Fixed Point Numbers

- The binary integer arithmetic you are used to is known by the more general term of Fixed Point arithmetic.
  - ⇒*Fixed Point* means that we view the decimal point being in the same place for all numbers involved in the calculation.
  - ⇒For integer interpretation, the decimal point is all the way to the right

$C0	192.	Unsigned integers, decimal point to the right.
+ $25	+ 37.	
--------	--------	
$E5	229.	

A common notation for fixed point is 'X.Y', where X is the number of digits to the left of the decimal point, Y is the number of digits to the right of the decimal point.

Fixed Point (cont).

- The decimal point can actually be located anywhere in the number -- to the right, somewhere in the middle, to the right

Addition of two 8 bit numbers; different interpretations of results based on location of decimal point

$11	17	4.25	0.07
+ $1F	+ 31	+ 7.75	+ 0.12
--------	--------	--------	--------
$30	48	12.00	0.19

xxxxxxxx.0 decimal point to right. This is 8.0 notation.	xxxxxx.yy two binary fractional digits. This is 6.2 notation.	0.yyyyyyyy decimal point to left (all fractional digits). This is 0.8 notation.

Unsiged Overflow

- Recall that a carry out of the Most Significant Digit is an unsigned overflow. This indicates an error - the result is NOT correct!

Addition of two 8 bit numbers; different interpretations of results based on location of decimal point

$FF	255	63.75	0.99600
+ $01	+ 1	+ 0.25	+ 0.00391
--------	--------	-----------	-----------
$00	0	0	0

xxxxxxxx.0 decimal point to right	xxxxxx.yy two binary fractional digits (6.2 notation)	0.yyyyyyyy decimal point to left (all fractional digits). This 0.8 notation

Saturating Arithmetic

- Saturating arithmetic means that if an overflow occurs, the number is clamped to the maximum possible value.
  - ⇒ Gives a result that is closer to the correct value
  - ⇒ Used in DSP, Graphic applications.
  - ⇒ Requires extra hardware to be added to binary adder.
  - ⇒ Pentium MMX instructions have option for saturating arithmetic.

$FF	255	63.75	0.99600
+ $01	+ 1	+ 0.25	+ 0.00391
--------	--------	-----------	-----------
$FF	255	63.75	0.99600

xxxxxxxx.0 decimal point to right	xxxxxx.yy two binary fractional digits.	0.yyyyyyyy decimal point to left (all fractional digits)

Saturating Arithmetic

The MMX instructions perform SIMD operations between MMX registers on packed bytes, words, or dwords.

The arithmetic operations can made to operate in Saturation mode.

What saturation mode does is clip numbers to Maximum positive or maximum negative values during arithmetic.

In normal mode: FFh + 01h = 00h (unsigned overflow)
In saturated, unsigned mode: FFh + 01 = FFh (saturated to maximum value, closer to actual arithmetic value)

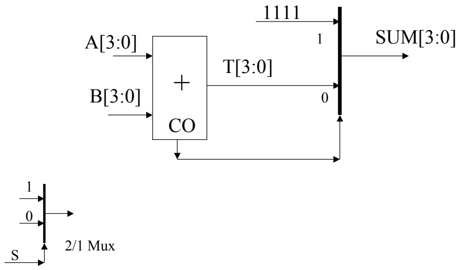
In normal mode: 7fh + 01h = 80h (signed overflow)

In saturated, signed mode: 7fh + 01 = 7fh (saturated to max value)

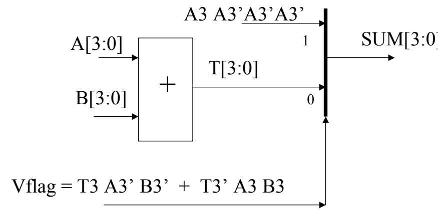
Saturating Adder: Unsigned and 2'Complement

- For an unsigned saturating adder, 8 bit:
  - ⇒Perform binary addition
  - ⇒If Carryout of MSB =1, then result should be a $FF.
  - ⇒If Carryout of MSB =0, then result is binary addition result.
- For a 2's complement saturating adder, 8 bit:
  - ⇒Perform binary addition
  - ⇒If Overflow = 1, then:
    - →If one of the operands is negative, then result is $80
    - →If one of the operands is positive, then result is $7f
  - ⇒If Overflow = 0, then result is binary addition result.

Saturating Adder: Unsigned, 4 Bit example



Saturating Adder: Signed, 4 Bit example



Vflag is true if sign of both operands are the same (both negative, both positive) and different from Sum (overflow if add two positive numbers, get a negative or add two negative numbers and get a positive number. Can't get overflow if add a postive and a negative).

Saturated value has same sign as one of the operands, with other bits equal to NOT (sign) : 0111 (positive saturation), 1000 (negative saturation).

Altera Parameterized Modules

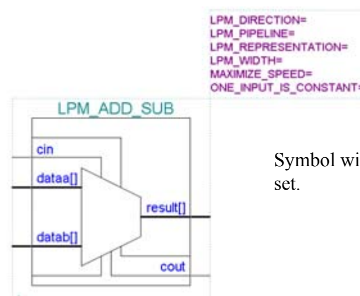
We will use Altera parameterized modules (LPMs) for many datapath functions such as adders, multipliers, muxes, counters, etc.

The port/parameter list is used to set values of parameters (such as data width) and enable/disable optional pins. Enabling/disabling optional pins adds/subtracts functionality from the LPM.

LPMs are found in the 'mega_lpm' library when you access the Altera parts list.

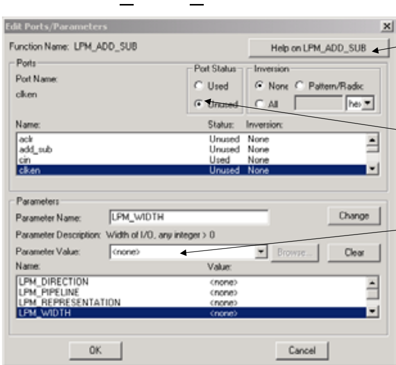
Once an LPM is placed in your schematic, select the component and choose 'Edit Ports/Parameters' to change the ports or parameters.

LPM_ADD_SUB



Symbol with no parameters set.

LPM_ADD_SUB Ports/Parameters



Help button for this LPM

Used to enable/disable selected port

Set value of selected parameter

More on LPM_ADD_SUB ports/parameters

At a minimum, you must set the *lpm_width* parameter to some value. This determines the width of the inputs/outputs.

You can enable the *add_sub* port if you want this LPM to do both addition and subtraction.

Enabling the *clock* pin and setting the *lpm_pipeline* value to something other than 0 will add pipelining to the adder (will discuss what this means later).

The parameter *maximize_speed* can be set to an integer between 0 and 10 – the higher the value, the more the adder structure will be optimized for speed.

Use the "Help on LPM_ADD_SUB" to get a full description of the ports and parameters.

Multi-Dimensional Busses

Some LPMs use multi-dimensional busses. Two separate busses:

A[7..0], B[7..0]

Can be represented by a single multi-dimensional bus:

DATA[1..0][7..0]

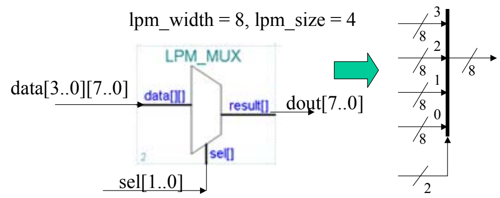
Can refer to each separate 8-bit bus via:

DATA[0][7..0]
DATA[1][7..0]

LPM_MUX

LPM_WIDTH = 8, LPM_SIZE = 4 gives a 4-to-1 mux with 8 bit inputs

lpm_width = 8, lpm_size = 4



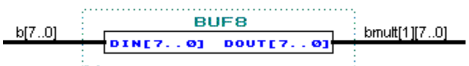
lpm_size controls number of input busses, *lpm_width* controls width of each input bus. Width of 'sel' input will ceiling($\log_2$(*lpm_size*))

Connecting to Multi-dimensional busses

To connect a single dimensional bus such as A[7..0] to a multi-dimensional bus, one way to do it is to write a VHDL mode that is a buffer function:

dout <= din;

These buffers will be removed during the optimization/mapping process.

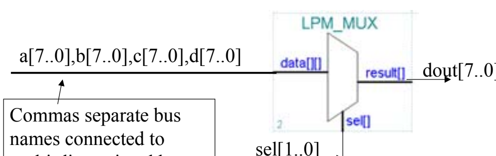


Connects single dimensional bus *b[7..0]* to multi-dimension bus *bmult[1][7..0]*

Connecting to Multi-dimensional busses (cont)

Another way is through bus labeling

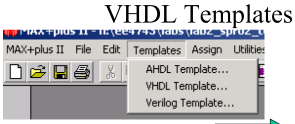
lpm_width = 8, lpm_size = 4

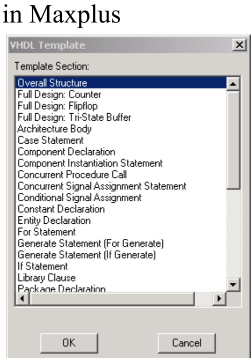


Commas separate bus names connected to multi-dimensional bus. First bus name connected to highest index, last bus name to lowest index.

a[7..0] → data[3][7..0]
b[7..0] → data[2][7..0]
c[7..0] → data[1][7..0]
d[7..0] → data[0][7..0]

VHDL Templates in Maxplus





When editing a VHDL text file, can insert VHDL templates into the file. Do this to avoid common syntax errors.

Also, make sure the file extension is '.vhd' when saved. The default extension is '.tdf', which causes the file to be processed with 'Altera HDL' compiler, not the VHDL compiler!!!

Multiplication

Multiplication of a K bit number by an L bit number gives a product that is K+L bits wide.

Usually, both operands are same width. So N x N multiplication gives a product that is 2N bits wide:

```
   % 110              6
 x % 111          x   7
 ----------       --------
      110            42
     110
    110
 ----------
   101010 = $2A = 42
```

Note that 3 bits x 3 bits gives 6 bit product.

Multipliers and Datapaths

Typically, a datapath is of fixed width. A multiplier output then needs to be the same width as the operands. So, for N bit operands, only N bits of the 2N bit product will be kept.

Obviously, want to drop the N least significant bits to form the truncated result.

```
   % 110          0.75
 x % 111       x  0.875
 ----------    --------
      110        0.65625
     110
    110        Fixed point
 ----------    representation
   101010
```

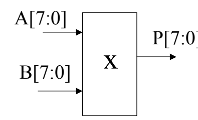
6 bits of precision:
101010 = 0.5 + 0.125 + 0.03125
= 0.65625

3 bits of precision
101 = 0.5 + 0.125
= 0.625

Multiplier as a Building Block

We will use a multiplier as a building block in this class. A Computer Arithmetic class can show you how the internals of a multiplier is constructed.

The most common use of a multiplier is as a combinational logic block. Given a change on the INPUTs, the OUTPUT will be ready after a propagation delay.



8x8 multiplier, only keep 8 bits of product.

We will talk more about multipliers later in the semester.

Example Fixed Point Application

Colors in Computer Graphics applications represented by Red, Green, Blue (RGB) components.

Each component (RGB) is 8 bits; hence the term 24 bit color.

As an 0.8 Fixed point number, colors range from:
0.0 =< color < 1.0
(dark colors) (light colors)

0.0 = % 00000000

Blue variation

0.99 = % 11111111

Blend Operation

A blend operation takes two colors and blends them together to form a new color. The *Blend Factor* (F) controls how much each color contributes

Cnew = Ca * F + (1 - F) Cb

If F is 0.5 (%10000000) then the new color is an equal blend of Ca, Cb.

If F is 0, then new color is simply Cb.

If F is 1, then new color is simply Ca. (can't get 1.0 with just 8 bits, will talk more about this problem later).

Representing 1.0

When the multiplication Ca * F is performed, if F = 1.0 want the result to be exactly equal to the original value 'Ca'.

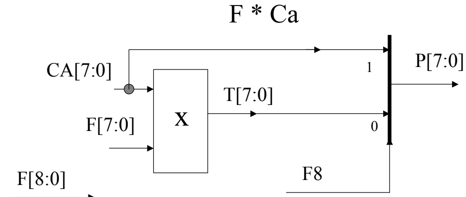
However, the closest we can get to 1.0 using 8 bits (assuming 0.8 fixed point notation) is $0.11111111_2 = 0.996_{10}$

0.996 x Ca is NOT EQUAL to Ca!

To solve this problem, we will use 9 bits to represent the 'F' value. The lower 8 bits will be the fractional representation of F. If F=1.0, then the MSB of F is equal to a '1', and the other bits are a don't care.

When multiplying Ca * F, will use the lower 8 bits of F for the multiply. If the MSB of F = '1', then ignore output of multiplier and use 'Ca'.

F * Ca



If F = 1.0, then F = '1xxxxxxxx' (MSB of F = '1').

Note that an 8 x 8 bit multiply actually produces 16 bits. We are dropping the lower 8 bits. In 0.8 fixed point notation, this means that we are ignoring the lower 8 least significant bits which is a fractional part that is less than $1/2^8$ (ignoring fractional part < 0.00390625).

1.0 -F

Because speed is important, 1-F will not use a subtractor. The following is done instead:

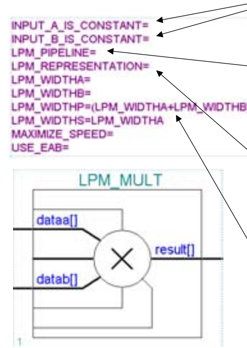
If F = 1.0 (F8 = ‘1’), then result of 1.0 -F = 0.0 (‘000000000’)

Else if F = 0, the result of 1.0 – F = 1.0 (F = ‘100000000’)

else F8 = ‘0’, F[7..0] = complement of (F[7..0]).

Note that if F is not equal to 1.0 or 0.0, the subtraction of 1.0 –F is estimated by complementing the lower 8-bits of F. This will be incorrect by 1 LSB, but will save gates and increase speed.

Multiplier LPM - *lpm_mult*



Can specify a constant input value for ‘A’ or ‘B’, will generate a more efficient multiplier.

Will use this later to add pipeline stages to multiplier. Also will have to enable the clock pin.

Use this to specify “SIGNED” or “UNSIGNED” (default value) – hardware is different!

Width of output, we will usually set width of output to be same as input A,B input width – will drop least significant bits.