# Exercise 4: Sequential Logic

Prof. Dr. Sven Simon

## 1 Unintended latches

Given the following processes (in `procs.vhd`):

```
proc1: process (A, B, SEL) is
begin  -- process proc1
  Z <= B;
  if (SEL = '1') then
    Z <= A;
  end if;
end process proc1;

proc2: process is
begin  -- process proc2
  Z <= B;
  if (SEL = '1') then
    Z <= A;
  end if;
end process proc2;

proc3: process is
begin  -- process proc3
  -- Z <= B;
  if (SEL = '1') then
    Z <= A;
  end if;
end process proc3;

proc4: process is
begin  -- process proc4
  if (SEL = '1') then
    Z <= A;
  end if;
  Z <= B;
end process proc4;
```

- Draw the circuit you expect to be synthesized out of the process. Any latches?
- Synthesize the processes and check your circuits against the ones synthesized by the tool.
- Which signals should be there in the sensitivity list for the behavior of the simulated process to match that of the circuit?
- What happens if you add unnecessary signals to the sensitivity list. Does that affect synthesis? Does that affect simulation? Is it a good practice?
- Run the testbench (`procs_tb.vhd`) and observe the waveforms. Do the same for every process.

## 2 Signals VS. variables

Given the following processes (`sig_var.vhd`):

```
proc1: process (clk) is
begin  -- process proc1
  if (clk'event and clk = '1') then      -- rising clock edge
    tmp <= a and b;
    q <= tmp;
  end if;
end process proc1;

proc2: process (clk) is
begin  -- process proc2
  if (clk'event and clk = '1') then      -- rising clock edge
    q <= tmp;
    tmp <= a and b;
  end if;
end process proc2;

proc3: process (clk) is
  variable tmp : std_logic;
begin  -- process proc3
  if (clk'event and clk = '1') then      -- rising clock edge
    tmp := a and b;
    q <= tmp;
  end if;
end process proc3;

proc4: process (clk) is
  variable tmp : std_logic;
begin  -- process proc4
  if (clk'event and clk = '1') then      -- rising clock edge
    q <= tmp;
    tmp := a and b;
  end if;
end process proc4;
```

- Draw the circuit you expect to be synthesized out of each process.
- Synthesize the processes and check your circuits against the ones synthesized by the tool.
- Run the testbench (`sig_var_tb.vhd`) and observe the waveforms. Do the same for every process.

## 3 Summation operator

- Define the entity and implement the architecture of a summation operator. It adds up eight numbers (each of width 8 bits) in a binary tree-like structure and outputs the sum . In order to ensure that the result of summation is correct let the wordlegnths grow after every partial sum operation to avoid overflows.
- Make your architecture fully pipelined by adding registers after every partial sum.