LINKÖPING UNIVERSITY
Department of Electrical Engineering

TSIU03, SYSTEM DESIGN

LECTURE 6

Kent Palmkvist
Kent.Palmkvist@liu.se

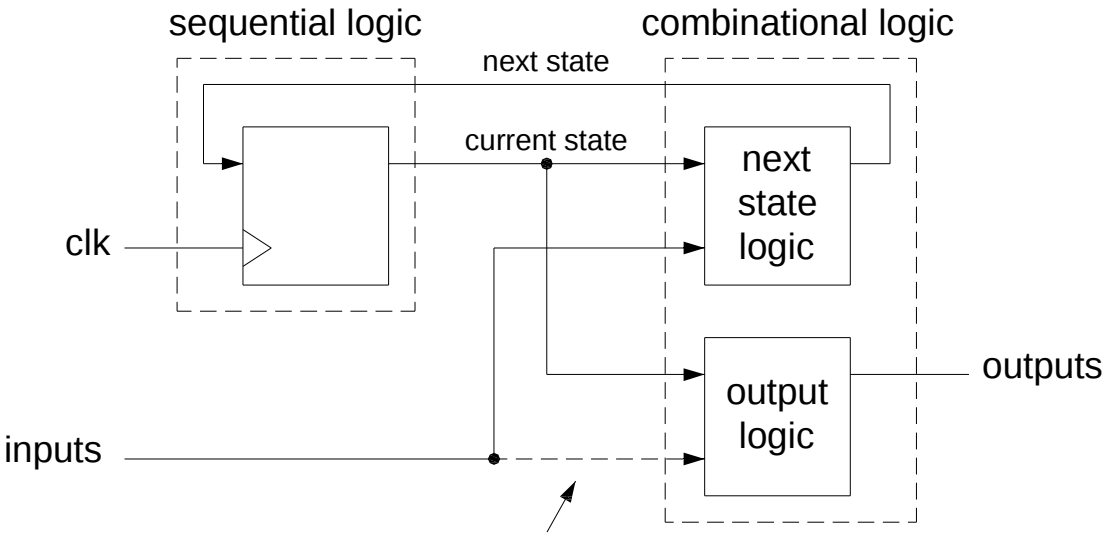
Slides by: Mario Garrido Gálvez (mario.garrido.galvez@liu.se)

Linköping, 2021

TODAY

- State Machines.
- Pipelining.
- Latency.
- Hierarchy.

FINITE-STATE MACHINES



- Moore: The output is obtained from the current state => it only changes when the clock changes.
- Mealy: The output is obtained from the current state and from the inputs=> it may change at any time due to a change in the inputs.

FINITE-STATE MACHINES

- A “finite-state machine” or just “state machine” is a circuit that evolves through different states.
- It can only be at one state at a time.
- Every clock cycle the state is updated. The transitions among states and the output values depend on the current state and on the input values.
- It combines sequential and combinational logic. The sequential logic consists in one or several registers that store the current state. The combinational logic is used to calculate the next state and the outputs.

- Ref: [A4.3.1]
- Example: Given a sequence of bits, how can we detect all the instances of the pattern 101? [R12.2]

HOW TO DESIGN A STATE MACHINE

1) Define the states of the state machine:

- Moore: One state for each combination of internal variables.
- Mealy: One state for each combination of internal variables and input values.

2) Draw the state diagram.

3) Draw the transition table. The transition table is a truth table with the input signals and current states as inputs. From them we calculate the values for the next state and for the output signals.

4) Obtain the logic function of each of the variables that define the next state as a function of the inputs and current state. Obtain the logic function of each output as a function of the current state (and also the inputs in case of Mealy).

5) Implement the circuit.

EXAMPLE: DETECT “101”

- Design a circuit that detects a sequence “101”.
- Example from [R12.2].

STATE MACHINES IN VHDL

```
architecture arch of pattern_detector is
  type stateFSM is (start, found1, found0, detect);
  signal state, next_state: stateFSM;
begin
-- sequential part: updates the stage at each clock cycle.
process (reset, clk)
begin
  if reset = '1' then
    state <= start;
  elsif rising_edge (clk) then
    state <= next_state;
  end if;
end process;
```



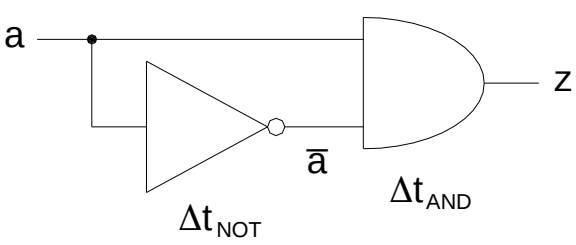
```
-- combinational part: calculates the next state and the output.
process (state,data_in)
begin
  case state is
    when start =>
      found <= '0';
      if data_in = '1' then
        next_state <= found1;
      else
        next_state <= start;
      end if;
    when found1 =>
      found <= '0';
       if data_in = '1' then
        next_state <= found1;
      else
        next_state <= found0;
      end if;
    when found0 =>
      found <= '0';
      if data_in = '1' then
        next_state <= detect;
      else
        next_state <= start;
      end if;
    when detect =>
      found <= '1';
       if data_in = '1' then
        next_state <= found1;
      else
        next_state <= found0;
      end if;
  end case;
end process;
end arch;
```

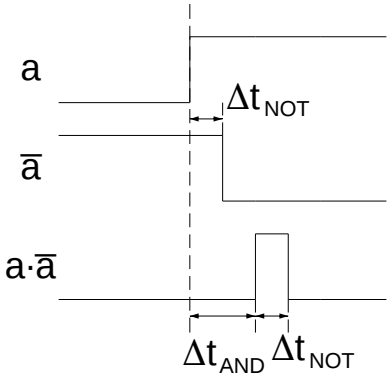
STATE MACHINES IN VHDL

- To make the state machine understandable, separate it into a combinational process and a sequential one.
- Define explicitly all the cases that may cause a transition. If a certain combination of input values is not explicitly defined but actually happens at the input, the state machine may jump to an unexpected state.
- At each state, define the output value of every output signal. If the output value of a certain output signal is not defined, the output signal will keep the value that it had.

DELAY & CRITICAL PATH

- Propagation delay: time that a combinational circuit takes to calculate the output since the time in which the inputs are stable.
- Glitch: undesired transition in a signal that occurs before the signal settles to its intended value.
- Critical path: path with the highest propagation delay.





PIPELINING

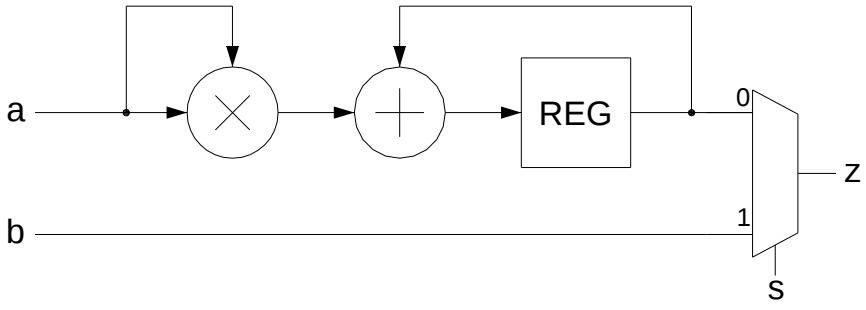
- Pipelining is a technique that consists of inserting additional registers in one or several parallel paths of the circuit.
- The main uses of pipelining are:
  - Equalize the delay of different paths to assure that data arrive at the same clock cycle.
  - Increase the clock frequency by reducing the critical path.
- Example: Design a circuit to calculate:

$$z = \begin{cases} \sum (a[n])^2 & if \quad s = 0 \\ b[n] & if \quad s = 1 \end{cases}$$

PIPELINING: EXAMPLE

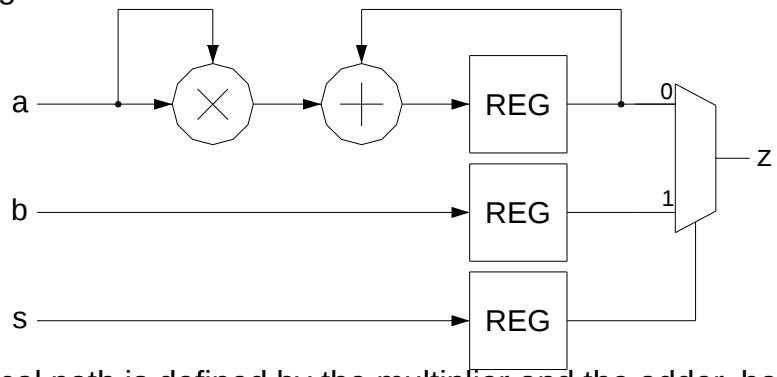
- Does this circuit calculate the function? Why?

$$z = \begin{cases} \sum (a[n])^2 & if \quad s = 0 \\ b[n] & if \quad s = 1 \end{cases}$$



PIPELINING: EQUALIZE DELAYS

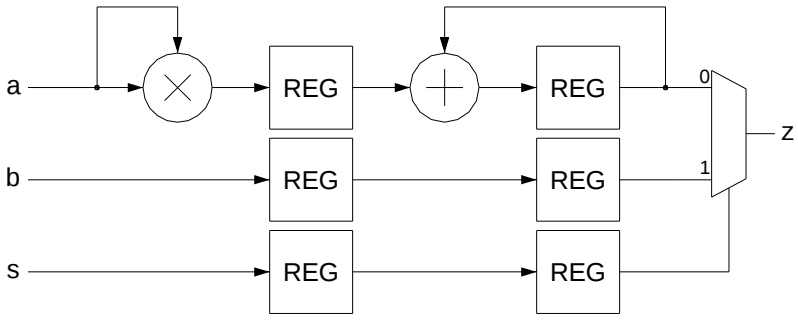
- Here pipelined is used to equalize the delays of all the paths. Now the signals are synchronized when they arrive to the multiplexer.
- Note that pipelining must be applied to all the paths, including the control signal!!



- If the critical path is defined by the multiplier and the adder, how can we now reduce and, thus, increase the clock frequency?

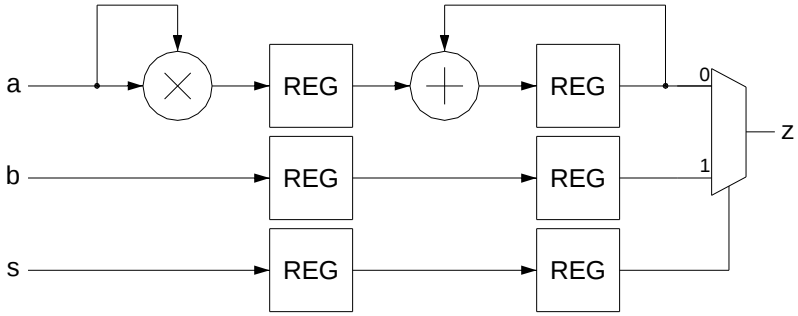
PIPELINING: REDUCE CRITICAL PATH

- Here pipelining has been used to reduce the critical path and, thus, increase the clock frequency.
- The critical path is reduced by introducing a register between the multiplier and the adder. Now the critical path will be the longest among the delays of the adder and the multiplier, but not the sum of both.
- The maximum clock frequency of a system is the inverse of the delay in the critical path (the longest delay between any two registers).



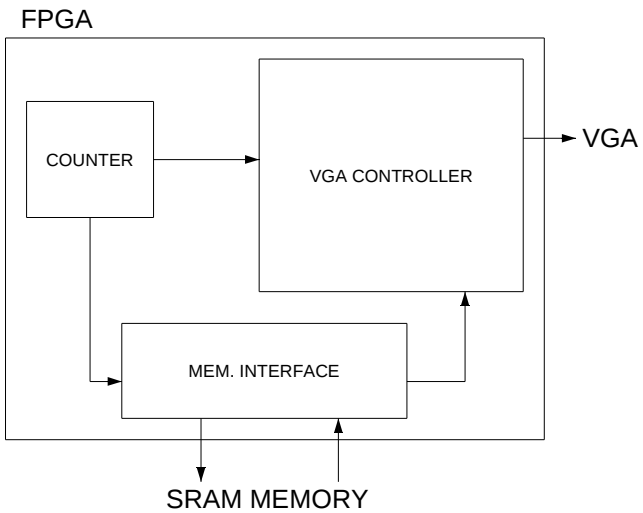
LATENCY: INPUT TO OUTPUT DELAY

- Latency: How long time (in seconds or in clock cycles) will it take for an input value to affect the output value?
- Adding a register is in the path adds delay (one more clock cycle)
- Example have two clock cycle latency (a specific input value will affect the output two clock cycles after entering).
- Latency different from throughput (new inputs per second)
- Latency different from time between input values (clock frequency)



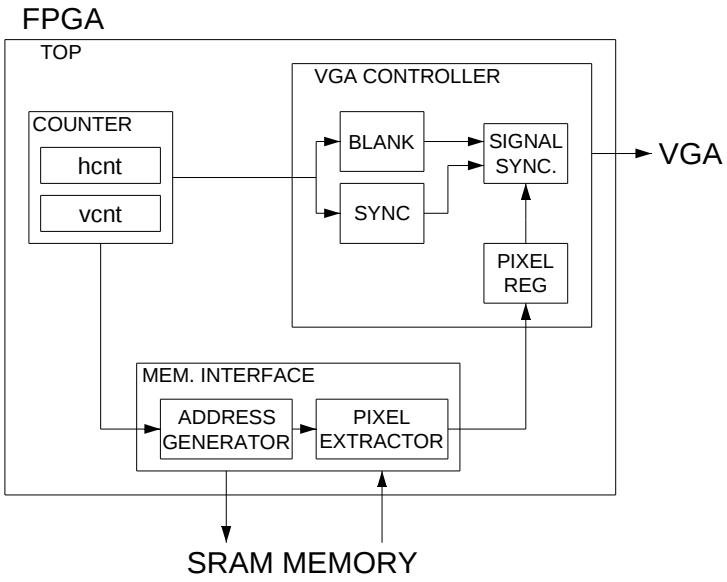
HIERARCHY

- A hardware system can be described as a set of modules that are interconnected.



HIERARCHY

- Each module can be divided in submodules that are interconnected. This creates a hierarchy in the design.



HIERARCHY IN VHDL

- In VHDL we can include other .vhd files as submodules in the design.
- Declaration:
  - The submodules are declared inside the **architecture** in the declaration part (before **begin**).
  - We use the reserved word **component** to declare submodules.
  - The declaration shows the **entity** of the submodule.
- Instantiation:
  - The instantiation of the submodules is done inside the architecture after **begin**.
  - We use **port map** for the instantiation.
  - The instantiation shows where the ports of the component are connected.

HIERARCHY IN VHDL: EXAMPLE

```
architecture arch of top is
  signal x, y, xy : signed;
  ...
  component multiplier   -- Declaration
  port (a, b: in signed (7 downto 0);
        z: out signed(7 downto 0);
  end component;
begin
  mult1: multiplier      -- Instantiation
    port map(a => x,     -- The line ends with comma.
         b => y,
         z => xy);
end arch;
```


MULTIPLE INSTANCES

- We can define multiple instances of the same component.
- Example: If we want to design a filter that includes 2 multipliers, we can declare the component once and then create 2 instances of it:

```
mult1: multiplier     -- Instantiation 1
  port map( a => x1, b => y1, z => xy1);
mult2: multiplier     -- Instantiation 2
  port map( a => x2, b => y2, z => xy2);
```

- Note that each multiplier is connected to different signals.
- Ports of different instances can be interconnected by connecting them to the same signal.
- We always have to respect the input/output nature of the ports.

DIRECT INSTANTIATION

- We can also instantiate the submodules directly without declaring them.
- Instantiation using indirect instantiation (as shown before), when we have already declared the component:

```
mult1: multiplier
   port map( a => x1, b => y1, z => xy1);
```

- Instantiation using direct instantiation, without having declared the component:

```
mult1: entity work.multiplier
  port map( a => x1, b => y1, z => xy1);
```

- Note that we have to specify the library (in this case “work”) where the component is included.

CHECKLIST FOR LECTURE 6

- State machines: design, description in VHDL.
- Propagation delay, glitch, critical path, pipelining.
- Hierarchy: representation as blocks, direct and indirect instantiation.

AT HOME

- Review the checklist for lecture 6 and check that you understand all the concepts and you know how to use them.
- Prepare Lab 3.
- Work on handin