Imperial College
London

Lecture 15

Adders and DSP Blocks

Peter Cheung
Department of Electrical & Electronic Engineering
Imperial College London

URL: www.ee.imperial.ac.uk/pcheung/teaching/ee2_digital/
E-mail: p.cheung@imperial.ac.uk

Lecture Objectives

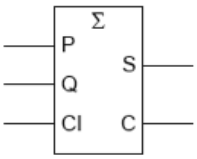
- Understand how to add both signed and unsigned numbers
- Appreciate how the delay of an adder circuit depends on the data values that are being added together
- Understand how adders are implemented on Cyclone V devices
- Understand DSP Blocks in Cyclone V and its flexible multipliers

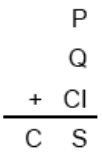
In this last lecture of the course, I will be looking at how adder circuits work. In particular, I will be examining the implementation of adders on the Cyclone V FPGAs. Furthermore, since Cyclone V also contain a large number of multipliers inside DSP blocks, I will brief explain how these could be instantiated.

Full Adder

- Output is a 2-bit number counting how many inputs are high
- Symmetric function of the inputs
- Self-dual: Invert all inputs => invert all outputs
  - If *k* inputs high initially then 3–*k* high when inverted
  - Inverting all bits of an *n*-bit number make $x \rightarrow 2^n - 1 - x$

Note: P ⊕ Q ⊕ CI = (P ⊕ Q) ⊕ CI = P ⊕ (Q ⊕ CI)





$$C = P \cdot Q + P \cdot CI + Q \cdot CI$$

$$S = P \oplus Q \oplus CI$$

P	Q	CI	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The building block for an n-bit adder is a one-bit full adder. This is essentially a component that adds THREE one-bit values together: P Q and CI. It produces two outputs: Sum S, and Carry C.

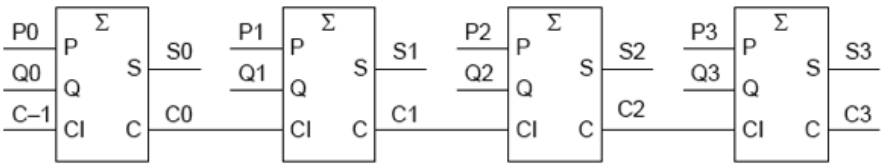
Note that all three inputs are symmetrical – they can be swapped without any change in the outputs.

Further, if you invert all inputs, the outputs are also all inverted. This is known as “self-dual”.

The Boolean equation for C and S are shown here. This is something that has been covered from the first year.

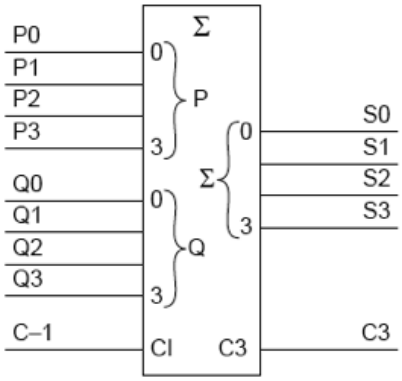
N-bit adder

- We can make an adder of arbitrary size by cascading full adder sections:



- The main reason for using 2's complement notation for signed numbers is that:

Signed and **unsigned** numbers can use **identical** circuitry for add or subtract

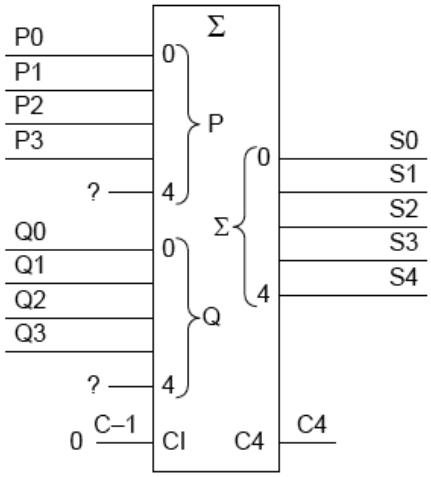


To build a 4-bit adder from 1-bit full adders, we can connect four of these serially as shown here. It is important to appreciate that this 4-bit adder works for both signed and unsigned input, provided that signed numbers are using 2's complement representation. In other words, if you interpret input as unsigned 4-bit numbers, then the adder produces unsigned 5-bit output (including the carry out signal).

If you interpret the input as 2's complement signed numbers, then the output is correct as a 4-bit SIGNED output. (In this case, you cannot use C3 as the 5th bit).

Adder Size Selection

- The number of bits needed in an adder is determined by the range of values that can be taken by its **output**
- If we add two 4-bit numbers, the answer can be in the range:
  - 0 to 30 for *unsigned* numbers
  - -16 to +14 for *signed* numbers
- In both cases we need a 5-bit adder to avoid any possibility of overflow
- We need to expand the input numbers to 5 bits. How do we do this ?



If you add two 4-bit numbers together, the inputs are unsigned, then the output will be in the range of 0 to 30. However, if you want to use the adder for signed number addition, the input range is -8 to +7, therefore the output range is -16 to +14. In both cases, we would need 5-bit adder to avoid any overflow.

For the unsigned case, you could use the carry out as the 5th output bit. For now, let us use only the sum output, and we need a 5-bit adder.

For signed addition, we cannot use the carry out as the 5th bit. We MUST use a 5-bit adder. So, we need expand the input numbers from 4-bti to 5-bit. What do we do with the MSB?

Expanding Binary Numbers

Unsigned numbers

- Expand an unsigned number by adding the appropriate number of 0's at the MSB end:

5	0101	00000101
13	1101	00001101

Signed numbers

- Expand a signed number by duplicating the MSB the appropriate number of times:

5	0101	00000101
–3	1101	11111101

- This is known as ***sign extension***

For unsigned numbers, exampling a 4-bit number to an 8-bit number simply requires adding 4 extra '0' to the MSB part of the number.

For signed numbers, we need to extend to the left four sign bits in order to preserve the signed of the number and maintain the correct value. This is known as "sign extension".

Shrinking Binary Numbers

Unsigned

- Can delete any number of bits from the MSB end so long as they are all 0's

Signed

- Can delete any number of bits from the MSB end so long as they are all the same as the MSB that remains

Both Signed and Unsigned

- Can truncate or round LSB end (but will generally introduce truncation or rounding errors)

PYKC 28 Nov 2017 E2.1 Digital Electronics Lecture 15 Slide 7

To shrink a binary number to smaller number of bits, it is easy for unsigned numbers – simply delete all leading '0's for MSB. The value of the unsigned number will not change.

For signed numbers, if the MSB is 0, you can delete all '0' from MSB down except the last '0'. If the MSB is '1', you can delete all '1's from MSB down except the last '1'. In that way, you preserve the correct sign of the number.

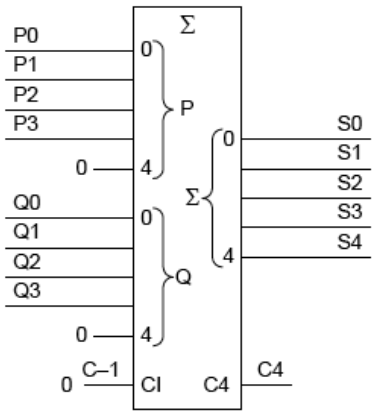
You can also shrink the number of bits by removing unwanted LSBs through truncation. Alternatively you can perform rounding.

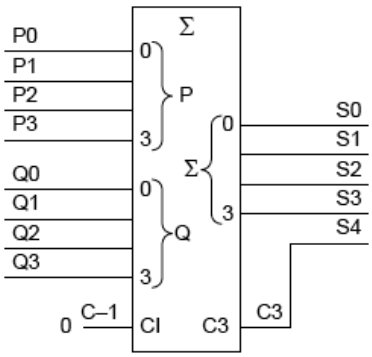
Truncation is easy – to reduce an 8-bit number to 4-bit, just remove the least significant 4-bit.

Rounding is harder, and there are various method of rounding that can be used. The simplest method in the case of 8-bit rounded to 4-bits is to add 8'b00001000 to the number, than truncate the bottom 4-bit. Basically, you add half of the LSB of the final number to the original number before chopping off the lower bits. This is the same as how we round with decimal numbers.

Adding Unsigned Numbers

- To avoid overflow, we use a 5-bit adder:
- The MSB stage is performing the addition: 0 + 0 + C3
- Therefore S4 is always equals C3 and C4 always equals 0





- We can use a 4-bit adder with C3 as an answer bit for unsigned adder

PYKC 28 Nov 2017 E2.1 Digital Electronics Lecture 15 Slide 8

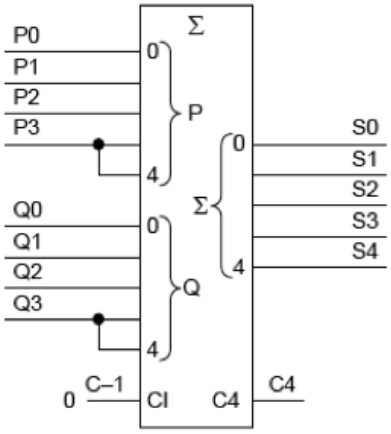
In order to avoid overflow when adding two 4-bit numbers together, we need to use a 5-bit adder. For unsigned add, we zero extend the input sto 5-bit and then use a 5-bit adder to produce S4:0 as shwon here.

Of course, we could have used the 4-bit adder and use the carry out C3 as S4.

Adding Signed Numbers

- To avoid overflow, we use a 5-bit adder:
- This is different from the unsigned case because P4 and Q4 are no longer constants. We cannot simplify this circuit by removing the MSB stage
- If P and Q have different signs then S4 will not equal C3

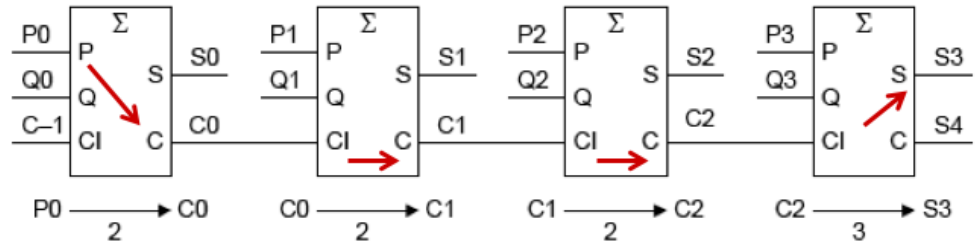
e.g. P=0000, Q=1111
Unsigned P+Q=01111, Signed P+Q=11111



- Some minor simplifications are possible:
  - If the C4 output is not required, the circuitry that generates it can be removed
  - S4 can be generated directly from P3, Q3 and C3 which reduces the circuitry needed for the last stage

To build a 4-bit signed adder WITHOUT overflow, we need to first extend the input to 5 bits with sign extension sa shown here. Then use a 5-bit adder circuit to produce a 5-bit signed result.

Adder Propagation Delay



- Delays within each stage (in gate delays):
  P, Q, CI ➔ S = 3 P, Q, CI ➔ C = 2
- Worst-case delay is:
  P0 ➔ C0 ➔ C1 ➔ C2 ➔ S3 = 3 × 2 + 3 = 9
- Note: We also have Q0 ➔ S3 = 9 and C–1 ➔ S3 = 9
- For an N-bit adder, the worst delay is (N–1) × 2 + 3 = 2N+1
- Example of worst case delay:
  - Initially: P3:0=0000, Q3:0=1111 => S4:0=01111
  - Change to: P3:0=0001, Q3:0=1111 => S4:0=10000

Let us consider the propagation delay through a 4-bit adder. The worst case path is from P0 or Q0 input, then pass through the carry chain to the MSB sum output.

Assuming the gate delay to C is 2 andto S is 3, then the worst case delay is 9.

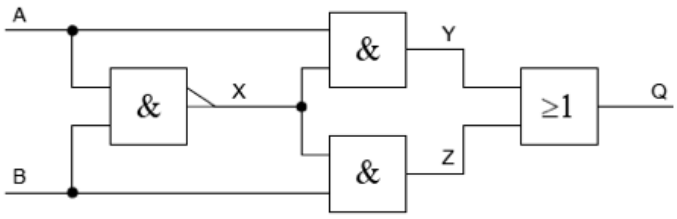
This is called a ripple carry adder because the carry signal has to propagate all the way from the LSB stage to the MSB stage.

An example scenario for the worst case propagation delay is shown here. If initially P=4'b0000 and Q=4'b1111, the S=5'b10000.

Now if P changes to 4'b0001, then this '1' in the LSB is propagated all the way to S4. The worst-case path is exercised.

Delays are Data-Dependent

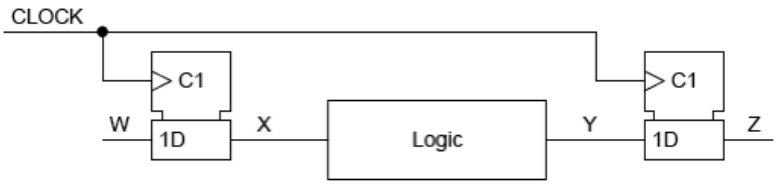
- To determine the delay of a circuit, we need to specify:
  1. The circuit
  2. The initial value of all the inputs
  3. Which of the inputs changes
- Example: What is the propagation delay A➔Q ?



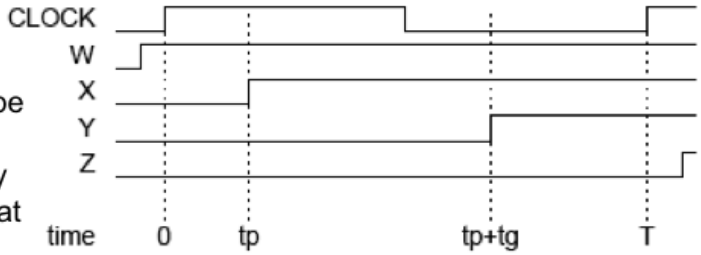
- Answer 1 (B=0):
  - Initially: A=0, B=0 => X=1, Y=0, Z=0, Q=0
  - Then: A↑ => Y↑ => Q↑ 2 gate delays
- Answer 2 (B=1):
  - Initially: A=0, B=1 => X=1, Y=0, Z=1, Q=1
  - Then: A↑ => X↓ => Z↓ => Q↓ 3 gate delays

Worst-Case Delays

- We are normally interested only in the worst-case delay from a change in any input to any of the outputs.
- The worst-case delay determines the maximum clock speed in a synchronous circuit:



$$t_p + t_g + t_s < T$$



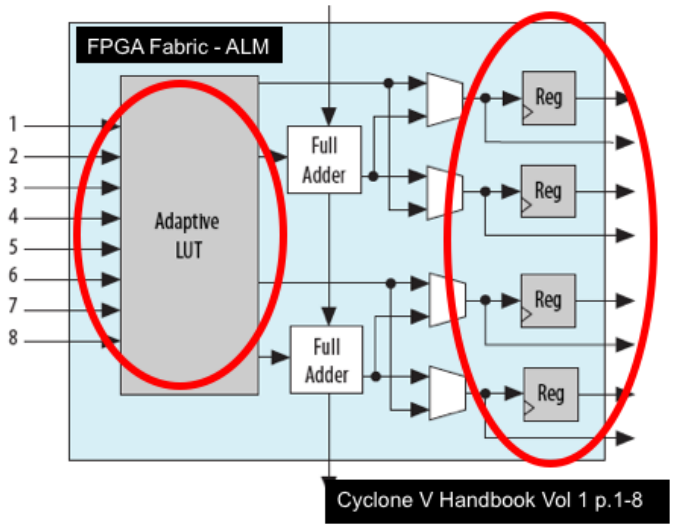
- Since the clock speed must be chosen to ensure that the circuit always works, it is only the worst-case logic delay that matters.

Cyclone V's Adaptive Logic Module (ALM)

- Let us revisit the Cyclone V logic element, the ALM (Lecture 2 slide 10)
- The Cyclone V chip on the DE1 board (5CSEMA5F31C6) has 32,000 ALMs.

- As can be seen here, the ALM has a flexible LUT for combinational logic
- It also has TWO dedicated full adders

- Each ALM can be configured to operate in one of four different modes:
  - Normal mode
  - Extended LUT mode
  - **Arithmetic mode**
  - Shared arithmetic mode



PYKC 28 Nov 2017 E2.1 Digital Electronics Lecture 15 Slide 13

We have already discussed the inside structure of the FPGA in Lecture 2. Here is a reminder.

The Altera Cyclone V FPGA has a more advanced programmable logic element than the simple 4-input LUT that we have considered up to now. The call this a Adaptive Logic Module or ALM.

An ALM can take up to 8 Boolean input signals and produces four outputs with or without a register. Additionally, each ALM also can perform the function of a 2-bit binary full adder. This is what interest us most for this lecture.

As a user of the Cyclone V FPGA, you don't actually need to worry too much about exactly how the ALM is configured to implement your design. The CAD software will take care of the mapping between your design and the physical implementation using the ALMs. It is however useful to know that as the technology evolves, more and more complicated programmable logic elements are being developed by the manufacturers in order to improve the area utilization of the FPGAs.

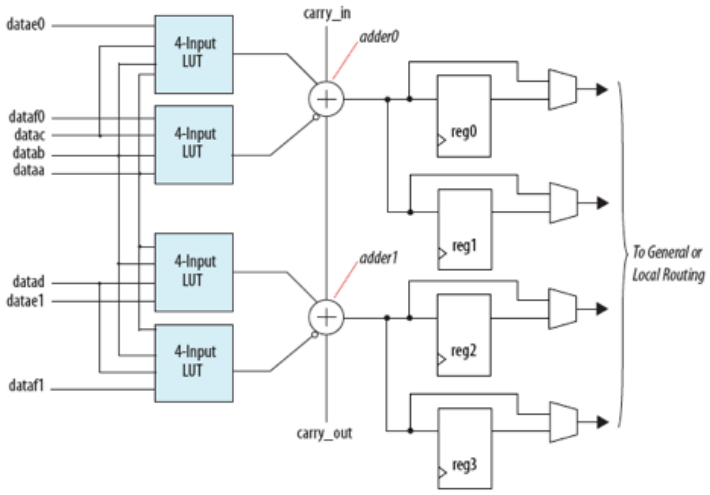
The Cyclone V on the DE1-SOC board has 32,000 ALMs, which could be estimated to be equivalent to 85K+ the old style LEs. Putting this in context,

you could put onto this one chip 2,000 32-bit binary adder circuits!

ALM's Arithmetic Mode

- The ALM in arithmetic mode uses two sets of TWO 4-inputs and two dedicated 1-bit full adders.

- The full adder can add the output of the two 4-input LUT (which can be performing some logic functions).
- There is also a dedicated carry chain, which allows fast ripple carry function.



Cyclone V Handbook Vol 1 p.1-9

How NOT to specify an adder

- You could specify a full adder in gate form
- THIS IS **BAD** for FPGA's because it will not exploit the internal **fast carry chain**
- Always use one of the following:
  - Verilog or VHDL specification with '+' operator
  - Altera (or Xilinx) adder or arithmetic block (e.g. Megafunction or Coregen)

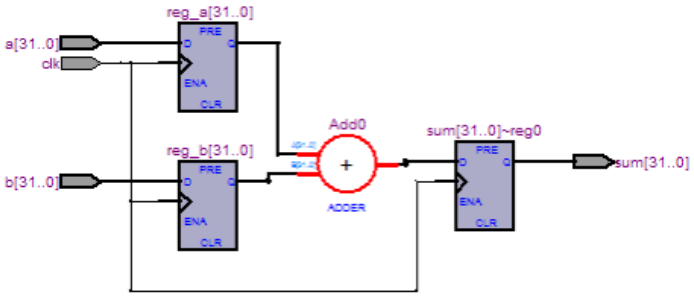
```
module full_adder(
  input a, b, cin,
  output s, cout);

  assign s = a^b^cin;
  assign cout = a&b | cin&(a^b);
endmodule
```

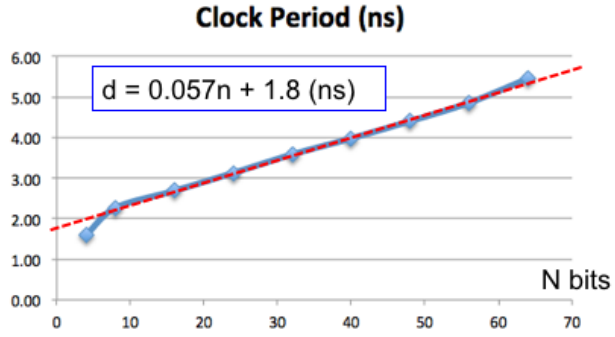
```
module add32v (a, b, sum, clk);

parameter SIZE=32;

input  [SIZE-1:0]    a,b;
input    clk;
output [SIZE-1:0]    sum;

reg    [SIZE-1:0]    reg_a, reg_b;
reg    [SIZE-1:0]    sum;

always @ (posedge clk)
    begin
        reg_a <= a;
        reg_b <= b;
        sum <=reg_a + reg_b;
    end

endmodule
```


Given that the FPGA has special adder mode, you should never specify your adder as individual full adder circuits connected together. The synthesis system WILL NOT be able to exploit the dedicated adder mode configuration shown in the previous slide. Instead use the '+' operator in Verilog as shown here. It is simpler and will produce a very fast adder.

How fast are adders in Cyclone III

- The Verilog code produces the following circuit (obtained via Tools > Netlist viewers > RTL Viewer)
- Delay obtained after compiling hardware
  (using 85 degree C, slowest timing)
- Each adder bit adds a delay of around 57ps
- Clock to Q delay + setup time ≈ 1.8ns





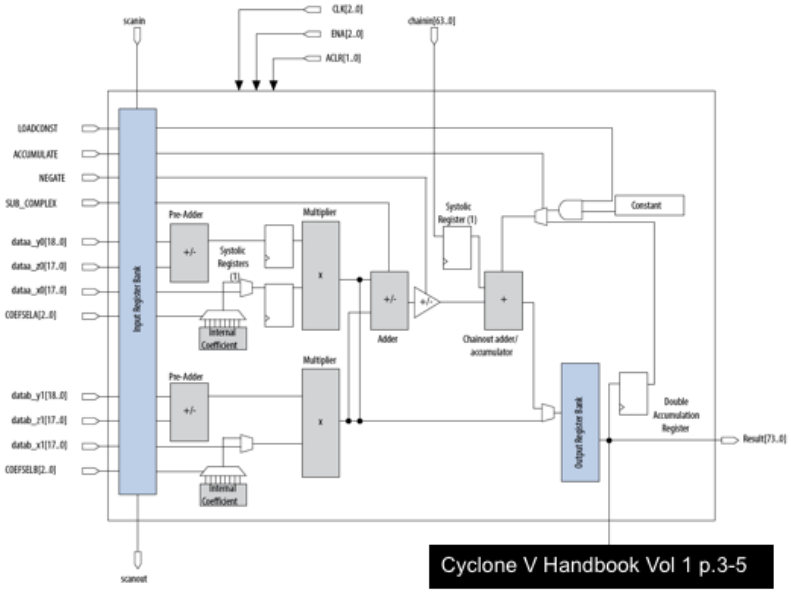
How fast are adders within a typical FPGA? Here is an n-bit adder circuit sandwiched between registers. The plot is based on a Cyclone III FPGA. (I don't have the data for Cyclone V.)

We can use the timing analyzer to estimate how fast we can clock this circuit without error as the number of bits n is increased from 1 to 64. The equation of the red fitted line. This shows that each adder bit adds around 57ps delay. In addition, there is a 1.8 ns delay from clock to Q + register setup time. I expect the timing for the Cyclone V devices to be faster.

Cyclone V DSP Block (1)

- Cyclone V has embedded DSP blocks designed for fast DSP functions.
- Each DSP block consists c

  - Input register bank
  - Pre-adder
  - Internal coefficient memory
  - Multipliers
  - Adder
  - Accumulator
  - Systolic registers (don't' worry about this for now)
  - Double accumulation register
  - Output register bank



Cyclone V Handbook Vol 1 p.3-5

Older FPGA (s.g. Cyclone III) has embedded multipliers to make implementation of digital signal processing algorithms more efficient on the FPGA. Cyclone V has DSP support far superior to just simple configurable multipliers.

The DSP blocks on the Cyclone V can be used for a combination of different functions. It can do multiplication of different precision and also to perform multiply-accumulate function.

Accumulator is an adder whose output is used as one of the two inputs of the adder on the next clock cycle. Therefore an accumulator usually only has one input, whose value get "accumulated" cycle-by-cycle.

The DSP block also has internal storage to store a constant value. Typically this is a filter coefficient for implementing a finite-impulse response (FIR) filter. Detail of how to use DSP block to implement one tap (or one stage) of a FIR filter is beyond the scope of this course. Those interested can read Cyclone V Devices Handbook, Vol. 1, p. 3-17.

Cyclone V DSP Block (2)

- For this lecture, we will only focus on the multiplier function.
- For our chip, there are 87 DSP blocks, each block can be configured as three 9x9, or two 18 x18, or one 27 x 27 multiplier(s).
- Each multiplier output could feed an adder, or an adder with output feedback to the input of the input via a register (which makes it an accumulator).
- We are using such a multiplier in Part IV of the VERI experiment.

Variant	Member Code	Variable-precision DSP Block	Independent Input and Output Multiplications Operator			18 x 18 Multiplier Adder Mode	18 x 18 Multiplier Adder Summed with 36 bit Input
			9 x 9 Multiplier	18 x 18 Multiplier	27 x 27 Multiplier		
Cyclone V SE	A2	36	108	72	36	36	36
	A4	84	252	168	84	84	84
	A5	87	261	174	87	87	87
	A6	112	336	224	112	112	112

Cyclone V Handbook Vol 1 p.3-3

For this lecture, we concentrate on the configurable multipliers in the DSP block. Each DSP block can be configured as three 9 x 9 multipliers. This is particularly useful for real-time video processing since each pixel values are often represented as an 8-bit unsigned value (or 3 x 8 bit unsigned number if we are using colour).

Alternatively, each block can provide TWO 18 x 18 bit, or one 27 x27 bit multiplier(s). Of course if you need lower number of bits (say 14 x 14), you can always either zero-extend, or sign-extend the operands to fit and use the 18 x 18 multiplier.

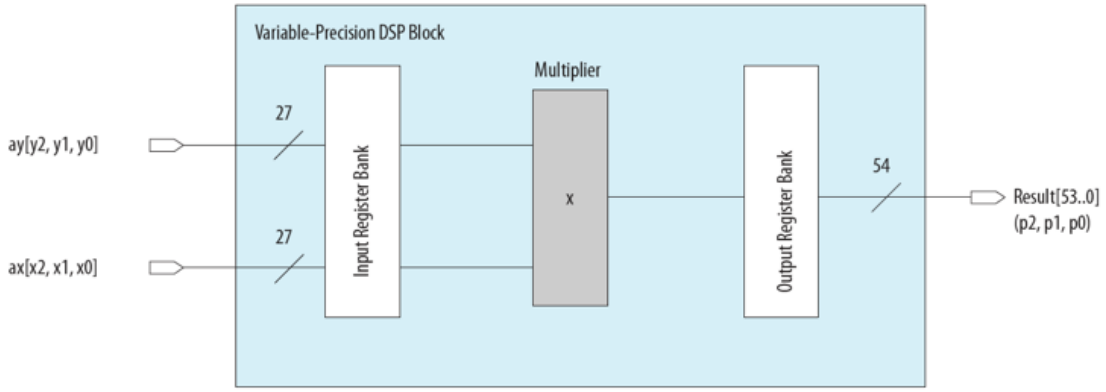
Each multiplier can also be configured to operate with the adder or the accumulator at the output.

It is important to understand that when you multiply two N x M bits unsigned numbers together, you get a product which is N+M bits. The same also applies if you multiply one signed and one unsigned number together. You can try this yourself for two four-bit numbers!

However, if you multiply two signed 2's complement numbers together, you get a product that is only N+M-1 bits. The top two-bits are always the same and they both provide the sign of the product value (i.e. you get two identical sign bits in the product). Again try this yourself with two 4-bit signed multiplication.

Cyclone V DSP Block (3)

- For example, if we the DSP block in the 9x9 multiplier mode, the function of the DSP block is shown below.
- Three pairs of 9-bit data are packed into the ax and ay input ports (27 bits).
- The three 18-bit products are packed into 56-bit Results.

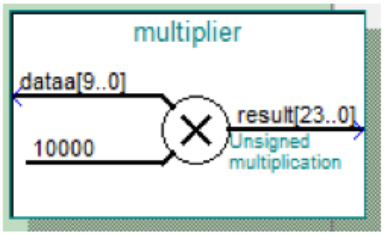


Cyclone V Handbook Vol 1 p.3-12

If you configure a DSP block to be THREE 9 x 9 multipliers, the two input operands ax and ay are bit-cascade to form a 27 bit input values to the DSP block as shown above. Similar, the three products are provided as three 18-bit RESULT value as a 54-bit value.

Using DSP Blocks in Quartus

- Quartus software includes IP cores that you can use to control the operating modes of DSP block's multipliers.
- For Part 3 and 4 of VERI, you may need a simple constant multiplier.
- Use the IP Catalog system and select **LPM_MULT** for the configurable multiplier from **Library -> Basic Functions -> Arithmetic**



To instantiate multipliers in a Cyclone V, we can use the IP Catalog tool under Quartus. Multiplier is shown under Library > Basic Functions > Arithmetic category. You should choose LPM_MULT. A dialogue form will pop up. You can then create the type of multiplier you need for your design.

Shown above is a 10 bit x 14 bit multiplier used in ex15 in Part 3 of VERI. The 10-bit input is the frequency value specified by the switches or the ADC, and the 14-bit is the multiplying constant 14'd10000.