LINKÖPING UNIVERSITY
Department of Electrical Engineering

TSIU03, SYSTEM DESIGN

LECTURE 3

Kent Palmkvist
Kent.Palmkvist@liu.se

Slides by: Mario Garrido Gálvez (mario.garrido.galvez@liu.se)

Linköping, 2021

TODAY

- Changing the word length.
- Circuits for mathematical operations and how to handle them in VHDL: Adders, subtracters, multipliers, dividers.
- Description of Lab 2.

CHANGING THE WORDLENGTH

- Given a binary number represented in 2's complement or as unsigned, it is possible to change the word length of the number and still represent the same number.
- If we increase the number of bits, we will call it sign extension.
- If we reduce the number of bits, we will call it truncation.

SIGN EXTENSION

- Unsigned: the sign extension for unsigned consists of adding 0 to the most significant part of the number:

1001 ≡ 9 (represented with 4 bits)

00001001 ≡ 9 (represented with 8 bits)

```
z <= "000…000" & a;
```

Signed: the sign extension for signed consists of copying the MSB and adding it to the most significant part of the number.

01001 ≡ 9 (represented with 5 bits)

00001001 ≡ 9 (represented with 8 bits)

1001 ≡ -7 (represented with 4 bits)

11111001 ≡ -7 (represented with 8 bits)

```
z <= a(n-1)& a(n-1) & … & a(n-1) & a;   (a has n bits)
```

ADDING BITS TO THE LSB

- What happens if we add zeros to the least significant part of the number?
- Unsigned: 1001 ≡ 9
  10010 ≡
  100100 ≡
- Signed: 1001 ≡ -7 01001 ≡ 9
  10010 ≡ 010010 ≡
- Multiplication by powers of 2:

```
z <= a & "000…00";
```


TRUNCATION (REMOVING BITS)

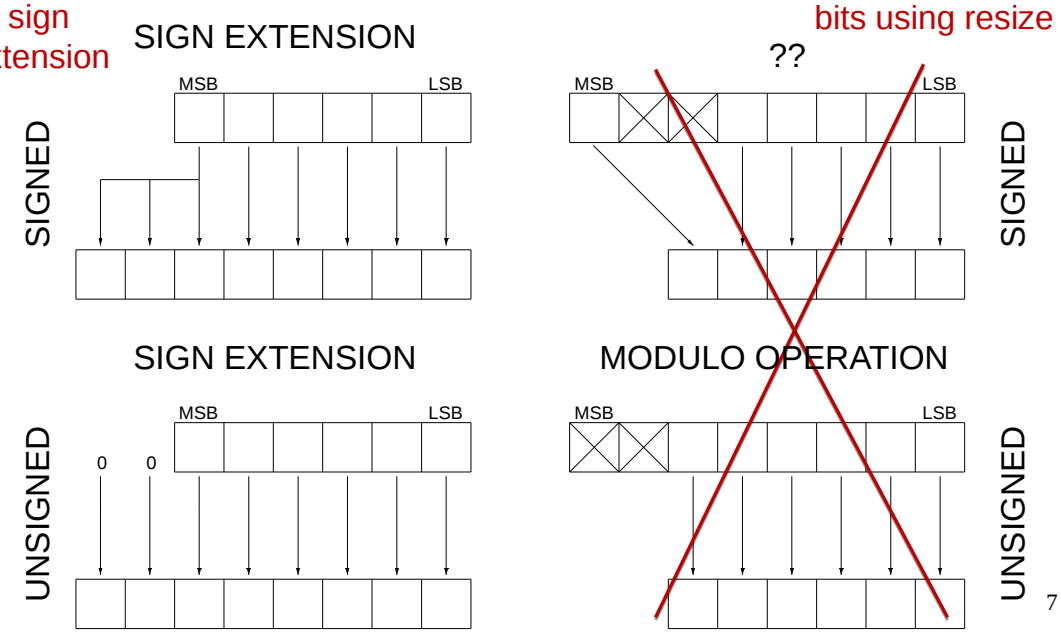
	Unsigned	Signed	Operation
Remove MSBs:	`z <= a(k-1 downto 0);`		$\mathrm{mod}(a,2^k)$
	1001 ≡ 9	1001 ≡ -7	
	001 ≡ 1	001 ≡ 1	
Remove LSBs:	`z <= a(n-1 downto k);`		$\lfloor A/2^k \rfloor$
	1001 ≡ 9	1001 ≡ -7	
	100 ≡ 4	100 ≡ -4	

RESIZE

Useful for sign extension

```
z <= resize (a,numBits);
```

Not advisable to reduce bits using resize



ADDITION OF BINARY NUMBERS

- Addition of unsigned and 2's complement binary numbers is done in the same way as is done for decimal numbers.

$$0010 \equiv 2$$
$$\underline{+\ 0011 \equiv 3}$$
$$0101 \equiv 5$$

- Which is the results of adding these two binary numbers?

$$1010 \equiv 10 \quad \text{(or -6 in 2's complement)}$$
$$\underline{+\ 1001 \equiv 9} \quad \text{(or -7 in 2's complement)}$$

- What happens with the wordlength? What happens if we want to keep the wordlength?

OVERFLOW

- When a number gets larger than the number of bits that we can use to represent it, and some of the most significant bits are trashed, we may get unexpected results. Therefore, be sure that you use enough bits to represent any number that you could get.
- Example in 2´s complement:

$$\begin{aligned} 0110 &\equiv 6 \\ +\ 0011 &\equiv 3 \\ \hline 1001 &\equiv -7 \end{aligned}$$

In order to represent 9, we need to add 1 bit:

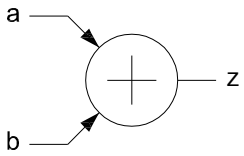
$$01001 \equiv 9$$

WHAT DOES THIS CIRCUITS DO?

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity add1 is
   port (a,b: in unsigned (7 downto 0);
         z : out unsigned (7 downto 0));
end add1;
architecture rtl of add1 is
begin
   z <= a + b;
end rtl;
```

ADDER

- An adder adds two binary numbers:




```
z <= a + b;
```

- In VHDL, a, b and z must have the same word length. However, this may cause overflow!!
- We can:

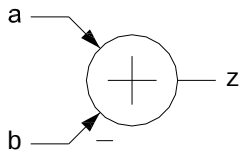
  - Sign extension of a and b by one bit before the addition, and define z as one bit longer than a and b.

  - … and then truncate the LSB of z if we want to keep the WL.

  - Make sure that the input values will never cause overflow.

SUBTRACTER

- A subtracter subtracts two binary numbers:




```
z <= a - b;
```

- The symbol – is used to indicate which input has to be subtracted from the other one.
- The same as an adder with respect to overflow.

COMPARATOR

- With the circuits studied so far, how can I design a circuit that compares two numbers in 2's complement (8 bits) according to:

$$z = \begin{cases} 1 & if \quad a > b \\ 0 & otherwise \end{cases}$$

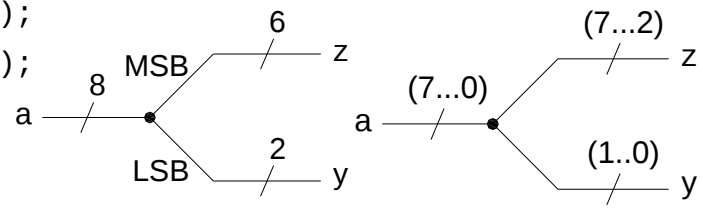
COMPARATOR IN VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity comp is
   port (a,b: in signed (7 downto 0);
         z : out std_logic);
end comp;
architecture arch of comp is
   signal p: signed (8 downto 0);
begin
   p <= resize(b,9) – resize(a,9);
   z <= p(8);
end arch;
```

SPLIT / MERGE BITS

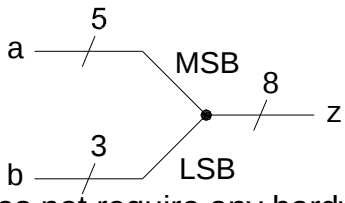
- If we want to split the bits of a signal or extract some bits from the signal, we can represent it in a circuit in the following way:

```
z <= a(7 downto 2);
y <= a(1 downto 0);
```



- If we want to merge bits of different signals into the same signal, we can represent it as:

```
z <= a & b;
```



- Splitting and merging bits does not require any hardware component. They are done for free. Here we only show how to represent them.

FROM LAST LECTURE

- By the way, which mathematical function does this line of code implement? Can you draw a circuit that calculates this function?

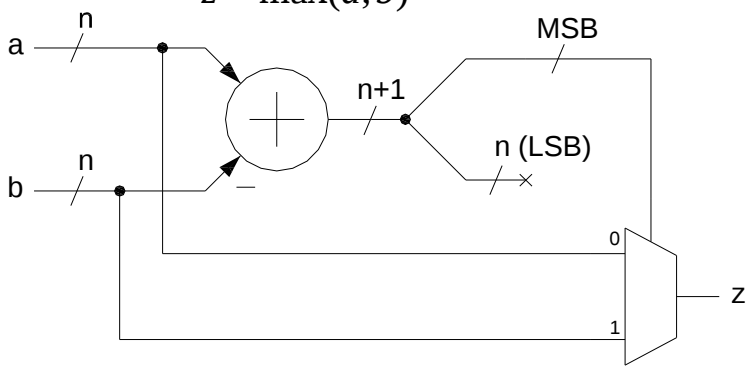
```
z <= a when a > b else b;
```

FROM LAST LECTURE

- By the way, which mathematical function does this line of code implement? Can you draw a circuit that calculates this function?

```
z <= a when a >= b else b;
```

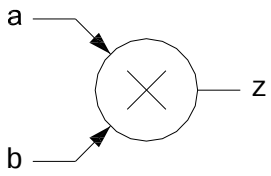
$$z = \max(a, b)$$



- Note how to represent the word length and the truncation in a circuit.

MULTIPLIER

- A multiplier multiplies two binary numbers:

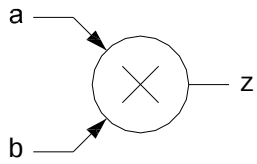



```
z <= a * b;
```

- The output z must have a word length equal to the sum of the word lengths of a and b.
- Therefore, z may result in a very large number of bits.
- If z gets too large, the most typical approach is to truncate some LSBs of z, and take into account that the representation of the number is scaled.

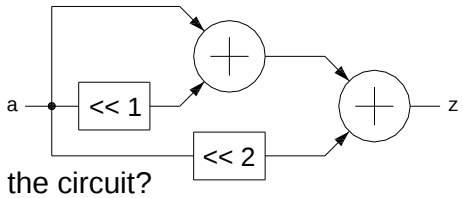
CONSTANT MULTIPLIER

- And if one of the inputs is constant?



- One option: just use the multiplier as normal.
- Other option with less resources (an addition is much cheaper than a multiplication):

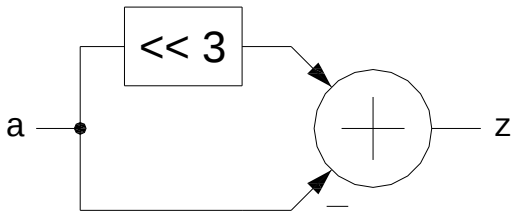
Multiplication by 7:



- Which are the word lengths in the circuit?
- Multiplications by powers of 2 are for free in hardware!!
- Can we simplify the circuit even more?

…YES!

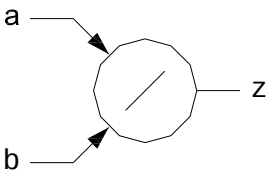
- Constant multiplication by 0111 ≡ 7 using only one adder:



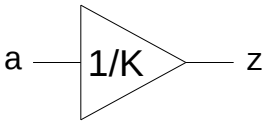
- 7 is equal to 8 - 1, isn’t it?

DIVIDERS

- A divider divides two binary numbers:

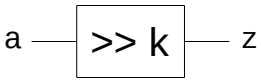


- A divider is costly and requires a specific design.
- For constant division we can use a multiplier that multiplies by its inverse:



- And if it is a division by a power of two, we can just remove the least significant bits (truncation happens):

```
z <= a (WL -1 downto k);
```



WHICH IS THE DIFFERENCE?

```
library ieee;
use ieee.std_logic_1164.all;
entity and1 is
   port (a,b: in std_logic;
         z : out std_logic);
end and1;
architecture arch of and1 is
   signal p: std_logic;
begin
   p <= a AND b;
   z <= p;
end arch;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity and2 is
   port (a,b: in std_logic;
         z : out std_logic);
end and2;
architecture arch of and2 is
   signal p: std_logic;
begin
   z <= p;
   p <= a AND b;
end arch;
```

LAB 2: KEYBOARD

- Connect a Keyboard to the DE2 board.
- Detect codes of the keyboard (numbers pushed).
- Show the code that is received using LEDs.
- Show the number pushed in the 7-segment display.
- Create a test bench to test the circuit.

CHECKLIST FOR LECTURE 3

- Sign extension, truncation, overflow.
- Combinational circuits: adders, subtracters, multipliers, constant multipliers, dividers.
- VHDL language: resize, +, *, -, the order of the statements in VHDL does not matter.

AT HOME

- Review the checklist for lecture 3 and check that you understand all the concepts and you know how to use them.
- Have a look at Lab 2
- Complete and submit answers to Assignment 1 if you have not done that yet. Deadline 7/9 at 08.15 (before lecture 5)
- Do the Assignment 2. It has to be submitted in Lisam before end of 9/9.