Lab session #9

Activity #1: Square-root approximation circuit

Given 2 8-bit signed integers *a* and *b*, compute an approximation of $\sqrt{a^2 + b^2}$ using the following formula:

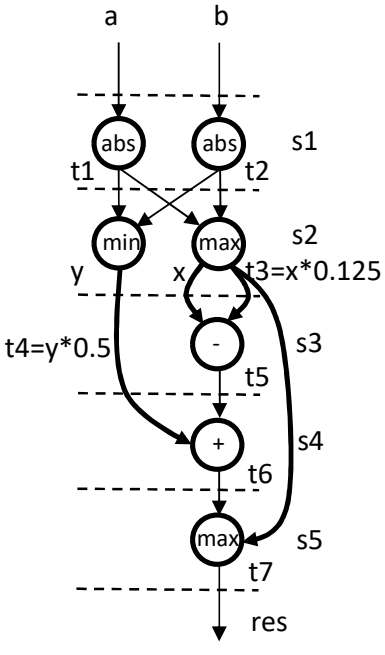
$$\sqrt{a^2 + b^2} \approx max(\left(x - \frac{1}{8}x\right) + \frac{1}{2}y), x)$$

where $x = \max(\text{abs}(a), \text{abs}(b))$ and $y = \min(\text{abs}(a), \text{abs}(b))$. The following pseudo-code implements this computation:

```
read(a);
read(b);
t1 = abs(a);
t2 = abs(b);
x = max(t1, t2);
y = min(t1, t2);
t3 = x*0.125;
t4 = y*0.5;
t5 = x - t3;
t6 = t4 + t5;
t7 = max(t6, x)
res = t7;
```

This circuit is *data-dominated*, i.e., it involves mainly data manipulations and arithmetic operations, and the control part is negligible. In this case the circuit could be a purely combinational one, but this would entail the use of a large amount of functional units (cfr. pseudo-code). An alternative is to resort to a FSM-D, deciding in advance how many functional units may be used to perform required operations (*binding*) and when (*scheduling*). If there is a large degree of parallelism in circuit operation, having many functional units boosts speed at the expense of hardware cost. In this case the degree of parallelism is limited, so saving functional units at the expense of speed may be a good trade-off. Binding and scheduling are typical tasks of high-level synthesis tools.

A possible schedule is shown in the following figure, where horizontal dashed lines represent the "boundaries" between the states of the FSM-D.



Design a FSM-D that implements the above schedule. The FSM-D has the 8-bit *a* and *b* signals as inputs, as well as a *start* signal. Its outputs are the 9-bit *res* signal and the *ready* signal. Do not just map variables `ti` to registers, but minimize the number of registers, reusing a register for another variable when the value it contained is no more needed. Remark that dividing by 2 and by 8 does not require any functional unit.

Steps:
1. Draw the HLSM state diagram and write the VHDL code to capture the HLSM behavior (cfr. VHDL: Embedded Systems RTL Design: capture HLSM behavior).
2. Design a FSM-D:
   a. creating 2 processes for behavioural datapath description from the HLSM state diagram (cfr. VHDL: Embedded Systems RTL Design: Describing a datapath behaviourally)
   b. create 2 processes for behavioural controller description from the HLSM state diagram (cfr. VHDL: Embedded Systems RTL Design: Describing the controller behaviourally)
   c. connecting datapath processes to controller processes to create a single architecture (cfr. VHDL: Embedded Systems RTL Design: Connecting controller and datapath).

Create a suitable testbench to test the circuit on relevant values.

Activity #2: Sequential add-and-shift multiplier
In Lab. #4 you designed a combinational adder-based multiplier. Design a **sequential add-and-shift multiplier** for 2 operands A_{in} and B_{in} on 8 bits. Operands are unsigned.

The multiplication of two 4-bit numbers is illustrated in the following figure:

					a_3	a_2	a_1	a_0	multiplicand
×					b_3	b_2	b_1	b_0	multiplier
					a_3b_0	a_2b_0	a_1b_0	a_0b_0	
				a_3b_1	a_2b_1	a_1b_1	a_0b_1		
			a_3b_2	a_2b_2	a_1b_2	a_0b_2			
+		a_3b_3	a_2b_3	a_1b_3	a_0b_3				
	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	product

The algorithm works in 4 steps:
1. Load A_{in} and B_{in} in A and B. Any change in A_{in} or B_{in} in the future will not affect the current computation
2. Multiply each digit of the multiplier (b_3, b_2, b_1 and b_0) by the multiplicand A. This corresponds to logically AND-ing b_i and the digits of A
3. Shift $b_i \cdot A$ to the left by i positions
4. Add column by column the shifted $b_i \cdot A$ terms to obtain the final product.

Remarks: let P be the partial product, initially set to 0. As b_i is either 0 or 1, this entails that A shifted to the left by i positions doesn't contribute to the partial sum (b_i is 0) or that it contributes (b_i is 1). As in hardware it is expensive to do indexing (i.e., accessing the n-th bit) and general shifting (i.e., shifting by n positions), "intelligently" shift A and B one position **in each iteration** according to the following pseudo-code:

```
load(A, Ain);
load(B, Bin);
N = 8;
P = 0;
while (N != 0){
  if (b[0] == 1)
    P = P + A;
  A = A << 1;
  B = B >> 1;
  N--;
}
res = P;
```

Design a FSM-D that implements the above pseudo-code. The FSM-D has the 8-bit *A* and *B* signals as inputs, as well as a *start* signal. Its outputs are the 16-bit *res* signal and the *ready* signal.

Steps:

1. Draw the HLSM state diagram and write the VHDL code to capture the HLSM behavior (cfr. VHDL: Embedded Systems RTL Design: capture HLSM behavior)
2. Design a FSM-D:
    a. creating 2 processes for behavioural datapath description from the HLSM state diagram (cfr. VHDL: Embedded Systems RTL Design: Describing a datapath behaviourally)
    b. create 2 processes for behavioural controller description from the HLSM state diagram (cfr. VHDL: Embedded Systems RTL Design: Describing the controller behaviourally)
    c. connecting datapath processes to controller processes to create a single architecture (cfr. VHDL: Embedded Systems RTL Design: Connecting controller and datapath).

Create a suitable testbench to test the circuit on relevant values.