is multiplied by the respective coefficient. In contrast to the Transposed FIR filter, the coefficients are aligned from left to right with the first coefficients on the left side of the structure. The adder chain stores the gradually combined inner products to form the final result. As with the Transposed FIR filter, no external logic is required to support the filter and the structure is extendable to support any number of coefficients.

The configuration of the DSP48 slice for each segment of the Systolic FIR filter is shown in Figure 4-6. Apart from the very first segment, all processing elements are to be configured as shown in Figure 4-6. OPMODE is set to multiply mode with the adder combining the results from the multiplier and from the previous DSP48 slice through the dedicated cascade input (PCIN). OPMODE is set to binary 0010101. The dedicated cascade input (BCIN) and dedicated cascade output (BCOUT) are used to create the necessary input data buffer cascade.

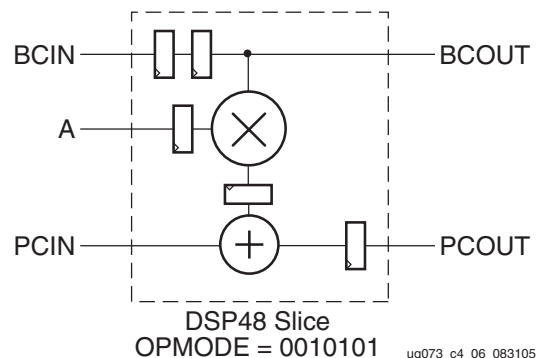


Figure 4-6: Systolic Multiply-Add Processing Element



## Advantages and Disadvantages

The advantages to using the Systolic FIR filter are:

- **Highest Performance:** Maximum performance can be achieved with this structure because there is no high fanout input signal. Larger filters can be routing-limited if the number of coefficients exceeds the number of DSP slices in a column on a device.
- **Efficient Mapping to the DSP48 Slice:** Mapping is enabled by the adder chain structure of the Systolic FIR Filter. This extendable structure supports large and small FIR filters.
- **No External Logic:** No external FPGA logic is required, enabling the highest possible performance.

The disadvantage to using the Systolic FIR filter is:

- **Higher Latency:** The latency of the filter is a function of how many coefficients are in the filter. The larger the filter, the higher the latency.

## Resource Utilization

An N coefficient filter uses N DSP48 slices.

## Symmetric Systolic FIR Filter

In Chapter 3, “MACC FIR Filters,” symmetry was examined, and an implementation was illustrated to exploit this symmetric nature of the coefficients. Exploiting symmetry is extremely powerful in Parallel FIR filters because it halves the required number of multipliers, which is advantageous due to the finite number of DSP48 slices. Equation 4-3 demonstrates how the data is pre-added before being multiplied by the single coefficient.

$$(X_0 \times C_0) + (X_n \times C_n) \dots \searrow (X_0 + X_n) \times C_0 \quad (\text{if } C_0 = C_n) \quad \text{Equation 4-3}$$

Figure 4-7 shows the implementation of this type of Systolic FIR Filter structure.

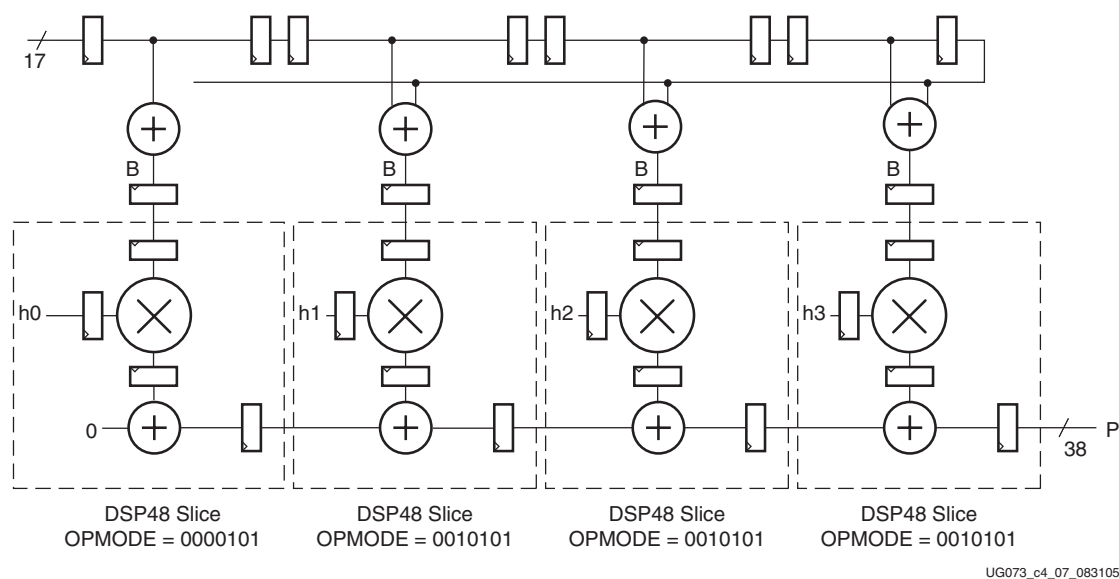


Figure 4-7: Symmetric Systolic FIR

In this structure, DSP48 slices have been traded off for device logic. From a performance viewpoint, to achieve the full speed of the DSP48 slice, the logic-slice-based 18-bit adder has to run at the same speed. To achieve this, register duplication can be performed on the output from the last tap that feeds all the other multipliers.

To save on logic area, the two register delay in the input buffer time series is implemented as an SRL16E and a register output. A further benefit of the symmetric implementation is the reduction in latency, due to the adder chain being half the length.

Figure 4-8 shows the configuration of the DSP48 slice for each segment of the Symmetric Systolic FIR filter. Apart from the very first segment, all processing elements are to be configured as in Figure 4-8. OPMODE is set to multiply mode, with the adder combining results from the multiplier and from the previous DSP48 slice via the dedicated cascade input (PCIN). OPMODE is set to binary 0010101.

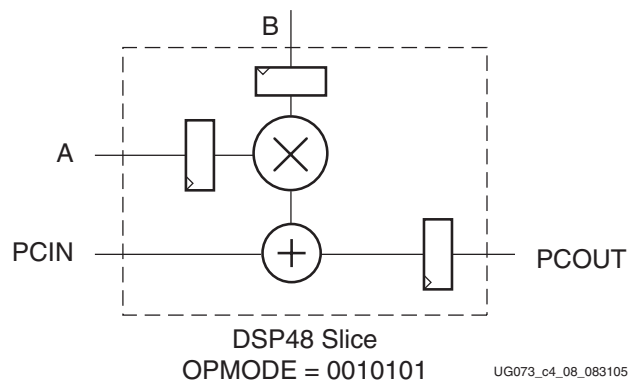


Figure 4-8: Symmetric Systolic Multiply-Add (MADD) Processing Element

## Resource Utilization

An  $N$  symmetric coefficient filter uses  $N$  DSP48 slices. The slice count for the pre-adder and input buffer time series is a factor of the input bit width ( $n$ ) and  $N$ . The equation for the size in slices is:

$$((n+1) * (N/2)) + (n/2) \quad \text{Equation 4-4}$$

For the example illustrated in Figure 4-7, the size is  $(17+1) * 8/2 + 17/2 = 81$  slices.

## Rounding

The number of bits on the output of the filter is much larger than the input and must be reduced to a manageable width. The output can be truncated by simply selecting the MSBs required from the filter. However, truncation introduces an undesirable DC data shift. Due to the nature of two's complement numbers, negative numbers become more negative and positive numbers also become more negative. The DC shift can be improved with the use of symmetric rounding, where positive numbers are rounded up and negative numbers are rounded down.

The rounding capability in the DSP48 slice maintains performance and minimizes the use of the FPGA fabric. This is implemented in the DSP48 slice using the C input port and the Carry In port. Rounding is achieved by:

For positive numbers: Binary Data Value + 0.10000... and then truncate

For negative numbers: Binary Data Value + 0.01111... and then truncate

The actual implementation always adds 0.0111... to the data value through the C port input as in the negative case, and then it adds the extra carry in required to adjust for positive numbers. Table 4-1 illustrates some examples of symmetric rounding.

Table 4-1: Symmetric Rounding Examples

Decimal Value	Binary Value	Add Round	Truncate: Finish	Rounded Value
2.4375	0010.0111	0010.1111	0010	2
2.5	0010.1000	0011.0000	0011	3
2.5625	0010.1001	0011.0001	0011	3
-2.4375	1101.1001	1110.0000	1110	-2
-2.5	1101.1000	1101.1111	1101	-3
-2.5625	1101.0111	1101.1110	1101	-3

For both the Transposed and Systolic Parallel FIR filters, the C input is used at the beginning of the adder chain to drive the carry value into the accumulated result. The final segment uses the MSB of the PCIN as the carry-in value to determine if the accumulated product is positive or negative. CARRYINSEL is used to select the appropriate carry-in value. If positive, the carry-in value is used, and if negative, the result is kept the same (see Figure 4-9).

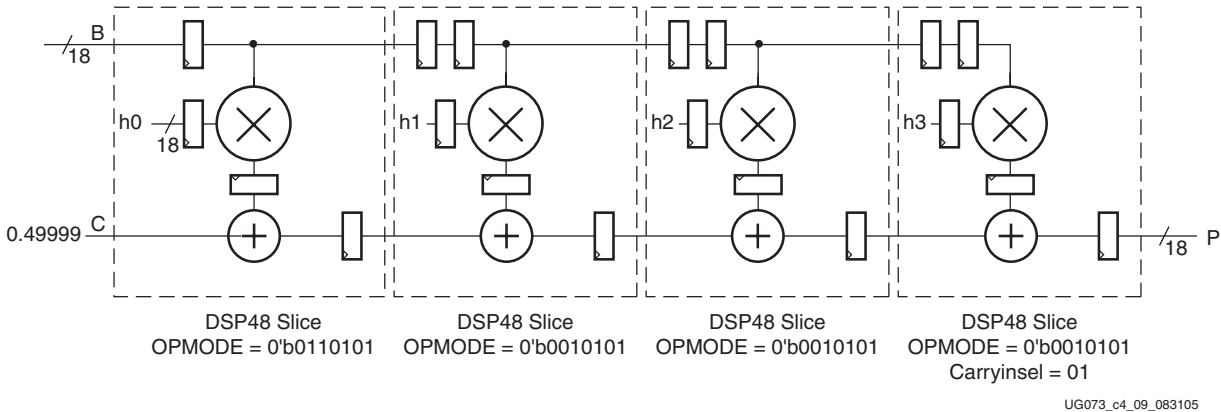
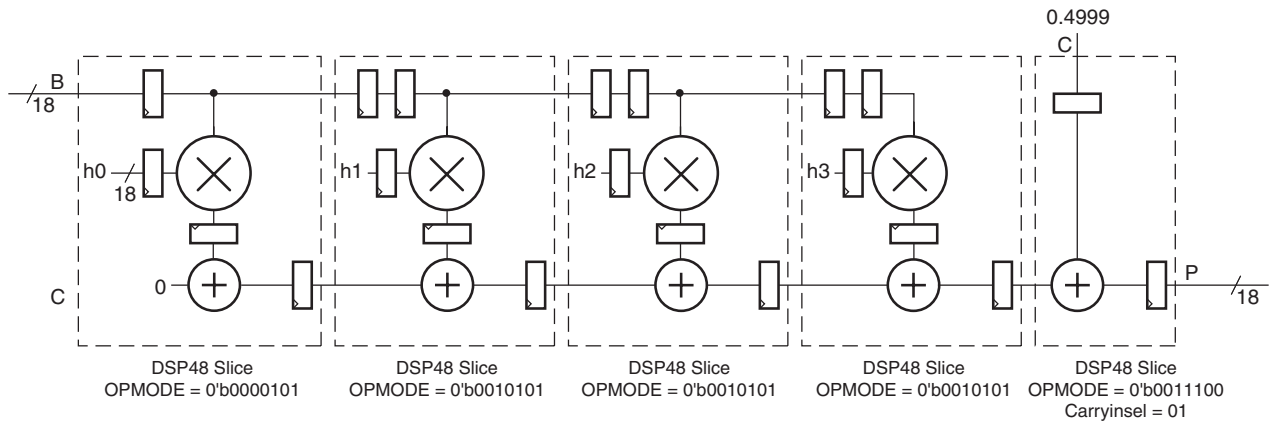


Figure 4-9: Systolic FIR Filter with Rounding

The one problem with the rounding solution occurs when the final accumulated inner product input to the final DSP48 slice is very close to zero. If the value is positive and the final inner product makes the result negative (leading to a rounding down), then an incorrect result occurs because the rounding function assumes a positive number instead of a negative. The last coefficient in typical FIR filters is very small, so this situation rarely occurs. However, if absolute certainty is required, an extra DSP48 slice can perform the rounding function (see Figure 4-10). A Transposed FIR filter can have exactly the same problem as the Systolic FIR filter.



UG073\_c4\_10\_083105

Figure 4-10: Systolic FIR Filter with Separate Rounding Function

## Performance

When examining the performance of a Virtex-4 Parallel FIR filter, a Virtex-II Pro design is a valuable reference. Table 4-2 illustrates the ability of the Virtex-4 DSP48 slice to greatly reduce logic fabric resources requirements while improving the speed of the design and reducing the power utilization of the filter.

Table 4-2: Performance Analysis

Filter Type	Device Family	Size	Performance	Power (Watts)
18 x 18 Parallel Transposed FIR Filter (51 Tap Symmetric)	Virtex-II Pro FPGA	1860 Slices 26 Embedded Multipliers	300-MHz Clock Speed 300 MS/S	TBD
18 x 18 Parallel Systolic FIR Filter (51 Tap Symmetric)	Virtex-II Pro FPGA	2958 Slices 26 Embedded Multipliers	300-MHz Clock Speed 300 MS/S	TBD
18 x 18 Parallel Transposed FIR Filter (51 Tap Symmetric)	Virtex-4 FPGA	0 Slices 51 DSP48 Slices	400-MHz Clock Speed 400 MS/S	TBD
17 x 18 Systolic FIR Filter (51 Tap Non-Symmetric)	Virtex-4 FPGA	0 Slices 51 DSP48 Slices	450-MHz Clock Speed 450 MS/S	TBD
17 x 18 Systolic FIR Filter (51 Tap Symmetric)	Virtex-4 FPGA	477 Slices 26 DSP48 Slices	400-MHz Clock Speed 400 MS/S	TBD

## Reference Design File

The reference design files associated with this chapter, ug073\_c04.zip, can be found at:  
[http://www.xilinx.com/xlnx/xweb/xil\\_publications\\_display.jsp?category=User+Guides/FPGA+Device+Families/Virtex-4&show=ug073.pdf](http://www.xilinx.com/xlnx/xweb/xil_publications_display.jsp?category=User+Guides/FPGA+Device+Families/Virtex-4&show=ug073.pdf)

## Conclusion

Parallel FIR filters are commonly used in high-performance DSP applications. With the introduction of the Virtex-4 DSP48 slice, DSPs can be achieved in a smaller area, thereby producing higher performance with less power penalty.

Designers have tremendous flexibility in determining the desired implementation. They also have the ability to change the implementation parameters. The ability to “tune” a filter in an existing system or to have multiple filter settings is a distinct advantage. By making the necessary coefficient changes in the synthesizable HDL code, the reconfigurable nature of the FPGA is fully exploited. The coefficients can be either hardwired to the A input of the DSP48 slices or stored in small memories and selected to change the filter characteristics. The HDL and System Generator for DSP reference designs are easily modified to achieve specific requirements.



## Are These Guys Dense, or What? Newest Class of FPGAs Makes Dense Cool

### Are These Guys Dense, or What?

Newest Class of FPGAs Makes Dense Cool

Context can drastically impact the meaning of a simple word. If you're walking down the street minding your own business, you might find yourself feeling more than a little bit offended if someone calls you dense. You may even experience a brief but painful flashback to that dreaded walk through the gauntlet of cool kids lining the halls in school, hearing any number of rude, if inaccurate, comments (after all, who's calling who dense?) thrown with casual abandon in your direction. Now, change the circumstance. You're at a tradeshow, heading back to a demo station in your company's booth. You find yourself facing another gauntlet of sorts, but this time it's lined with people wanting to see your latest product and admiring its components, including, by the way, some "wicked dense FPGAs." Suddenly, dense is cool.

This new class of FPGAs has just two members: Stratix II from Altera, and Virtex-4 from Xilinx. Although they both carry on their family names, following Stratix and Virtex-2 Pro, respectively, these 90nm device families deliver much more than standard generational improvements over their 130nm forefathers.

As expected, both device families offer higher density, lower cost per gate, and better performance. But before the vendors could dazzle us with all the trappings of their new offerings, they had to face the most daunting (and perhaps most anticipated) hurdle in making a move to 90nm: power. In an FPGA, power is broken down into two components: static and dynamic. It used to be that dynamic power (i.e., the power it takes to perform functions when the chip is running) was the big eater. But every time process geometry gets smaller, gates get thinner. Thinner gates leak more current (upping the static power consumption). There was a concern that if nothing was done to stop the trend, leakage current could all but take over as the power hog and potentially threaten the long-term viability of high-density FPGAs.

Both vendors successfully overcame the power issue, albeit in completely different ways. Xilinx tackled static power, using a new leakage-reducing “triple-oxide” technology where transistors can have one of three different oxide widths, depending on the speed demands they’re facing. Altera had started their power reduction strategy at 130nm with an all-copper process on 300mm wafers. At 90nm, they incorporated low-k dielectric material and actually improved power dissipation. Given the different approaches that Altera and Xilinx have taken to tackle power at 90nm, it will be interesting to see how each continues on the next process node.

Process improvements may have addressed power, but the next challenge was performance. “On its own, the move from 130nm to 90nm delivered a performance improvement of only 15-30% over our Virtex-2 Pro device family, so we needed to make fundamental changes to the architecture,” said Per Holmberg, Senior Director of Virtex Marketing at Xilinx. Xilinx leveraged the flexibility of flip-chip technology to create their Advanced Silicon Modular Block (ASMBL) architecture, which enables them to deliver three domain-optimized platforms that offer designers varying mixtures of capabilities at every price point. The base platform (called LX) is fairly generic, while the other two are intended for specific classes of design. SX focuses on DSP, and FX is targeted at embedded processing and high-speed serial I/O.

The architecture revolution was happening in parallel at Altera, where they created a new logic structure called Adaptive Logic Module (ALM) for Stratix II. ALMs can be flexibly repartitioned into various configurations for input sharing, enabling super efficiency for narrow logic elements and high performance for wide ones. According to Altera, compared to their previous LUT4-based architecture, the ALM delivers more logic capacity in less space, better performance, and much higher adaptability for applications.

Both device families have shiny new architectures designed to optimize performance, and both deliver significantly more hard IP, features, capabilities, and flexibility than their predecessors. Although they approached the challenge in very different ways, both FPGA vendors have ended the 90nm challenge at almost exactly the same point. It’s as though Altera started at “A” and Xilinx started at, well, “X” and they both ended at “M.” Density, performance, and power claims aside, it’s likely that the feature mix and toolsets will most often be key deciding factors in selecting a family.



## **If You Build it, They Will Come. Right?**

They're cool. They're capable. They're so flexible they could have an act in Cirque du Soleil. Now, where do they fit? The first impression you might have upon hearing all these new features and benefits is that these device families open the doors to huge new markets. Despite significant cost gains made by this generation of devices, cost is still guarding the door. It turns out that the real door openers for the FPGA vendors are the low-cost FPGA families – Xilinx's Spartan-3 and Altera's Cyclone II. These FPGAs are taking high-volume markets like consumer and automotive by storm with densities that approach first-generation Virtex and Stratix offerings. But that's another story.

So, why would you pay more for these high-end devices? Features, baby, and lots of 'em. You get more logic, a lot more DSP blocks, high-speed serial I/O, tons more memory, more embedded processor capabilities, and just more I/O in general (you can never be too rich or have too much I/O).

Traditionally, high-end FPGAs have had two primary footholds. The first is design teams using FPGAs for prototyping. For these customers, the 90nm devices get them more equivalent gates and more speed, resulting in incremental capability improvements. The second is designers using FPGAs in production. These guys are the real envelope pushers. They need the most density and performance that you can throw at them, but they also need the flexibility and rapid time-to-market of a programmable device. They don't mind the comparatively high unit cost of these devices. That's a tradeoff they're more than willing to make. These are the high-density "heartland" FPGA customers, coming mostly from markets like communications and networking.

Both Xilinx and Altera say that the high-density devices have helped deepen their reach into their existing customer base, moving them into new parts of the design. For example, a customer could be using an ASIC or ASSP for the part of the design that they're sure of, and an FPGA for the part that's still in flux. The size and flexibility of the Virtex-4 and Stratix II device families make them a viable choice for this type of design, creating more options for what can be put on the device and allowing designers to bring more of their system onto the FPGA.

A prime example of the trend toward increased FPGA integration is in the DSP realm, where designers who have not worked with FPGAs in the past are

seeing new possibilities for both flexibility and performance. The two keys to DSP success are the ability to parallelize arithmetic functions, particularly multiplication, and the tool flow to simplify the hardware design task for DSP designers. Both FPGA vendors offer solutions specific to DSP users, with dedicated hardware blocks that can dramatically accelerate DSP performance and specialized design tools for DSP design. Each FPGA vendor offers devices with hundreds of DSP blocks that include 18x18 multipliers, offering the potential for dramatic DSP acceleration when compared to traditional DSP processors.

Another trend is in embedded design. Xilinx's Virtex-4 FX product includes an embedded PowerPC and Ethernet MAC, as well as their RocketIO multi-gigabit serial transceivers. In addition to the hard-core PowerPC, Xilinx has two soft-core processors (the 8-bit PicoBlaze and the 32-bit MicroBlaze). Altera's Stratix II emphasizes flexibility with their established Nios and Nios II 32-bit soft cores combined with the versatility of their SOPC builder and its rich selection of embedded peripherals and IP. As these device families continue to strengthen their embedded capabilities, they become increasingly viable for the embedded design community at large.

Now back to those pesky, well-guarded new market doors. Although the high-density devices don't open them on their own, they have carved an important niche for themselves in higher-volume applications when paired with some innovative cost-reduction techniques. Altera's innovation is a structured ASIC solution called HardCopy. If you're starting with a Stratix II FPGA, you can move it into HardCopy to reduce your cost per device and get the benefits of a structured ASIC (faster, lower power, etc.). "By combining Stratix II with HardCopy, customers can get to volume production much faster, and react to changes with a lower NRE," said Paul Ekas, Senior Marketing Manager for high-density FPGAs at Altera.

Xilinx's answer to the cost conundrum is called EasyPath. EasyPath keeps cost down by design-specific testing of Virtex-4 devices. Essentially, you keep the convenience, power, and features of the FPGA but reduce your cost by giving up some of the reprogramming flexibility.

## **Addressing the Design Challenge**

With all of the functionality and flexibility of these high-density devices comes a new level of design challenge, and according to the EDA vendors, it's not just the size of the designs that matters. "By adding more system functionality, designers have a reprogrammable platform that they can put

more memory into, more fixed IP blocks, processors, and more," said Juergen Jaeger, Director of Channel Marketing at Mentor Graphics. "That brings up challenges that have more to do with assembling the designs."

As ASIC-grade challenges like team design, design cycle predictability, and verification creep into the FPGA world, the EDA vendors can help with the transition. "Design size, complexity, clock speeds, and IP requirements are making these FPGA devices very similar to ASIC designs," said Gal Hasson, Director of Marketing, RTL Synthesis, at Synopsys. "Putting ASIC designs on FPGAs requires ASIC-grade tools."

In addition, the EDA vendors have their eye on designers migrating to high-density FPGAs for their new capabilities, particularly in DSP. "DSP applications are really starting to gain momentum," said Jeff Garrison, Director of Marketing at Synplicity. "It's looking like the primary inroad for the big devices, and we've added specific tools and functionality tailored to this area."

In other words, there's no shortage of opportunities when it comes to design challenges for high-density FPGAs, and the EDA vendors are working hard to stay ahead of the changing landscape to help customers get the most of every new feature.

So, dense is cool. And as the vendors continue to pile on features, performance enhancements, and cost-saving innovations, it's only going to get cooler.

*Amy Malagamba, FPGA and Programmable Logic Journal*

*June 14, 2005*



## Redefining Structured ASIC

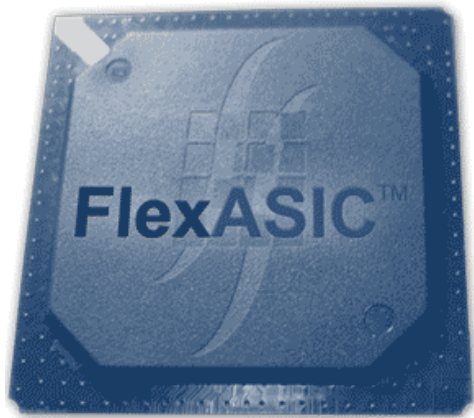
### eASIC's Better Idea

In most markets, there exist a set of de-facto rules. Sport-utility vehicles have bad fuel economy. Economy cars have limited cargo space. FPGAs use too much power. ASICs have staggering non-recurring engineering (NRE) costs. Generally, the players play by those rules, and the consumer enters every buying decision with a pre-defined understanding of the tradeoffs those rules imply and which basic option favors their situation. The final choice is then decided by a comparison of the less-critical factors that differentiate the products in each area. Once a design team has decided to go with a zero-NRE solution, for example, they typically find themselves comparing various FPGA offerings to find the one that best fits their needs.

Occasionally, however, someone breaks the rules. What if you could get an ASIC with near zero NRE? Would you still be locked into an FPGA solution? What if a structured ASIC offered you some degree of reprogrammability, or if you could vary the design on a small lot basis? Your decision approach, and even your entire design, might shift dramatically.

eASIC has announced its new FlexASIC family of structured ASIC devices that break the established rules. FlexASIC offers density, performance, and power consumption that come close to cell-based ASIC, with flexibility, NRE, and risk avoidance much closer to the FPGA end of the spectrum. This is accomplished by using an innovative architecture that combines FPGA-like look up table (LUT) cells connected by metal routing that is customized by a single via layer. This single via layer can be e-beam programmed for small production runs, or mask programmed for higher volumes.

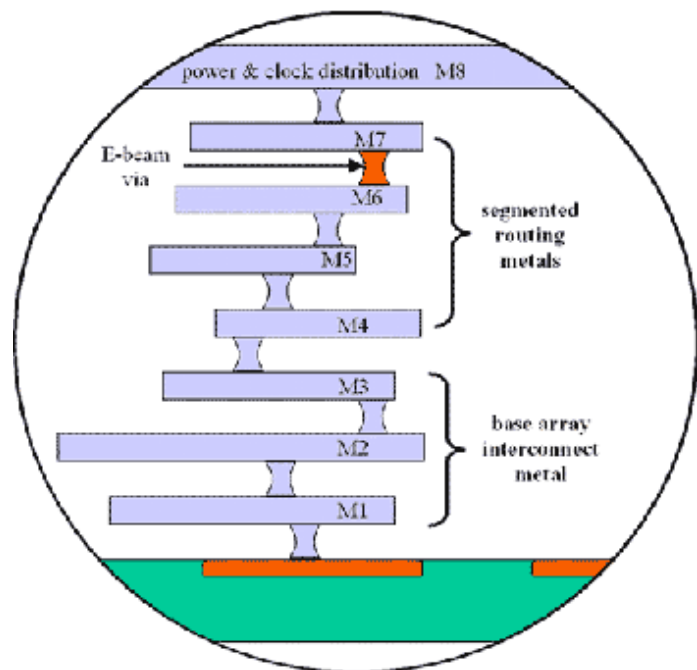
An additional advantage of the e-beam programming approach is the ability to easily segment wafers, putting multiple design variants on a single wafer. This eliminates minimum volume requirements and ultimately lowers unit



costs as a single production lot can be shared across many designs. While e-beam has suffered a somewhat dubious reputation in the past for impractical levels of performance, eASIC points out that their use of e-beam is comparatively fast because only a single via layer is being customized. When it comes time to go to production, no requalification is necessary because the mask-customized version will be identical to the e-beam version.

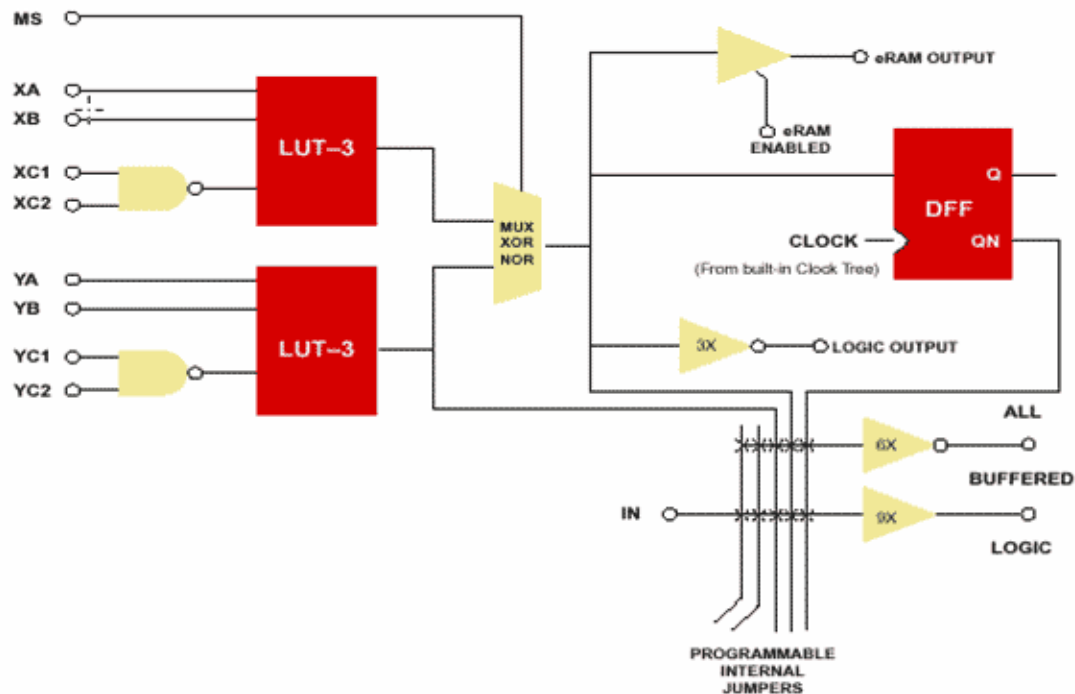
From a security perspective, FlexASIC also offers the best of both worlds. The via-based routing makes reverse-engineering of the routing fabric extremely difficult, and because the logic is bitstream programmed, the device itself does not contain the entire design until runtime. Potential thieves would have to recreate both elements to have a viable copy of the design.

Architecturally, FlexASIC is perhaps closest to Altera's HardCopy in that it uses metal-to-metal connections between LUT structures, but FlexASIC is actually closer to an FPGA in that the LUTs are bitstream programmed where HardCopy's logic is all mask programmed. The key to the unique value of FlexASIC is the single-layer programming. Because programming is accomplished solely via vias, and because those vias can even be e-beam programmed, eASIC is able to offer a near-zero NRE customization with very fast turnaround. From the system design perspective, FlexASIC comes very close to FPGAs in turnaround, NRE, and risk, where it comes out virtually identical to structured ASIC in performance, price, and power consumption.





From a density perspective, FlexASIC most closely overlaps high-end FPGA families with a density range from 250K to 3M "ASIC gates". Since we're dealing with a LUT-based architecture, however, density requires a bit more explanation. The family contains from 16K to 192K "eCells". Each eCell is structurally similar to a Xilinx slice or an Altera ALM. The eCells contain two almost-four-input LUTs (meaning three-input LUTs with one input of the LUT occupied by a two-input NAND.) If we just compare LUT counts (and your LUT mileage may vary considerably depending on factors like synthesis and layout efficiency, the topology of your design, and the phase of the moon) the largest FlexASIC has something like double the LUT count of today's largest 90nm FPGAs.



FlexASIC also includes from zero to about 2.8 megabits of block RAM, from one to eight PLLs, and from 118 to 744 user I/O pins. As in many FPGAs, the eCells can also be used as distributed RAM, adding to the memory capacity and performance when needed. The bitstream programmability of eCells also permits them to be used to create a virtual PLD within your design. Using this approach, parts of the design can be made to be field-customizable by reprogramming a PLD section.

## FlexASIC Structured ASIC Device Features

Feature	ASIC Gates	eCells	Max. Distributed RAM		Block RAM		PLLs	Max. User I/O***
			eRAM Blocks	eRAM Bits	bRAM Blocks	bRAM Bits		
<b>FA250L</b>	<b>250K</b>	<b>16K</b>	<b>64</b>	<b>256K</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>118</b>
<b>FA250</b>	<b>250K</b>	<b>16K</b>	<b>64</b>	<b>256K</b>	<b>6</b>	<b>192K</b>	<b>2</b>	<b>168</b>
<b>FA600</b>	<b>600K</b>	<b>40K</b>	<b>160</b>	<b>640K</b>	<b>12</b>	<b>384K</b>	<b>4</b>	<b>364</b>
<b>FA1000</b>	<b>1000K</b>	<b>70K</b>	<b>280</b>	<b>1120K</b>	<b>23</b>	<b>736K</b>	<b>6</b>	<b>488</b>
<b>FA2000</b>	<b>1800K</b>	<b>120K</b>	<b>480</b>	<b>1920K</b>	<b>32</b>	<b>1024K</b>	<b>8</b>	<b>656</b>
<b>FA3000</b>	<b>3000K</b>	<b>192K</b>	<b>768</b>	<b>3072K</b>	<b>88</b>	<b>2816K</b>	<b>8</b>	<b>744</b>

When it comes to verification, FlexASIC is most like FPGAs and structured ASICs. Tedious tasks like layout verification are all completed ahead of time as part of the base design so you don't have to worry about them, the risk they carry, or the expensive tools they require. FlexASIC has full BIST for logic and memories, so testing via conventional ASIC methods is possible.

FlexASIC is fabricated on 130nm technology, but eASIC claims that architectural advantages still give significant price, power, and performance advantages over 90nm FPGAs. While specific data is not yet available, eASIC claims that performance will be in the range of double that of FPGA, cost will be less than one-third, and total power will be more than an order of magnitude better. Given the metal-to-metal routing scheme, these claims seem reasonable enough.

On the tool side, the flow most closely resembles the structured ASIC flow. System-level design is completed using any conventional tools, and logic synthesis can be done either with Magma or Synopsys tools. With the Magma flow, physical optimization is also included. eASIC's proprietary tools are used for the highly-specialized and unique style of layout required for FlexASIC's single-via layer customization. The primary outputs of the design process are a via mask for layout customization and a bitstream for logic/LUT customization.

eASIC is partnering with Flextronics Semiconductor and Magma Design Automation, with Flextronics providing manufacturing services and Magma

offering design tool support. The FlexASIC devices will be offered by both eASIC and Flextronics semiconductor. Prototype silicon is slated for availability in Q3 2005 with production silicon in Q4. Several beta customers are in design and tape-out with the technology today.

As would be the case with any innovative, fledgling technology, FlexASIC lacks the robust infrastructure of tools, support and IP that might be typical of a well-established FPGA family. The benefits are compelling, however, and if the technology gets traction in design-ins, look for the infrastructure to mature rapidly. The new architecture puts considerable additional pressure on low-end ASIC, high-end FPGA, and the entire structured ASIC market. As with any time a rule is changed, watch for the various competitors to respond: structured ASIC vendors by working to reduce NRE and turnaround time requirements, FPGA vendors by leveraging hard-IP based features and infrastructure like DSP blocks, embedded processors, and the tool and IP ecosystem that surrounds them.

*Kevin Morris, FPGA and Programmable Logic Journal*

*May 24, 2005*





## LSI Logic's Leverage

RapidChip Heads to 90nm

What has over two million real ASIC gates, runs at over 200MHz with twenty levels of logic, burns about the power of a cell-based ASIC, and is economically feasible to deploy even in mid-to-small volume production? The answer is "Not an FPGA". While these specs may sound close to the marketing picture painted by programmable logic vendors, there is a vast gulf between the brochure and real-world performance in an average application. Structured and platform ASICs, however, can realistically reach these goals in your average application, and getting design teams to understand that fact is one of the biggest challenges faced by structured ASIC marketers in working with design teams jaded by years of spin-laden specsmanship from the FPGA industry.

LSI logic announced this week that their increasingly successful RapidChip platform/structured ASIC family is headed to 90nm. While the step to the next process node may come as no surprise, the implications merit serious consideration, particularly for high-end FPGA customers who are either pushing the limits of FPGA technology or going into volumes where FPGA unit costs are prohibitive. It also means that platform ASICs are taking another giant bite out of the cell-based market space. With their 90nm announcement, LSI rolled out densities up to 10 million "these-are-not-the-same-as-system" ASIC gates. This upward leap in density clearly blows any lingering theory that the company might be protecting its cell-based offering by throttling back on structured ASIC.

Structured and platform ASIC has now transcended the "will they catch on?" question and gone straight into "how big will they get?" At this juncture, the answer appears to be "really big." Structured ASIC now could join low-cost FPGA in the lucky category of devices with almost unbounded growth potential.

We can split the custom digital logic market into bins almost an unlimited number of ways, but consider the span of devices available from “simple” CPLDs through cell-based ASICs. If we limit ourselves to medium and higher volumes (let’s pick 4-digit numbers and up for convenience), we can then segment roughly by performance/density points. At the low end, we have essentially one bin that contains CPLDs and low-cost FPGAs. These devices all have small 2-digit or 1-digit price tags (US dollars) and can handle probably up to the range of 100K ASIC gates of complexity (or a little better, depending on your gate math). Low-cost FPGAs are cost effective in their density range from a volume of one up through millions of units. No analyst would come under suspicion for predicting huge growth potential for this market segment.

Moving up the density ladder one notch, we get to the problematic plateau of high-density FPGAs. High-density FPGAs face problems such as power, price, performance, and now platform ASIC that threaten to marginalize them into the role of “specialty” devices. Even today, to effectively cross into medium-volume effectiveness, high-density FPGAs depend on cost-reduction strategies such as Altera’s HardCopy (structured ASICs directly from FPGA) and Xilinx’s EasyPath (FPGAs cost-reduced by application-specific testing). Both of these strategies produce devices that are no longer fully reprogrammable FPGAs and that are density limited to the size of the original FPGA.

Carrying on to the next density level (the single digit millions of ASIC gates), we enter a domain that will probably become the exclusive property of structured and platform ASIC. With 90nm offerings moving this range up to 10-million ASIC gates with copious amounts of on-chip memory and robust IP libraries, there is no technology that threatens serious competition with these devices. FPGAs don’t match their density, performance, power consumption, or unit cost. Full-blown ASICs cost ten to a hundred times more to develop because of mask, NRE, tool, and design team costs.

Finally, in the rare air at the very top of the density/performance range, we have cell-based ASICs and custom silicon based on customer-owned tooling (COT) methodologies. With every process node, the design challenge for these devices becomes tougher, the mask and NRE costs rise higher, and the number of applications that require their unique performance or density capabilities becomes smaller. Already a good number of these designs are going into second-order applications where they are re-sold as ASSPs, FPGAs, platform ASICs, or other broadly useful devices that can take advantage of economy of scale to offset their massive development costs.

If this picture continues with no unexpected technology discontinuities, we get a scene with almost unbounded growth in structured/platform ASICs and low-cost FPGAs and marginal futures for higher-density FPGAs and ASICs. With each generation, the two more accessible technologies will press their density and performance numbers upward, stealing market share away from their less fortunate high-end brethren. LSI Logic is now proving that it has no compunction about pushing RapidChip straight toward the heart of their historical bread-and-butter product offerings. This means that at least one company sees enormous potential in platform ASIC.

LSI Logic is already on to the next step of the game. Many players will participate in the growing structured ASIC market, and everyone will have access to comparable levels of silicon technology. The axis most likely to determine success and failure in this space is application-appropriate IP. When a design team evaluates alternatives in structured ASIC, the deciding factor is most likely to be the IP collection that backs up each offering. LSI Logic has a long history of success at the IP- and application-focused marketing strategy. They come into platform ASIC with a wealth of experience helping their customers succeed by understanding the customer's design problems in advance and offering a combination of IP and services that accelerate and simplify the design process.

LSI is taking RapidChip into the communications, storage, consumer, and mil-aero markets where there are numerous mid-volume, capability-hungry applications waiting to take advantage of the low-NRE, high-performance combination offered by structured ASIC. In each area, LSI has a focused marketing effort, a comprehensive IP portfolio, partnerships with key industry players, and a growing list of blue-chip customers doing designs with RapidChip. Each market segment ideally demands slightly different mixes of hard-IP on their device, and one of the early criticisms of RapidChip was that almost every customer engagement created the need for a new "slice" (LSI's term for a particular platform's configuration of IP). With a couple of years of platform experience behind them, however, LSI now has a large selection of slices already designed, and the need to customize continues to dwindle.

One often overlooked advantage of the structured ASIC approach is the simplification of the design tool flow compared with cell-based ASIC and COT. In the case of RapidChip, the design process is supported by what LSI calls RapidWorx, which is a tool set that strongly resembles the design kits that have been used by FPGA designers for the past decade or so, right down to the Synplicity-based synthesis. Synplicity was the first to jump on the

structured ASIC wagon with both feet, and they stand to gain a commanding position if the market truly takes off. With the customers' tool expectations set much closer to the FPGA end of the dial, Synplicity had the infrastructure and experience to tackle the problem without much adjustment, and they got a big jump on the rest of the EDA industry in supporting structured-ASIC's needs.

From the customer perspective, there is no need to mix-and-match a point-tool based flow. Everything needed to design successfully is included in the design kit. Aftermarket tools are not ruled out, but they are most likely to show up in the front-end of the design flow in design creation and management, IP integration, and (we're still not sure exactly what it means) ESL. While the FPGA and structured ASIC boom means a mandatory adjustment for semiconductor companies, it carries strong implications for EDA as well.

Since these are "platform" devices, one would also expect a robust infrastructure for the development and deployment of embedded software and for the process of hardware/software co-design. To date, much of the focus from LSI seems to be on the hardware side of the equation, with the software side coming mainly as windfall from partnerships such as ARM and the supporting cast they bring with them. As more customers take advantage of the true embedded computing capabilities of structured ASIC platforms, however, watch for the software development component to push toward center stage.

Today, however, LSI's focus is on deepening the penetration with the current 130nm offering with capabilities like SerDes and increased memory offering, while driving toward their 90nm introduction in order to support emerging IP and interface standards not available at the 130nm process point. For designers of high-performance systems that surpass FPGA's capability but don't warrant ASIC's investment, progress can't come fast enough.

*Kevin Morris, FPGA and Programmable Logic Journal*

*July 5, 2005*