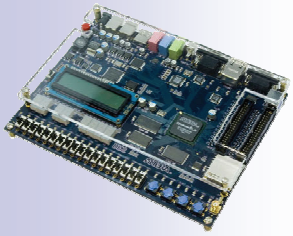
Struktury počítačových systémů



Přednáší: Richard Šusta

3. listopadu 2011 verze 1.0

ČVUT-FEL v Praze – kód předmětu A0B35SPS

Vaše úlohy - menší kritika s ukázkami

- Stopky - LE děs a hrůza vytvořená jako jeden výpočetní proces
  - Total logic elements : 4,390 / 33,216 (13 %)
  - Dedicated logic registers : 106 / 33,216 (< 1 %)
  - doba překladu cca 10 minut!!!

- Stopky - docela optimální,
  - sice psané také jako jeden proces, ale RTL stylem,
  tj. Register Transfer Level konstrukcemi
  - Total logic elements : 108 / 33,216 (< 1 %)
  - Total combinational functions : 108 / 33,216 (< 1 %)
  - Dedicated logic registers : 44 / 33,216 (< 1 %)
  - překlad - za cca 10 vteřin

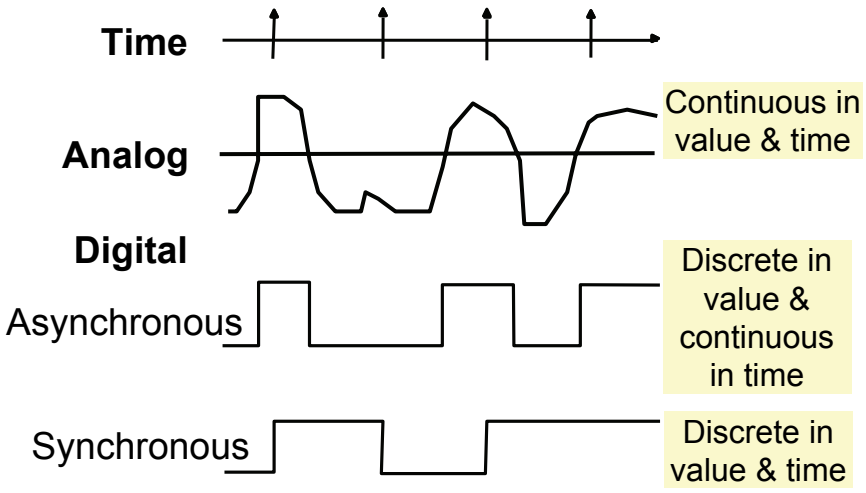
Synchronous/Asynchronous

Terminology
Synchronous/Asynchronous

- *Synchronous - happening, existing, or arising at the same time, from Greek syn-chronos*
- Two or more signals can be mutually synchronous or asynchronous
- An input of a circuit is referred to as a **synchronous input** if they have effect on the outputs only synchronously with the clock signal
- In contrast, "**asynchronous inputs**" of circuits influence outputs immediately.

Question: Is the clock input of flip-flop a synchronous or asynchronous input?

Signal Examples Over Time

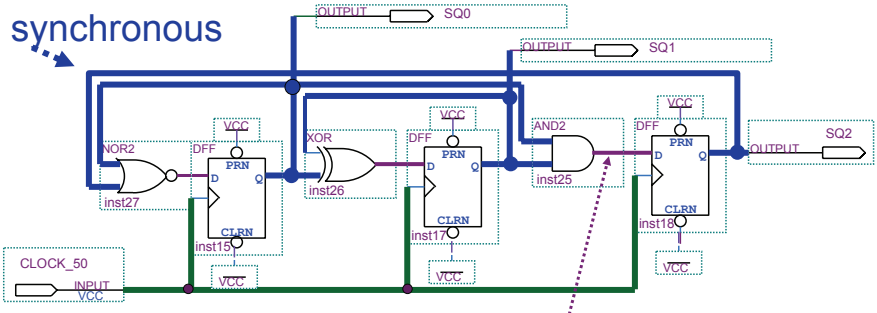


Counter 5 / Divider 5

The counter is referred as synchronous

All outputs change synchronously with clock

synchronous

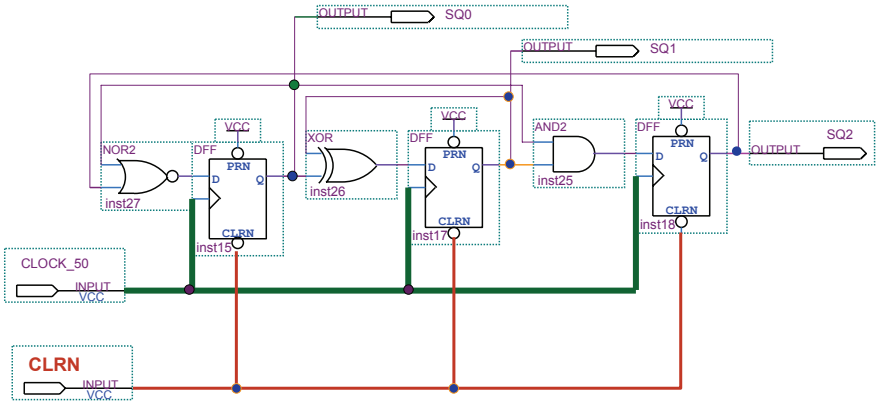


The clock inputs of all flip-flops are connected together

synchronous signal, but with possible hazards

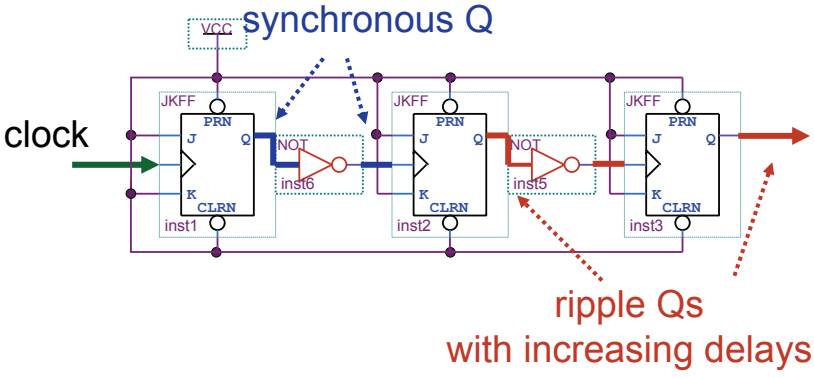
Counter 5 / Divider 5

The counter is referred as
synchronous with asynchronous clear



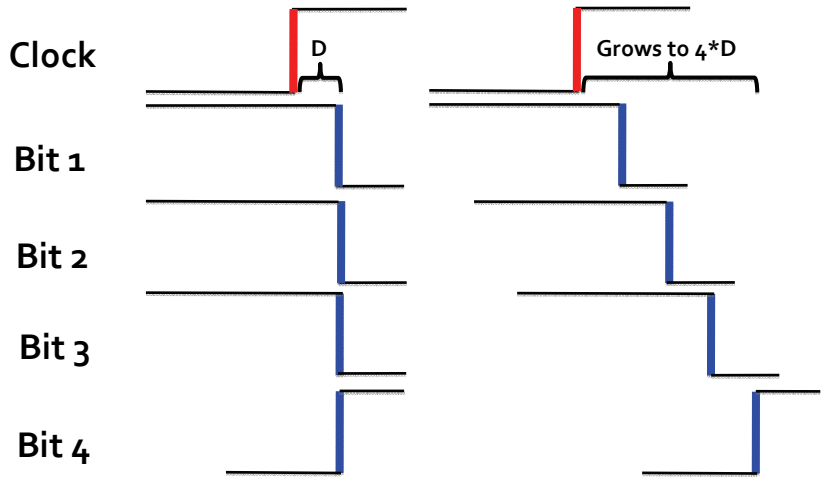
Asynchronous counter

- The counter is referred as asynchronous counter



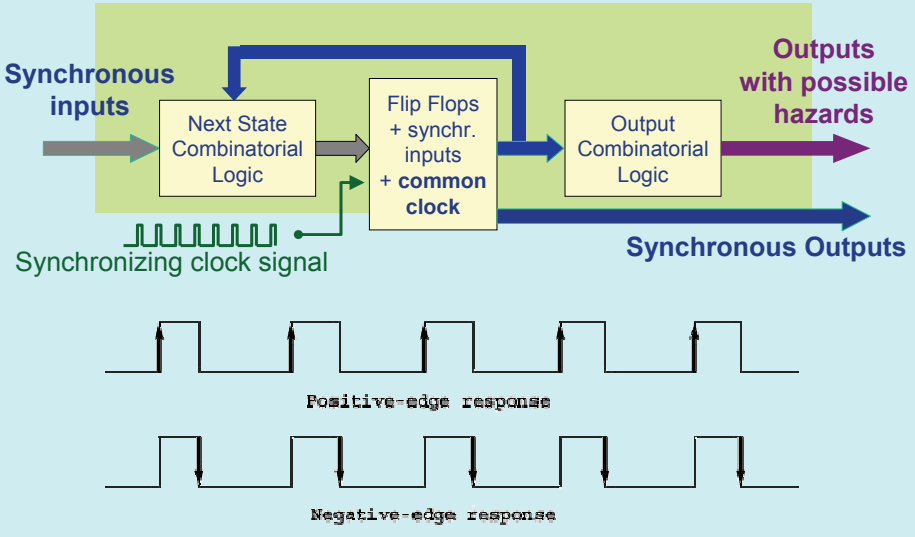
Synchronous inputs cannot be implemented. Why?

Synchronous versus asynchronous counter

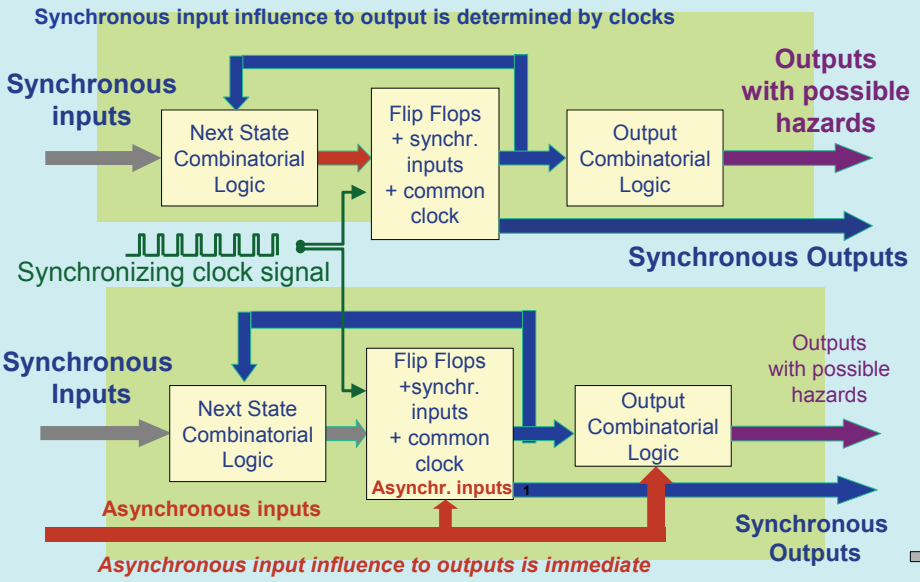


Synchronous / Asynchronous

Synchronous input influence to output is determined by clocks



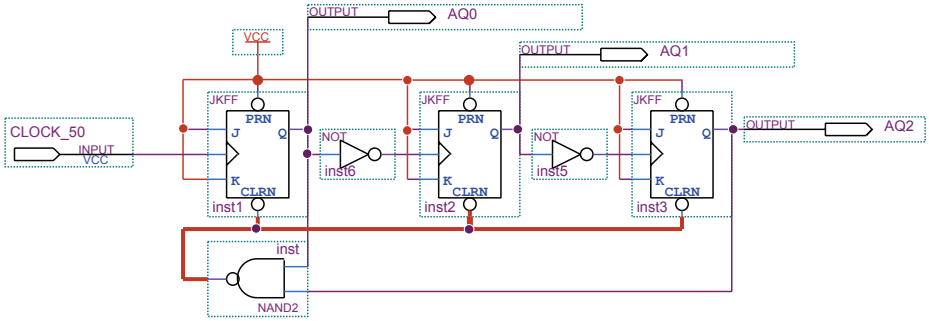
Synchronous / Asynchronous



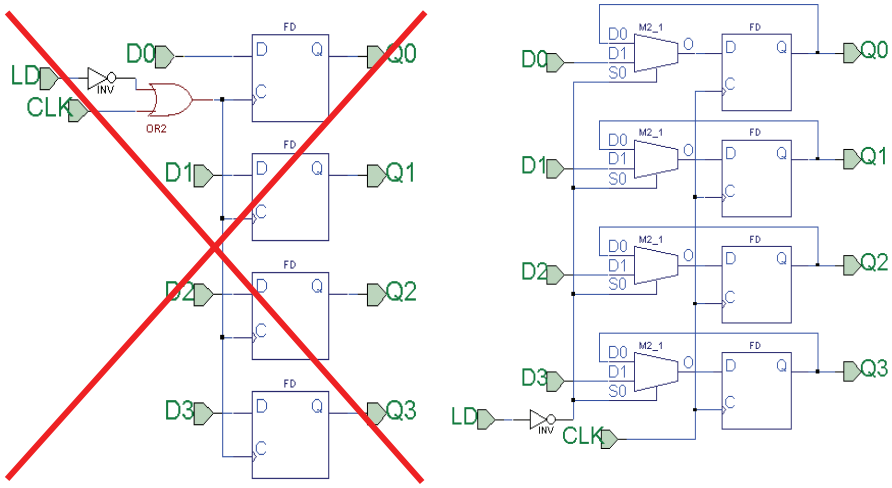
Inputs

- Precondition: *Combinational circuits can produce unwanted outputs called hazards, see lecture 3.*
- Rule: *All inputs effecting outputs immediately require only our desired signals.*
- Conclusion: ***Outputs of combinational circuits must not be connected to asynchronous inputs or clock inputs.***
- Exception: *Well known, deeply analyzed cases…*

Asynchronous counter is well known, deeply analyzed case



Simple Parallel Load Register



VHDL Specification for Register

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

 entity reg4 is
 port (clock, load : in std_logic;
       data : in std_logic_vector(3 downto 0);
       q4 : out std_logic_vector(3 downto 0));
 end reg4;

architecture reg4_arch of reg4 is
begin process(clock)
 begin
   if clock = '1' and clock'event then
     if (load = '1') then q4 <= data;
      end if;
   end if;
  end process;
end reg4_arch;
```

Questions:

- Add synchronous clear input
- Add asynchronous clear input

Universal Counter

Up/Down/Clear/Preload

Remeber: std_logic_arith - numeric_std

UNSIGNED ADDER WITH NO CARRYOUT

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_arith.all; -- not needed but keep for style
use IEEE.std_logic_unsigned.all;
entity adder is
  port(
    a : in STD_LOGIC_VECTOR(2 downto 0);
    b : in STD_LOGIC_VECTOR(2 downto 0);
    c : out STD_LOGIC_VECTOR(2 downto 0)
     );
end adder;
architecture adder_arch of adder is
begin
 c <= a + b;
end adder_arch;
```

Tells compiler to treat std_logic_vector like unsigned type

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.numeric_std.all;

entity adder is
  port(
    a : in STD_LOGIC_VECTOR(2 downto 0);
    b : in STD_LOGIC_VECTOR(2 downto 0);
   c : out STD_LOGIC_VECTOR(2 downto 0)   );
end adder;

architecture adder_arch of adder is
begin
 c <= std_logic_vector(
          unsigned(a) + unsigned(b)
                    );
end adder_arch;
```


Up-down Counter with Parallel Load

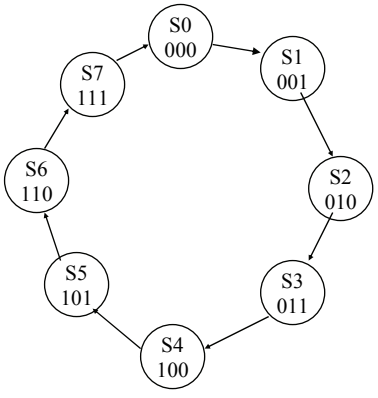
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Counter4 is
  port( Preset: in std_logic_vector(3 downto 0);
        Clock, Load, Reset, Up: in std_logic;
        Q: out std_logic_vector(3 downto 0) );
end Counter4;
architecture Behavioral of Counter4 is
  signal nn: std_logic_vector(3 downto 0);
begin
  process(Clock,Reset)
  begin
    if Reset='1' then  nn <= "0000";
    elsif ( Clock'event and Clock='0') then
        if Load='1' then nn <= Preset;
        elsif Up='0' then  nn <= nn + 1;
        else  nn <= nn - 1;
        end if;
    end if;
  end process;
  Q <= nn;
end Behavioral;
```

Finite State Machines

Automata

Automaton, pl. Automata
– odborná angličtina užívá latinské skloňování

Binary Counter State Diagram



VHDL State Model of Counter

```
library ieee; use ieee.std_logic_1164.all;
entity BIN_COUNTER is
port(CLK: in  std_logic;   Z: out
  std_logic_vector(2 downto 0));
end BIN_COUNTER;
architecture ALGORITHM of
  BIN_COUNTER is
begin
  --- see the next pages
end ALGORITHM;
```


VHDL State Model of Counter

```
architecture ALGORITHM of BIN_COUNTER is
begin
  process(CLK)
  variable PRESENT_STATE: std_logic_vector(2 downto 0) := "111";
  begin
     if (CLK'event and CLK = '1') then
     case PRESENT_STATE is
           when "000"  => PRESENT_STATE := "001";
           when "001"  => PRESENT_STATE := "010";
           when "010"  => PRESENT_STATE := "011";
           when "011"  => PRESENT_STATE := "100";
           when "100"  => PRESENT_STATE := "101";
           when "101"  => PRESENT_STATE := "110";
           when "110"  => PRESENT_STATE := "111";
           when "111"  => PRESENT_STATE := "000";
      end case;
     end if;
     Z <= PRESENT_STATE;
end process;
end ALGORITHM;
```

VHDL State Model of Counter

```
architecture ALGORITHM of BIN_COUNTER is
begin
  process(CLK)
  variable PRESENT_STATE: std_logic_vector(2 downto 0) := "111";
  begin
     if (CLK'event and CLK = '1') then
     case PRESENT_STATE is
           when "000"  => PRESENT_STATE := "001";
           when "001"  => PRESENT_STATE := "010";
           when "010"  => PRESENT_STATE := "011";
           when "011"  => PRESENT_STATE := "100";
           when "100"  => PRESENT_STATE := "101";
           when "101"  => PRESENT_STATE := "110";
           when "110"  => PRESENT_STATE := "111";
           when "111"  => PRESENT_STATE := "000";
      end case;
     Z <= PRESENT_STATE;
     end if;
    -- Z <= PRESENT_STATE;
end process;
end ALGORITHM;
```

6 LE

3 LE

VHDL State Model of Counter

```
process
  variable PRESENT_STATE: std_logic_vector(2 downto 0) := "111";
  begin
     case PRESENT_STATE is
           when "000"  => PRESENT_STATE := "001";
           when "001"  => PRESENT_STATE := "010";
           when "010"  => PRESENT_STATE := "011";
           when "011"  => PRESENT_STATE := "100";
           when "100"  => PRESENT_STATE := "101";
           when "101"  => PRESENT_STATE := "110";
           when "110"  => PRESENT_STATE := "111";
           when "111"  => PRESENT_STATE := "000";
      end case;
   Z <= PRESENT_STATE;
   -- funguje i na wait until(CLK='1'), ale nedoporuceno
   wait until (CLK'event and CLK = '1');
   end process;
end ALGORITHM;
```

Dodatek k přednášce pro samostudium

Processes with Sensitivity Lists

- Processes can be defined with an explicit "sensitivity list"
  - Sensitivity list is a list of signals that is monitored for changes
  - Sensitive processes are activated when any of the sensitivity list signals change state
  - A sensitivity list process ***cannot*** have wait statements defined within the process
    - There is an implicit "WAIT ON" statement at the end of the process
    - Process evaluation is suspended at the end of process.

[Iowa: Intro. to Digital Design]

Dodatek k přednášce pro samostudium

Processes without Sensitivity Lists

- Processes can be defined without any sensitivity list
  - These processes **MUST** have at least one WAIT statement.
    - Some tools require WAIT to be the first statement after BEGIN.
    - Initial process evaluation runs until first WAIT is encountered.
  - The WAIT statement defines signals that are monitored for change.
  - Non-sensitive processes are activated when WAIT statement signals change state
  - Process suspends when next WAIT statement is encountered
    - Some tools allow multiple WAIT statements per process.

[Iowa: Intro. to Digital Design]

Dodatek k přednášce pro samostudium

Altera Quartus

[http://www.altera.co.uk/support/kdb/solutions/rd12152006_288.html]

Problem

What types of VHDL wait constructs does Quartus II synthesis support?

Solution

The Quartus® II software supports only a single VHDL wait-until statement in a process. Other VHDL wait constructs such as wait-for statements, or processes with more than one wait statement, are not synthesizable.

For example, Quartus II integrated synthesis supports the following wait-until syntax:

```
architecture dff_arch of ls_dff is
begin
   output: process begin
      wait until (CLK'event and CLK='1');
      Q <= D;
      Qbar <= not D;
   end process output;
end dff_arch;
```

The software does not support the following types of wait statements, and generates an error during synthesis:

```
process --Unsupported process declaration
begin
   CLK <= '0';
   wait for 20 ns;
   CLK <= '1';
   wait for 12 ns;
end process;
output: process begin  --Unsupported process declaration
   wait until (CLK'event and CLK='1');
   Q <= D;
   Qbar <= not D;

   wait until (CLK'event and CLK='0');
   Q <= 0;
   Qbar <= 1;
end process output;
```

State Machines...

Nonlinear

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x},\mathbf{u})$$
$$\mathbf{y} = \mathbf{h}(\mathbf{x},\mathbf{u})$$

$\mathbf{x}$ is the state vector (length n)
$\mathbf{u}$ is an input vector (length m)
$\mathbf{y}$ is the output vector (length p)

Linear

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

$\mathbf{A}$ is $n \times n$, $\mathbf{B}$ is $n \times m$
$\mathbf{C}$ is $p \times n$, $\mathbf{D}$ is $p \times m$

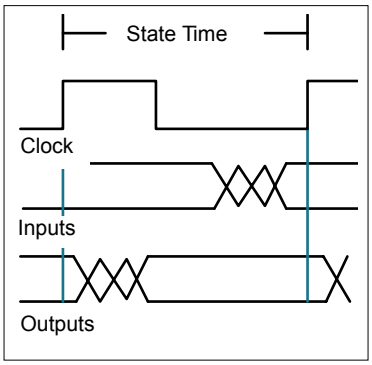
Discrete

$$\mathbf{x}[k+1] = \mathbf{f}(\mathbf{x}[k],\mathbf{u}[k])$$
$$\mathbf{y}[k] = \mathbf{h}(\mathbf{x}[k],\mathbf{u}[k])$$

Vectors are now at discrete time steps.

Concept of State Machine

Example: **Positive Edge Triggered Synchronous System**



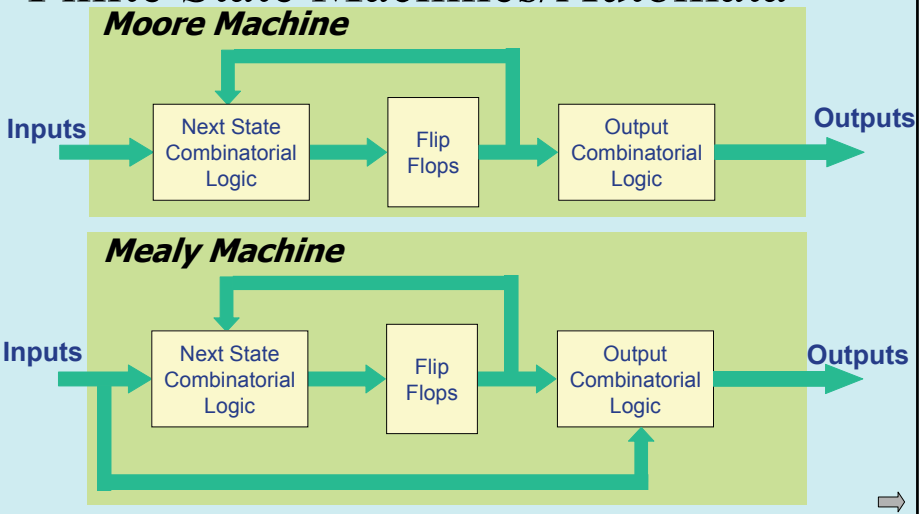
On rising edge, inputs sampled
outputs, next state computed

After propagation delay, outputs and
next state become stable

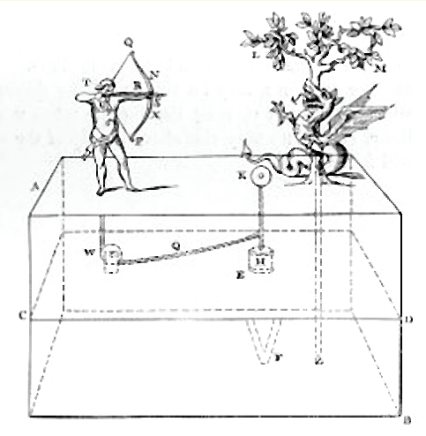
Immediate Outputs:
affect datapath immediately
could cause inputs from datapath to change

Delayed Outputs:
take effect on next clock edge
propagation delays must exceed hold times

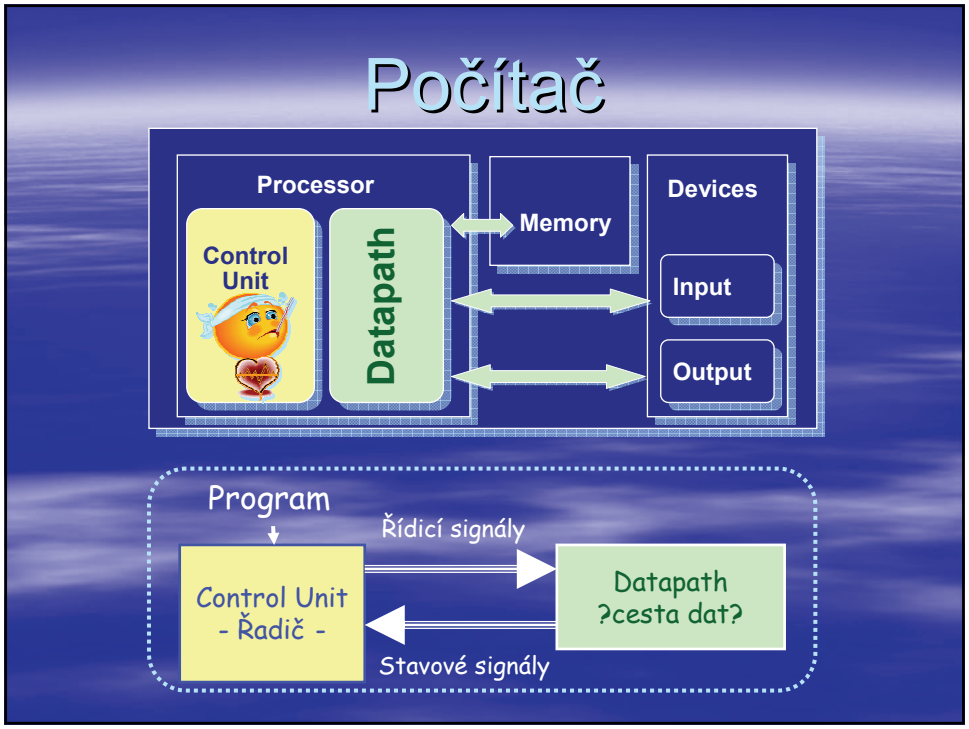
Mealy and Moore Finite State Machines/Automata



Řadiče

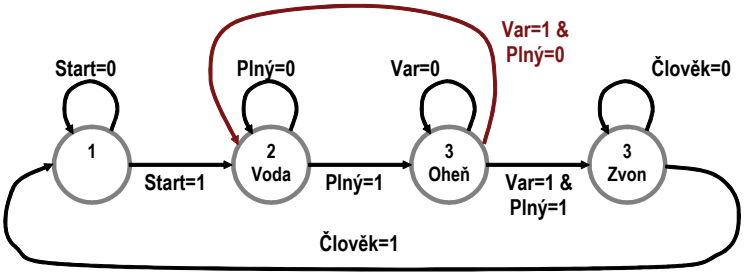


Starověký řadič: Herkules střílí na draka, Hero z Alexandrie, *Pneumatica*.



Formát zápisu podmínkového řadiče

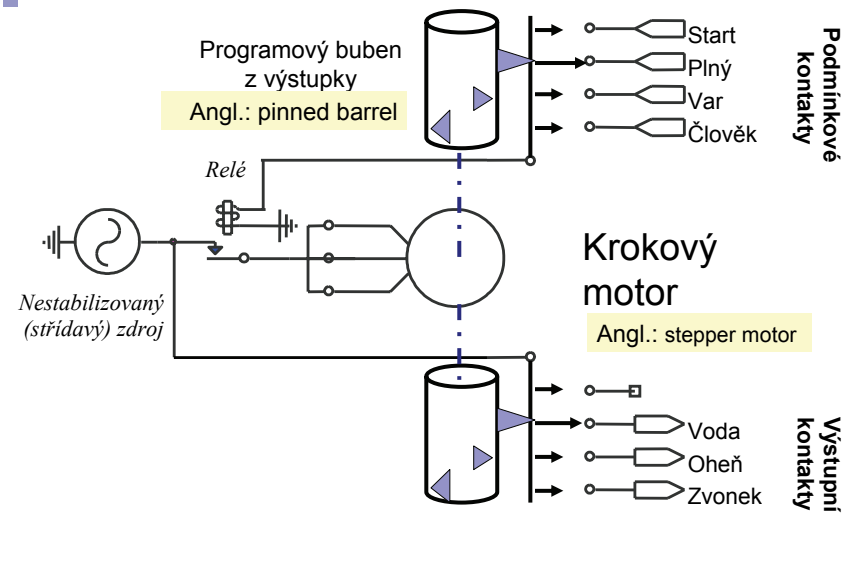
- Speciální případ binárního automatu Moore (Mealy)
  - nejčastější akce – přechod do následujícího stavu
    - např. řadič kotle



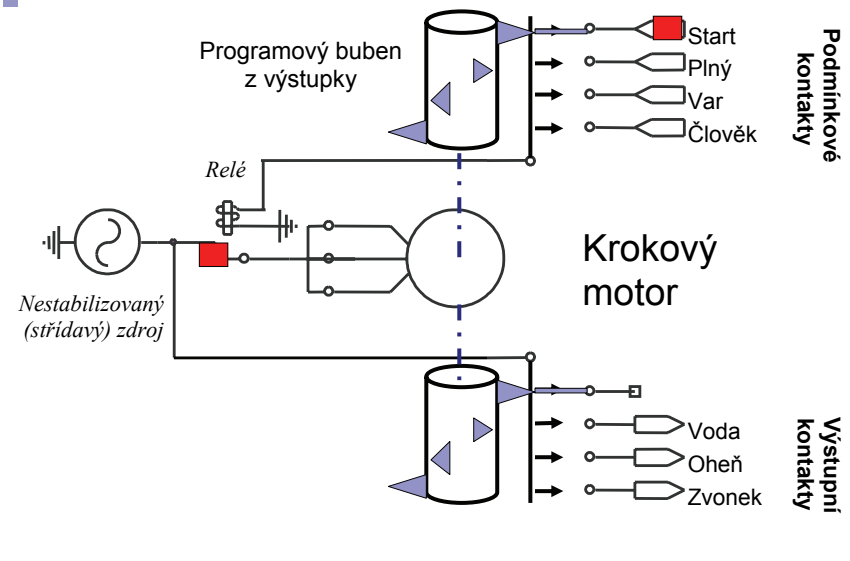
Řadič a jeho mechanická analogie

Hrací skřínka, Leopold Aucac Aine, Paris

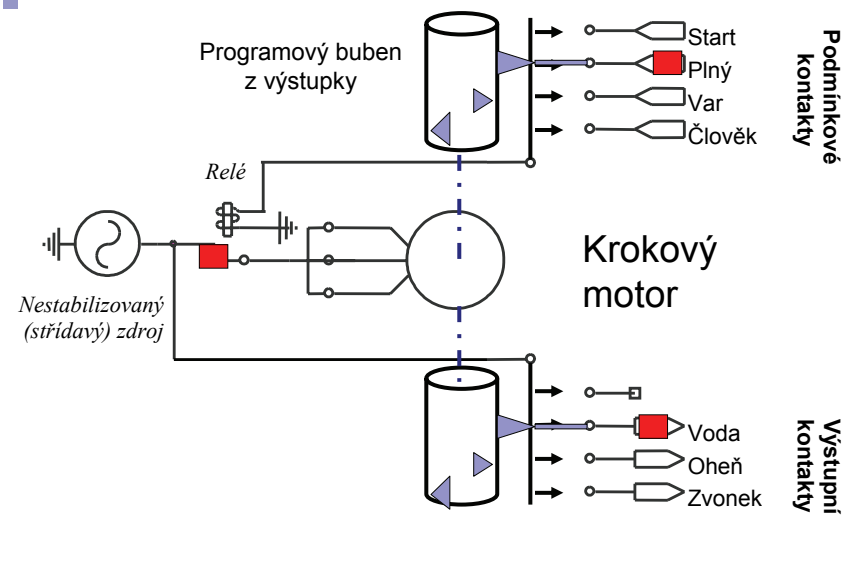
Elektromechanický automat



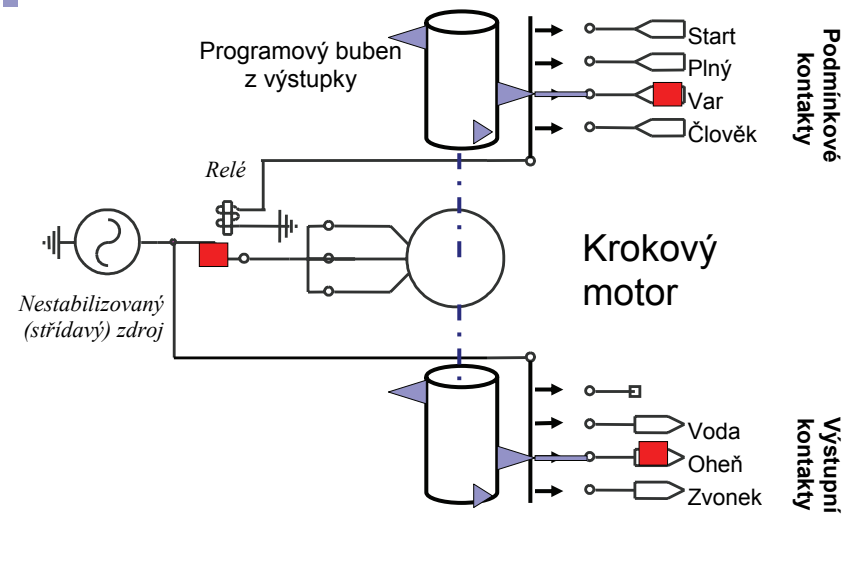
Animace: Elektromechanický automat 1/5



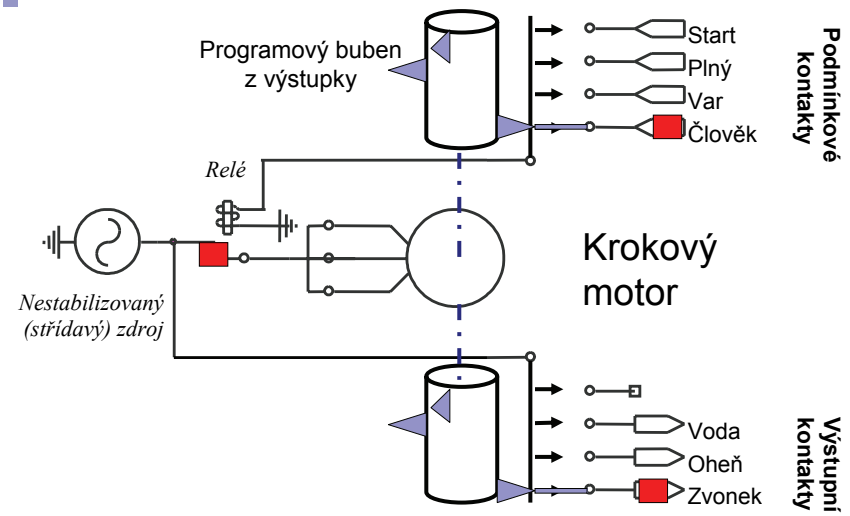
Animace: Elektromechanický automat 2/5



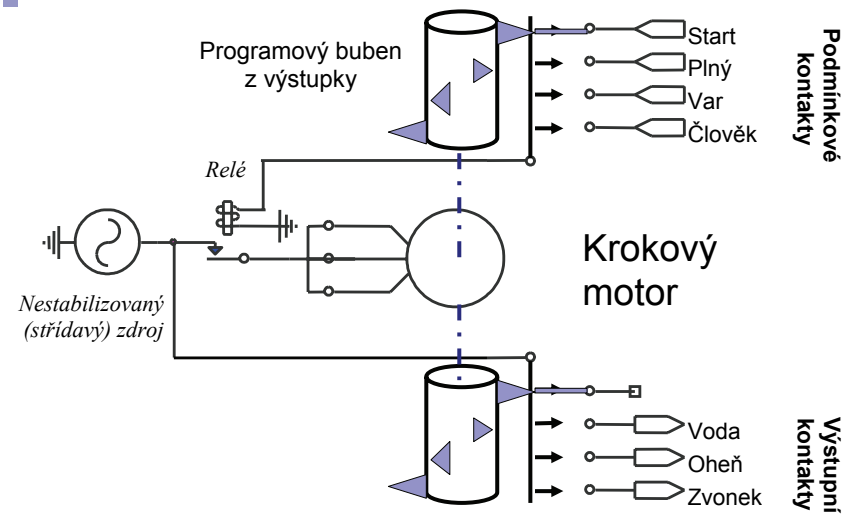
Animace: Elektromechanický automat 3/5



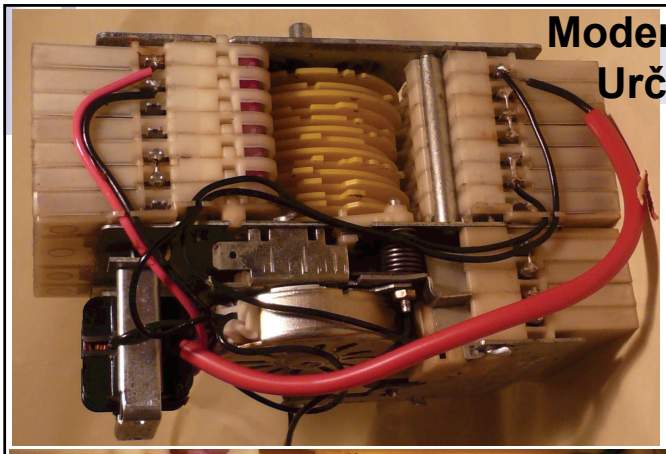
Animace: Elektromechanický automat 4/5

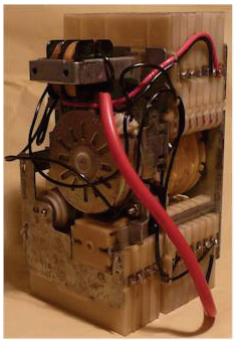


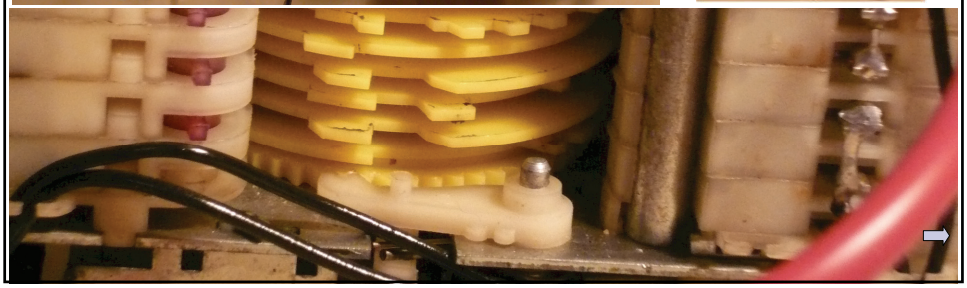
Animace: Elektromechanický automat 5/5



Moderní řadič kotle
Určitě ho znáte...

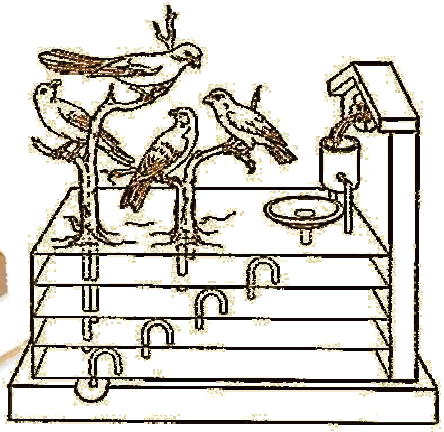






Také řadiče – ale bez podmínkové části

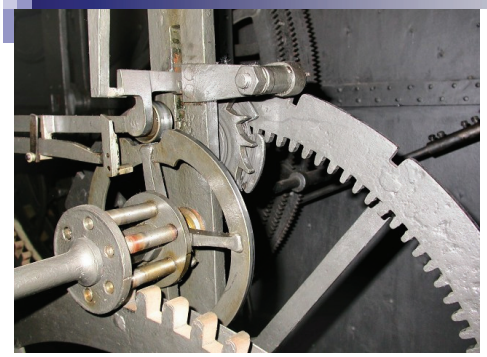
replika středověké hračky

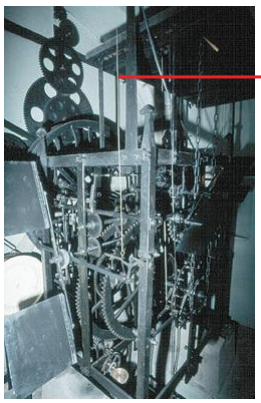


Heron of Alexandria, 1 století

Slavný automat 19. století

A co orloj?





Táhlo pro přenos informace od hodin

Program bicího stroje
- ozubená kola = paměť stavu

[Zdroj: Šíma, Z.: Astronomical clocks - HI-TECH of the 14th century, na webu lze najít pod: "Orloje - hi-tech 14. století" (1. a 2. část)]

Konečný automat a nekonečno...

- Konečné automaty mají své celočíselné limity....

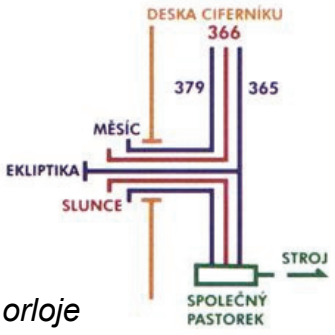


Schéma funkce 3 hlavních kol orloje

Schema funkce tří hlavních kol orloje.

Chyba pohybu Měsíce v závislosti na počtu zubů kol (tj. stavů), žel přesná hodnota je necelé číslo, a tak nelze popsat konečným automatem

Chyba pohybu Měsíce v závislosti na počtu zubů hlavních kol.

Řadič kotle řady 74

Potřebujeme jen

1 binární 4bitový čítač 74193

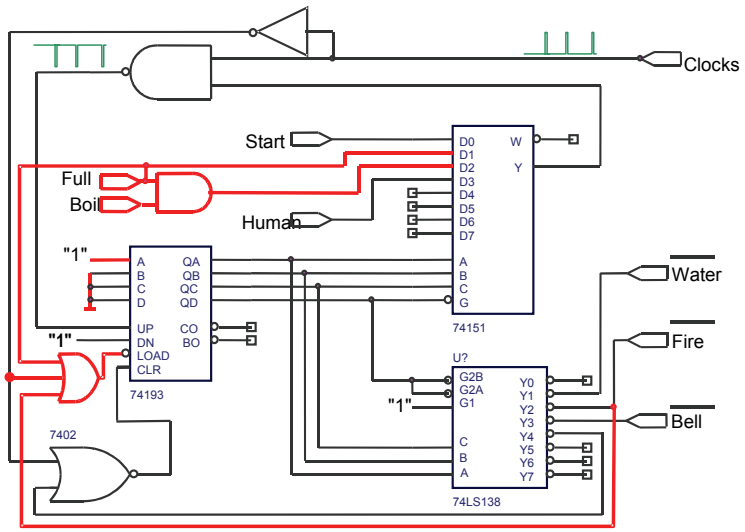
1 multiplexor 1 z 8: 74151

1 dekodér 8 z 1: 74138

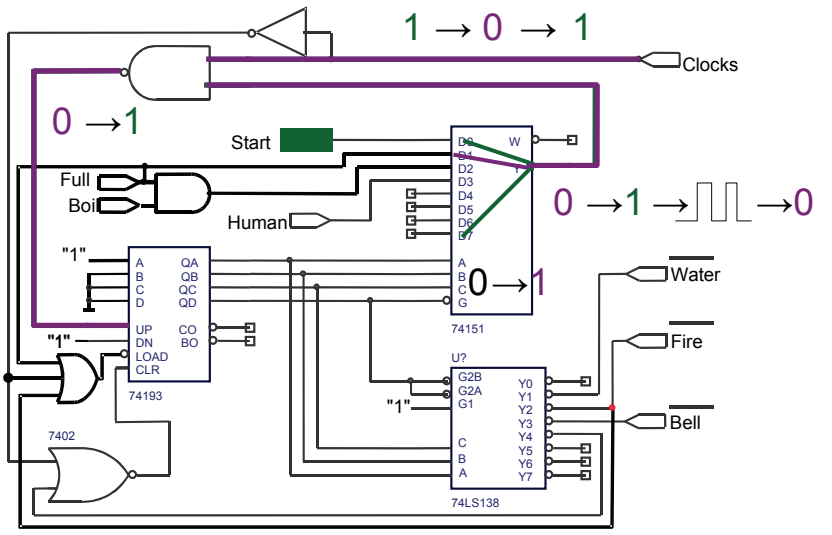
5 hradel

"a šťastnou hodinovou ruku"

Řadič kotle 74

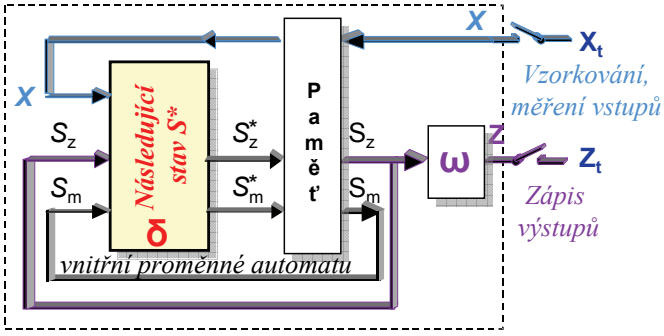


LSI řadič kotle - animace



Automat typu Moore

$M = \langle X, S = S_z \cup S_m, Z, \omega, \delta, S_0 \rangle$



Automaton/FSM in VDHL

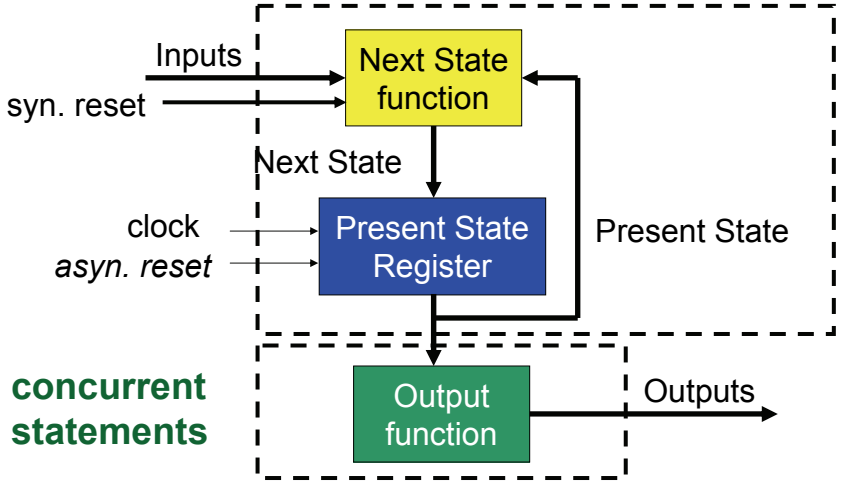
Based on Kris Gaj:
FSM Machines in VHDL and Their Testing,
George Mason 2004

FSMs in VHDL

- δ function - state transitions should be described in a process sensitive to *clock* and *asynchronous reset* signals only!
- Outputs of FSM should be described as concurrent statements outside the process!
- FSMs are preferable styles of process codes - they can be easily converted into logical elements!

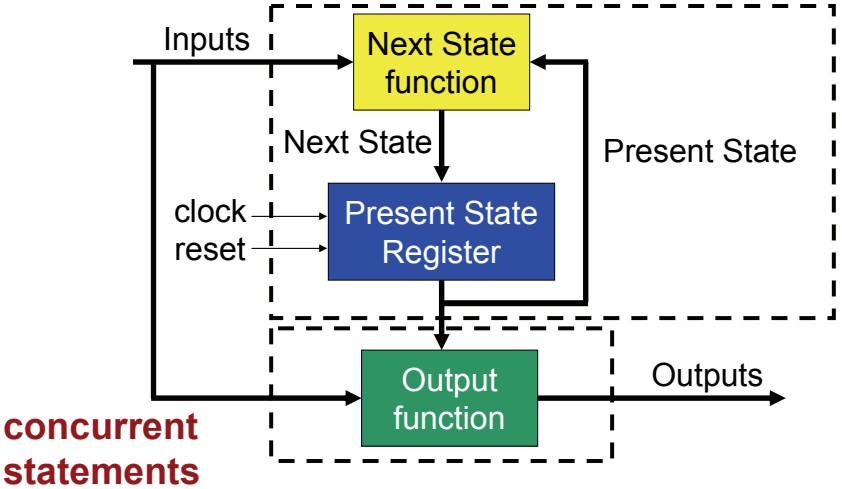
Moore FSM

process(clock, reset_asyn)



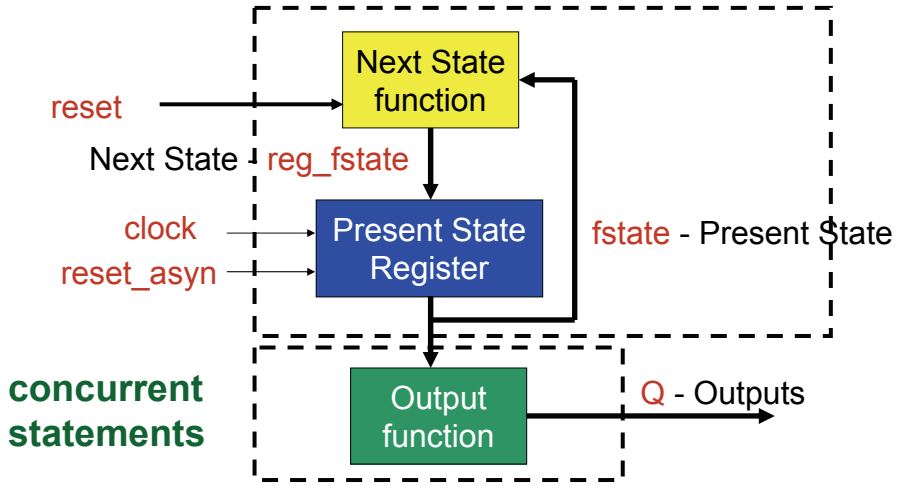
Mealy FSM

process(clock, reset_asyn)



Counter Moore FSM

process(clock, reset)



Counter

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
ENTITY citacFSM IS
   PORT (   reset : IN STD_LOGIC := '0';
            reset_asyn : IN STD_LOGIC := '0';
             clock : IN STD_LOGIC;
            Q : OUT std_logic_vector(2 downto 0)    );
END citacFSM;

ARCHITECTURE BEHAVIOR OF citacFSM IS
 SUBTYPE  type_fstate is STD_LOGIC_VECTOR(2 DOWNTO 0);
 CONSTANT state1: type_fstate := "000" ;   CONSTANT state2: type_fstate := "001" ;
 CONSTANT state3: type_fstate := "010" ;   CONSTANT state4: type_fstate := "011" ;
 CONSTANT state5: type_fstate := "100" ;   CONSTANT state6: type_fstate := "101" ;
 CONSTANT state7: type_fstate := "110" ;   CONSTANT state8: type_fstate := "111" ;
 SIGNAL fstate : type_fstate;
 SIGNAL reg_fstate : type_fstate;
```

```
BEGIN
PresentStateRegister :
   PROCESS (clock , reset_asyn)
   BEGIN ... END PROCESS;
NextStateFunction :
  PROCESS (fstate, reset)
  BEGIN ... END PROCESS;
OutputFunction :
  PROCESS(fstate)
   BEGIN ... END PROCESS;
END BEHAVIOR;
```


PresentStateRegister

```
PresentStateRegister :
  PROCESS (clock, reset_asyn)
   BEGIN
      IF reset_asyn='1' THEN fstate <= state1;
      ELSIF (clock='1' AND clock'event) THEN
         fstate <= reg_fstate;
      END IF;
   END PROCESS;
```

NextStateFunction

```
NextStateFunction :
PROCESS (fstate,reset)
 BEGIN IF (reset='1') THEN   reg_fstate <= state1;
             ELSE  CASE fstate IS
                      WHEN state1 => reg_fstate <= state2;
                      WHEN state2 => reg_fstate <= state3;
                      WHEN state3 => reg_fstate <= state4;
                      WHEN state4 => reg_fstate <= state5;
                      WHEN state5 => reg_fstate <= state6;
                      WHEN state6 => reg_fstate <= state7;
                      WHEN state7 => reg_fstate <= state8;
                      WHEN state8 => reg_fstate <= state1;
                      WHEN OTHERS =>
                         report "Reach undefined state"; -- error in design
                      END CASE;
          END IF;
  END PROCESS;
```


OutputFunction

```
OutputFunction :
  PROCESS(fstate)
   BEGIN
          q<=fstate;
  END PROCESS;
```