DAT093
Introduction to Electronic System Design

Laboratory assignment 1

Basic arithmetic

Sven Knutsson
svenk@chalmers.se
Dept. Of Computer Science and Engineering
Chalmers University of Technology
Gothenburg
Sweden

Goal

Learning to use basic arithmetic methods

- Addition
- Subtraction
- Multiplication
- We leave out division

We will continue the work we did in the introductory lab assignment

You should try to make your designs generic which in these cases mean that you can change the number of bits in the vectors used in the calculations by just changing the value of a generic parameter without the need to rewrite the code

Testing your designs

You should test your designs using simulation in QuestaSim

You should write script files (do files) to control the simulation

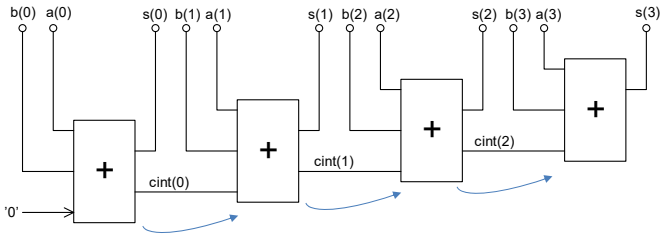
There are also QuestaSim test benches on the course homepage that you can use to test your designs but this doesn´t free you from writing your own do files

If your codes pass the supplied test benches then you pass the lab

Ripple carry adder

We will start with the ripple carry adder we used in the introductory lab assignment.

The figure shows a four bit instantiation of the ripple carry adder.

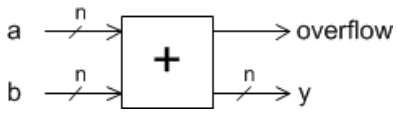


The carry bit ripples from bit to bit (LSB to MSB)

Note that in this case we have just two four bit vectors, no carry in or carry out bit

Overflow

We add overflow indication



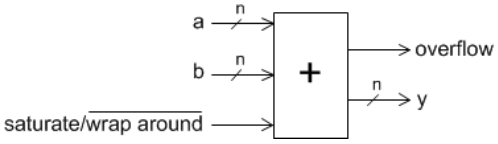
Overflow means that the result doesn´t fit within the given number of bits

In this case we only detect overflow, we don´t do anything about it

You can see in the lab assignment how to test for overflow

Saturation

We add the option for saturation

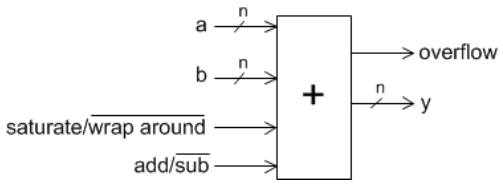


Saturation means that if the result overflows then the result should be set to maximal positive value or maximal negative value depending on at what border the overflow occurs

We add a control to the previous design controlling if the result should wrap around (overflow) or saturate

Subtraction

We don´t create a separate subtracter but we add the option of subtraction to our adder



Subtraction could easiest be done by using 2´s complement

```
a – b = a + (-b)
```

Instead of creating a subtractor we change the sign of the b value using 2's complement and use the earlier adder with some additional functions

Serial implementation

So far the implementations have been parallel meaning that we use one adder for each bit in our words

We can serialize the design by using only one adder and do the calculations bit by bit

Let´s look at a four bit example

$$\begin{array}{cccccc} c_3 & c_2 & c_1 & c_0 & 0 & \\ & a_3 & a_2 & a_1 & a_0 & \\ & b_3 & b_2 & b_1 & b_0 & \\ \hline & s_3 & s_2 & s_1 & s_0 & \end{array}$$

First calculation

No carry in

Serial implementation

So far the implementation have been parallel meaning that we use one adder for each bit in our words

We can serialize the design by using only one adder and do the calculations bit by bit

Let´s look at a four bit example

$$\begin{array}{cccccc} c_3 & c_2 & c_1 & c_0 & 0 & \\ & a_3 & a_2 & a_1 & a_0 & \\ & b_3 & b_2 & b_1 & b_0 & \\ \hline & s_3 & s_2 & s_1 & s_0 & \end{array}$$

Second calculation

Carry in from bit 0

Serial implementation

So far the implementation have been parallel meaning that we use one adder for each bit in our words

We can serialize the design by using only one adder and do the calculations bit by bit

Let´s look at a four bit example

$$\begin{array}{cccccc} c_3 & c_2 & c_1 & c_0 & 0 & \\ & a_3 & a_2 & a_1 & a_0 & \\ & b_3 & b_2 & b_1 & b_0 & \\ \hline & s_3 & s_2 & s_1 & s_0 & \end{array}$$

Third calculation

Carry in from bit 1

Serial implementation

So far the implementation have been parallel meaning that we use one adder for each bit in our words

We can serialize the design by using only one adder and do the calculations bit by bit

Let´s look at a four bit example

$$\begin{array}{cccccc} c_3 & c_2 & c_1 & c_0 & 0 & \\ & a_3 & a_2 & a_1 & a_0 & \\ & b_3 & b_2 & b_1 & b_0 & \\ \hline & s_3 & s_2 & s_1 & s_0 & \end{array}$$

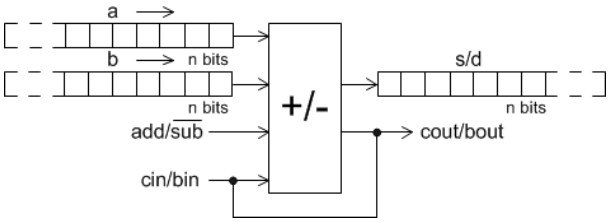
Fourth calculation

Carry in from bit 2

Serial implementation cont.

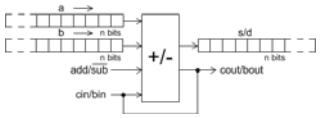
If we turn the math into a block diagram we get a design with only one adder component.

The subtraction is as before done by using 2´s complement



To handle the carry correctly we must go from LSB to MSB

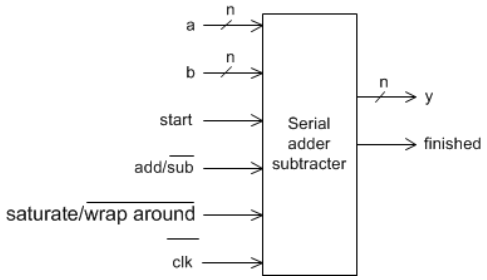
Serial implementation cont.



To make it work we must introduce a `clock` that will control the shifting in and out of bits

We also introduce a `start` signal to start the calculation

And a `finished` signal to indicate when all bits are done



Write your code so that the output is only updated when the full calculation is done, don´t output any intermediate results

Multiplication using succesive addition

If we look at multiplication done on paper

```
    1011
*   1101
```

We can interprete this as a number of multiplications, additions and shifts

Since our values are only one and zero the multiplication really means adding the value or not, don´t forget the shift though

The result depends on if we treat our values as signed or unsigned.

Multiplication using succesive addition

Let´s start with the unsigned case.

```
1011 ->  11
1101 -> *13
        143 -> 10001111
```

Multiplication using succesive addition

```
     1011
*    1101
 00000000
+00001011
 00001011
```

Add vector a

Multiplication using succesive addition

```
      1011
*     1101
 ---------
  00001011
+00000000
 ---------
  00001101
```

Don´t add vector a

Shift

Multiplication using succesive addition

```
      1011
*     1101
 ---------
  00001011
+00101100
 ---------
  00110111
```

Add vector a

Shift

Multiplication using succesive addition

```
     1011
*    1101   <-- Add vector a
 00110111
+01011000   <-- Shift
 10001111 -> 143
```

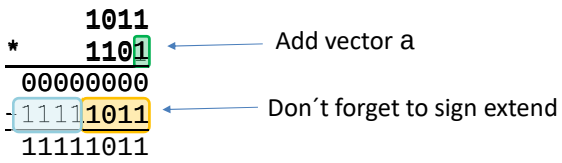
Correct!

Multiplication using succesive addition

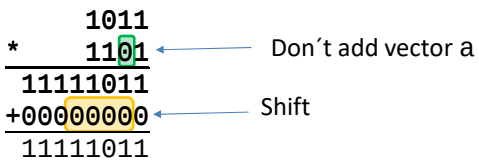
Now let´s look at the signed case.

```
1011 ->   -5
1101 -> *(-3)
          15 -> 00001111
```

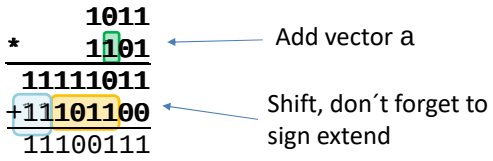
Multiplication using succesive addition



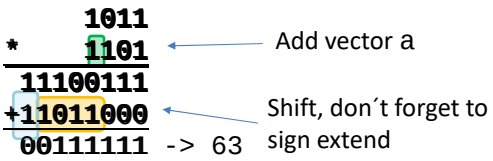
Multiplication using succesive addition



Multiplication using succesive addition



Multiplication using succesive addition



This is not correct, we were expecting the result 15.

What´s going on?

Multiplication using succesive addition

When we have signed number we must do a subtraction for the MSB, not an addition. We use 2´s complement

```
     1011
**   1101
-----------
 11100111
  00100000
+        1
-----------
 00001111  -> 15
```

Add vector a´s 2-complement

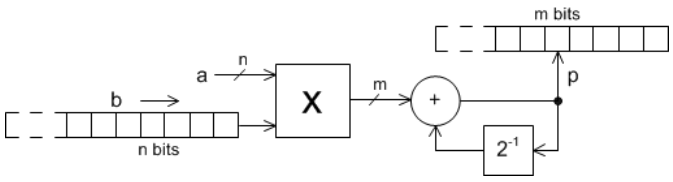
Shift, don´t forget to sign extend (not needed here)

Now the result is correct

We can simplify things by multiplying the absolute values of the numbers and then take care of the sign afterwards

Multiplication using succesive addition con´t.

We can implement this serially



In this schematic we don´t shift the operand to the left before the addition. Instead we shift the result to the right which will give the same result

Multiplication using succesive addition con´t.

The interface is very simular to the serial adder