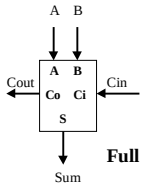
Arithmetic Operations

- We will review the arithmetic building blocks we have previously used, and look at some new ones.
  - Addition
  - incrementer
  - Addition/subtraction
  - decrementer
  - Comparison

Binary Adder

F (A,B,C) = A xor B xor C G = AB + AC + BC

These equations look familiar. These define a *Binary Full Adder* :

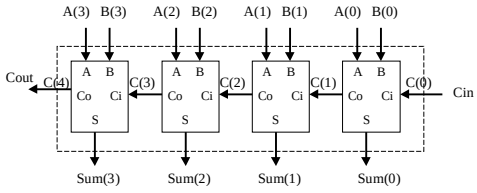


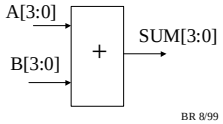
Sum = A xor B xor Cin

Cout = AB + Cin A + Cin B
= AB + Cin (A + B)

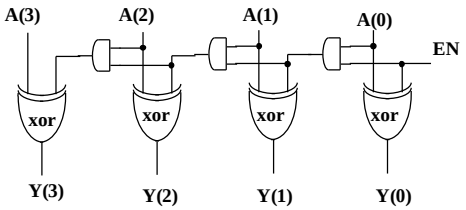
Full Adder (FA)

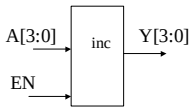
4 Bit Ripple Carry Adder





Incrementer





If EN = 1 then Y = A + 1
If EN = 0 then Y = A

BR 8/99

How did we get the Incrementer equations?

Full Adder equations:

Sum = A xor B xor Cin

Cout = AB or Cin A or Cin B
= AB or Cin (A or B)

Let B = 0, Cin = 1 so that Sum = A + 1. Then equations simplify to:
SUM = A xor 1 xor 0 = A xor 1 = A'
Cout = 0 or 1 (A or 0) = A.

If we want an "En" input, then we want SUM = A if En=0, else SUM = A+1 if En = '1'. Filling in the above equations:

SUM = A En' or A' En = A xor En
Cout = A En (note that Cout = 0 if En = 0).

The "Cout" of one bit becomes the "En" signal for the next bit!!!!

BR 8/99

A Subtractor

What is subtraction?

A - B = A + (-B)

How do you take the negative of a number? Depends on the sign representation (signed magnitude, 1s complement, 2s complement). Lets assume 2's complement since it is most common).

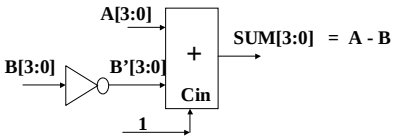
(-B) = B' + 1

So:

A - B = A + (-B) = A + B' + 1

BR 8/99

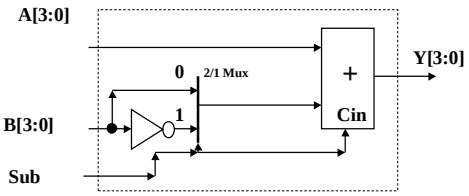
Subtractor using an Adder



What if we want a block that can do both addition and subtraction?

BR 8/99

Adder/Subtractor



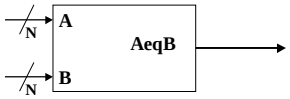
VHDL representation:

Y <= (A-B) when (Sub = '1') else A+B;

BR 8/99

Recall what a Comparator is...

Equality comparator.



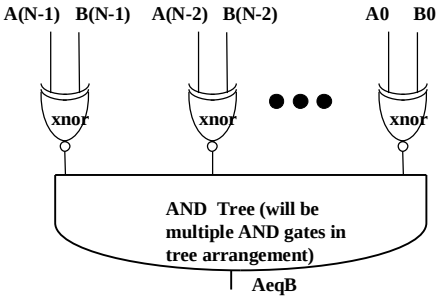
A=B if A(0) = B(0) and A(1) = B(1) … and A(n-1)=B(n-1)

Recall that "xnor" function is '1' if A=0, B=0 or A=1, B=1!
So AeqB is:

AeqB = (A(0) xnor B(0)) and (A(1) xnor B(1)) and ….etc.

BR 8/99

What is logic structure for equality comparator?

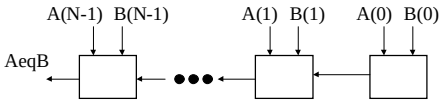


BR 8/99

Is there another Logic structure possible?

Compare "iteratively" from LSB to MSB

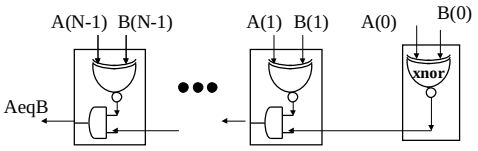
```
If (A(0) = B(0) then
  if (A(1) = B(1) then
   ….
    If (A(N-1) = B(N-1) then
        AeqB = '1';      !!!!!
```



Signal from one bit block to next is "enable" for that block.

BR 8/99

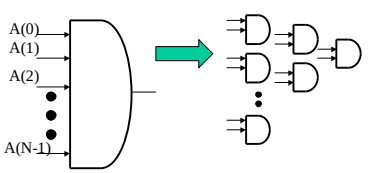
Iterative Comparator Structure



An advantage to this structure is that the design for each bit is the same same, and we can extend it indefinitely. But it will be slow.

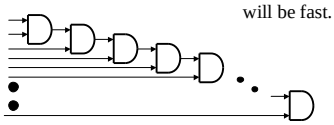
BR 8/99

Two ways to do a Large AND function



Multi-level. # of levels depends on total number of inputs, number of inputs on each gate. A tree arrangment like this will take more gates, but will be fast.

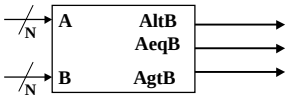
Serial arrangement. Will take less gates, but will be slow.



BR 8/99

What about "<" (less than), ">" (greater than?)

Full comparator.



The logic for AltB, AgtB depends on whether we are comparing signed numbers or not. We will assume unsigned numbers for now.

BR 8/99

Logic for "AgtB" (unsigned)

Consider A > B, both N bit numbers, A[N-1:0], B[N-1:0]

If (A(N-1) = '1' and (B(N-1) = '0') then
AgtB = '1'; ← A=1xxx… B=0xxxxx
elsif ((A(N-1) = B(N-1)) and (A(N-2) = '1' and (B(N-2)='0')) then
AgtB = '1';
etc...

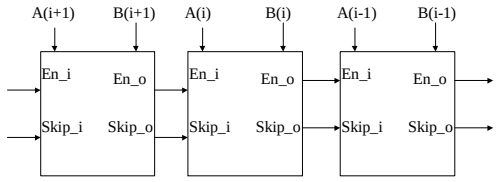
A=01xx… B=00xxxx
A=11xx… B=10xxxx

Look at "bit(i)". The enable signal from previous bit is A= B up until now . If this is '1', then we need to do a comparison.

However, if "AgtB" is already true, then we don't need to do comparison and can skip this comparison!

BR 8/99

Iterative Implementation of AgtB



en_o = (A xnor B) and en_i ; ← A=B signal

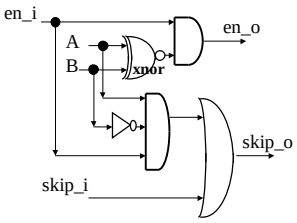
If (skip_i = '1') then
  skip_o = '1';
else
  skip_o = en_i and (A and B') ;
end if;

BR 8/99

Logic Implementation

en_o = (A xnor B) and en_i;

If (skip_i = '1') then
  skip_o = '1';
else
  skip_o = en_i and (A and B') ;
end if;



Can use a K-map to simplify this logic.

The *skip_o* of the LAST bit is the AgtB signal!

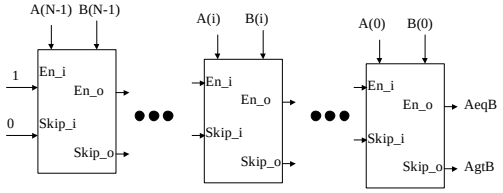
The *en_o* of the LAST bit is the AeqB signal!

What about AltB???

AltB = AgtB' and AeqB'

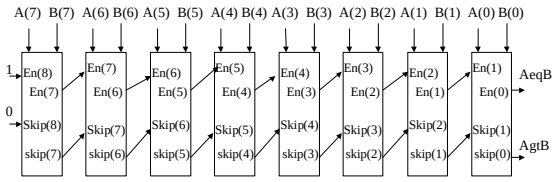
BR 8/99

Final Comparator



AltB <= not (AeqB) and not (AgtB);

BR 8/99



8 Bit Comparator

For i'th bit:

en(i) = (A(i) xnor B(i)) and en(i+1);

If (skip(i+1) = '1') then
  skip(i)= '1';
else
  skip(i) = en(i) and (A(i) and B'(i)) ;
end if;

BR 8/99

```
architecture a of comp is
signal en, skip:  std_logic_vector(8 downto 0);

 begin
  aeqb <= en(0);
  agtb <= skip(0);
  altb <= (not en(0))  and (not skip(0));

  process (a,b)
  begin
    en(8) <= '1';  skip(8) <= '0';
    for i in 7 downto 0 loop
      en(i) <=  not (a(i) xor b(i)) and en(i+1);
      if (skip(i+1) = '1') then
          skip(i) <= '1';
       else
          skip(i) <= en(i+1) and (a(i) and not b(i));
       end if;
    end loop;
  end process;
end a;
```

VHDL architecture that implements comparator logic as shown on previous slides.

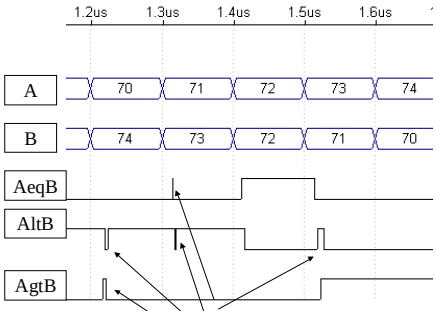
BR 8/99

Alternate VHDL specification

```
architecture a of compa is

 begin

  aeqb <= '1' when (a = b) else '0';
  agtb <= '1' when (a > b) else '0';
  altb <= '1' when (a < b) else '0';


end a;
```

Synthesis tool will pick a logic implementation for implementation of '=', '>', '<' based on user constraints such as propagation delay.

BR 8/99



Glitches in synthesized logic.
Ok as long as we are using FFs
to latch result.

BR 8/99