

This lecture gives a first introduction to finite-state machines as digital circuits, their components and how they can look like in VHDL. It forms the basis for upcoming lectures on state machine modeling and implementation.

HWMod WS24

FSM Basics Introduction FSM Circuits Example: Timer Hardware Modeling [VU] (191.011) - WS24 -

Finite-State Machine Basics

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2024/25

Modified: 2025-03-10, 16:07 (b25118c)

-Finite-State Machine Basics

Fundamental structure in digital designs

You should already have encountered finite-state machines – also called finite-state automata or simply state machines – in other courses, where they have probably been introduced as abstract mathematical models for computations. However, in this lecture, we will not focus on the theory of automata, but rather look into concrete circuit implementations of this concept. Finite-state machines – or FSMs in short – are a fundamental structure in digital design. Understanding them is crucial for developing efficient, reliable, and maintainable hardware, as they allow for the abstraction of complex processes into manageable sub-tasks and components.



Finite-State Machine Basics

Fundamental structure in digital designs
 An FSM consists of a
 set of inputs, outputs and states
 transition function

On a basic level, an FSM is defined by a set of inputs, outputs, and states, as well as a transition function that defines how the system moves from one state to another based on the current state and the input values. The outputs are set according to the current state and, in some instances, also the input values.



—Finite-State Machine Basics Introduction Introduction

Fundamental structure in digital designs
 An FSM consists of a
 set of inputs, outputs and states
 It transition function
 Synchronous FSMs: State changes **only** happen at clock events!

Since we are talking about digital circuits, the inputs and outputs are sets of binary signals. The states are encoded into binary values as well and the current state is stored in some sequential circuit element. Because we want to implement synchronous circuits, this element is implemented using flip-flops, which are clocked by a common clock signal to ensure that all state transitions occur synchronously to the clock. Thus, at each active clock edge, the values stored by the flip-flops that hold the state are updated according to the transition function.



-Finite-State Machine Basics

Fodomismetal structure in digital designs
 Art SFGE consistent at tables
 transition transition
 transition transition
 Synchronous FSME: State changes only happen at clock events!
 Important VHSE: concepts used in the locture
 Enternoism tradeting of contributional logic
 Sequential construmetions.

Note that this lecture, and the following ones on modeling and implementing FSMs, heavily rely on other topics we have already covered. In particular, we will use enumeration and record types, behavioral circuit modeling and sequential circuit elements. If you cannot recall these topics please refer back to the respective lectures before starting your journey into the world of finite-state machines, as these topics are crucial for understanding the upcoming content.



─Finite-State Machine Basics └─FSM Circuits └─State Machine as Digital Circuits



This slide shows the general structure of a circuit implementing a finite-state machine. It consists of three main components: the next-state logic, the state register and the output logic. Please note that the cloud shapes are used for purely combinational components. Over the next few slides we will explain each of the individual parts in detail.

State Machine as Digital Circuits HWMod **WS24** FSM Circuits ৢ Next-State Output State input output Logic Logic Register current next state state **RS**1 clock reset

2

─Finite-State Machine Basics └─FSM Circuits └─Next-State Logic



The next-state logic is a purely combinational circuit, meaning that it must not contain any sequential circuit elements. Hence, it cannot read or depend on the clock or reset signal in any way and – most importantly – must not contain any latches.

Next-State Logic

HWMod WS24

FSM Basics Introduction FSM Circuits Next-State Logic State Register Output Logic Example: Timer

- Purely combinational
 - Must not read the clock or reset
 - Must not contain latches



─Finite-State Machine Basics └─FSM Circuits └─Next-State Logic



It implements the transition function of the FSM by mapping the applied inputs and the current state to the next state. Notice that the current state is provided by the output of the state register, while the next state signal is its input.

Next-State Logic

HWMod WS24

FSM Basics Introduction FSM Circuits Next-State Logic State Register Output Logic Example: Timer

- Purely combinational
 - Must not read the clock or reset
 - Must not contain latches
- Implements the transition function
 - by reading the input and the current state
 - to calculate the next state of the FSM



─Finite-State Machine Basics └─FSM Circuits └─Next-State Logic



As a result state changes can only happen at the active clock edge, where the next state signal is captured by the state register and becomes the new current state. The timing diagram illustrates this behavior. It shows an FSM transitioning through the states A, B and C. At each rising clock edge the value of the next state signal becomes the current state.



─Finite-State Machine Basics └─FSM Circuits └─State Register



As already mentioned, the state register stores the actual state of the FSM and is the **only** sequential circuit element and implemented using a set of D flip-flops.

State Register

- Only synchronous component in the FSM
- Reads next state and outputs current state

─Finite-State Machine Basics └─FSM Circuits └─State Register



At this point we want to stress that, as always, you must strictly adhere to the code patterns for flip-flops discussed in the lecture about sequential circuit elements to avoid the introduction of bugs or non-synthesizable code.

State Register HWMod WS24 SM Basics Work State Logic SM Crocuss Work State Logic SM Pegister Work State Logic

- Reads next state and outputs current state
- Strictly stick to the code patterns for registers!

─Finite-State Machine Basics └─FSM Circuits └─State Register



The reset value of this register defines the initial state of the FSM. The reset can be implemented in a synchronous or asynchronous way. However, in this course we consistently only use the asynchronous variant as discussed in a previous lecture.

State Register HWMod **WS24** Next-State Output input output Logic Logic next current state state clock reset Only synchronous component in the FSM Reads next state and outputs current state Strictly stick to the code patterns for registers! Reset value defines initial state

└─Finite-State Machine Basics └─FSM Circuits └─Output Logic



Exactly like the next-state logic, the output logic is also a purely combinational circuit and it must also under no circumstances contain latches or depend on the clock.

Output Logic HWMod **WS24** Next-State Output input output Purely combinational Logic Register Logic next current state state RST Must not read the clock or reset Ą clock reset Must not contain latches

└─Finite-State Machine Basics └─FSM Circuits └─Output Logic



As its name suggests its purpose is to produce the actual output signals of the FSM.



─Finite-State Machine Basics └─FSM Circuits └─Output Logic



It also defines the state machine's type. Depending on whether the output logic only reads the current state or the current state **and** the inputs, the FSM is either classified as a Moore or a Mealy automaton, respectively.



─Finite-State Machine Basics └─FSM Circuits └─Output Logic



This type basically affects the way an FSM can react to input changes. In Mealy machines, as illustrated by the figure, there is a combinational path from the input to the output. Therefore, the circuit can adjust its output based on an input change in the same clock cycle without the need to wait for an active clock edge. In contrast, Moore machines only derive their outputs from their current state. Hence, they can only react to input changes after the next clock edge, that is, in the next cycle. Moreover, it is sometimes possible to implement the same functionality with a fewer number of states when using a Mealy instead of a Moore machine.



─Finite-State Machine Basics └─FSM Circuits └─Output Logic



Since the next-state and the output logic operate on overlapping sets of inputs, it is in practice often the case that logic can be shared between these two components. This holds true for both types of state machines, Mealy and Moore. However, this is nothing you need to worry about too much, as the synthesis tool will figure this out for you.



─Finite-State Machine Basics └─FSM Circuits └─Output Logic - Combinational Loops

Although it might seem that Mealy machines have clear advantages over Moore machines, care must be taken as they can easily introduce combinational loops into a system.

Output Logic - Combinational Loops

HWMod WS24

FSM Basics Introduction FSM Circuits Next-State Logic State Register Output Logic Example: Timer Interacting Mealy FSMs can lead to combinational loops!

Interacting Mealy FSMs can lead to combinational loops!

─Finite-State Machine Basics └─FSM Circuits └─Output Logic - Combinational Loops

 Interac Example 	ting Mealy FSMs can lead to combinational loop: le
# FS	M A: "Start when not busy"
E FS	M B: "Assert busy when started"
	FSM A busy FSM B

Consider the shown example, of two FSMs that interact with each other using a "start" and a "busy" signal. FSM A uses the start signal to initiate some operation in FSM B, while FSM B uses the busy signal to indicate to A that is ready to receive the start signal. Assume both A and B are Mealy machines. The output logic of A sets the start signal to high as soon as the busy signal it receives from B is low. At the same time, the output logic of B sets the busy signal to high as soon as it sees a high level on the start input. This will essentially lead to B simply forwarding the "start" to its "busy" signal, and to A negating the applied "busy" signal. In this scenario we thus end up with a combinational loop formed by the output-logic of FSM A and B. A possible fix would be to implement B as a Moore state machine and only set the busy signal in the cycle after the start signal was set to high.



—Finite-State Machine Basics —FSM Circuits —Output Logic - Combinational Loops



In summary: Be aware of this potential issue and try to avoid Mealy machines unless, they are really necessary.

Output Logic - Combinational Loops

HWMod WS24

FSM Basics Introduction FSM Circuits Next-State Logic State Register Output Logic Example: Timer

- Interacting Mealy FSMs can lead to combinational loops!
- Example
 - FSM A: "Start when not busy"
 - FSM B: "Assert busy when started"



Only use Mealy machines when really necessary

─Finite-State Machine Basics └─Example: Timer └─Simple Timer - Specification

 $\label{eq:second} \begin{array}{l} \hline \textbf{Means Descendence} \\ \hline \textbf{Means Descendence} \\ \mbox{array} & \mbox{$

As the introduction over the previous slides might have been a bit abstract, let's look at a simple concrete example to get a better understanding of the discussed components. The example that we will use for this is a very basic timer circuit, with the interface shown on the lower part of the slide. Its behavior is quite straightforward. The timer keeps an internal N-bit counter, which represents the state of the FSM. Whenever the en input is high, this counter is synchronously incremented. This is a task implemented by the next-state logic. When the counter reaches its maximum value $(2^N - 1)$ it simply overflows to zero. The output tick is set to high whenever the maximum value is reached and the en input is high.

Simple Timer - Specification

Behavior Description

HWMod

WS24

Specification

The simple_timer module keeps an internal N-bit counter that is (synchronously) incremented as long as en is asserted. When the counter reaches its maximum value $(2^N - 1)$ it overflows back to zero. If the counter hits its maximum and en is asserted the tick output is asserted.

Interface

```
1 entity simple_timer is
    generic (
3
      N : positive := 8
4
    );
5
    port (
6
      clk
             : in std_ulogic;
      res_n : in std_ulogic;
7
      en
             : in std_ulogic;
8
9
      tick : out std_ulogic
    );
11 end entity;
```

Finite-State Machine Basics
Example: Timer
Simple Timer - Circuit

Here we see a circuit that implements the specification formulated on the previous slide. By now you should already recognize most of the shown circuit elements. However, for the sake of completeness: The circle-shaped components with the equals sign and plus sign represent an adder and a comparator, respectively.

Simple Timer - Circuit



FSM Basics Introduction FSM Circuits Example: Timer Specification Circuit Implementation



─Finite-State Machine Basics └─Example: Timer └─Simple Timer - Circuit



Of course, the central D flip-flops represent the state register. Its output, that is the signal labeled ent, holds the current state of the FSM. It feeds the output as well as the next state logic. The next state logic uses a multiplexer to decide whether to increment the count value, based on the en input. It produces the ent_nxt signal, which is connected to the input of the state register. At the next rising clock edge this signal will be captured by the state register, thus becoming the new state. Besides the current state, the output logic also reads the en input, making the circuit a Mealy state machine.



─Finite-State Machine Basics └─Example: Timer └─Simple Timer - Implementation



Let us now look at a possible architecture for the simple_timer entity.

Simple Timer - Implementation

HWMod WS24

FSM Basics Introduction FSM Circuits Example: Timer Specification Circuit Implementation

```
clk → RST
                                         res_n
1 architecture arch of simple_timer is
                                               16
    subtype cnt_t is
                                               17
2
3
       unsigned (N-1 downto 0);
                                               18
     signal cnt, cnt_nxt : cnt_t;
4
                                               19
5
     constant M : natural := 2**N-1;
                                               20
6 begin
                                               21
7
                                               22
8
     state_reg : process(clk, res_n)
                                               23
9
    begin
                                               24
     if res_n = '0' then
                                               25
10
         cnt <= (others => '0');
                                               26
       elsif rising_edge(clk) then
12
                                               27
         cnt <= cnt_nxt;</pre>
                                               28
14
       end if;
                                               29
15
     end process;
```

ont nxt

```
16 next_state_logic : process(all)
17 begin
18 cnt_nxt <= cnt;
19 if en = '1' then
20 cnt_nxt <= cnt + 1;
21 end if;
22 end process;
23
24 output_logic : process(all)
25 variable comp : std_ulogic;
26 begin
27 comp := '1' when cnt=M else '0';
28 tick <= en and comp;
29 end process;
30 end architecture;
</pre>
```

 $2^{N} - 1$

-tick

─Finite-State Machine Basics └─Example: Timer └─Simple Timer - Implementation



To implement it, we first declare the signals cnt and cnt_nxt.

Simple Timer - Implementation

HWMod WS24

SM Basics Introduction FSM Circuits Example: Timer Specification Circuit Implementation

```
-tick
                                    ont nxt
                                         clk - RST
                                         res_n
1 architecture arch of simple_timer is
                                              16
                                                   next_state_logic : process(all)
    subtype cnt_t is
                                              17
                                                   begin
2
3
       unsigned (N-1 downto 0);
                                              18
                                                     cnt_nxt <= cnt;</pre>
    signal cnt, cnt_nxt : cnt_t;
                                                      if en = '1' then
4
                                              19
5
     constant M : natural := 2**N-1;
                                              20
                                                        cnt_nxt <= cnt + 1;</pre>
                                                      end if;
6 begin
                                              21
7
                                              22
                                                   end process;
8
     state_reg : process(clk, res_n)
                                              23
                                                   output_logic : process(all)
9
    begin
                                              24
      if res_n = '0' then
                                              25
                                                      variable comp : std_ulogic;
10
         cnt <= (others => '0');
                                              26
                                                   begin
       elsif rising_edge(clk) then
                                                      comp := '1' when cnt=M else '0';
12
                                              27
         cnt <= cnt_nxt;</pre>
                                              28
                                                      tick <= en and comp;
14
       end if;
                                              29
                                                   end process;
                                              30 end architecture;
15
     end process;
```

 $2^{N} - 1$

Finite-State Machine Basics Example: Timer **Simple Timer - Implementation**

	architecture arch of simple_timer is	16	next_state_logic + process(all)	
	subtype dat t is	17	begin	
	unsigned(N-1 downto 0);	18	005 HAT <- 0014	
	signal cat, cat_nat : cat_t;	10	if en = 'i' then	
	constant N i satural i- 2++N-1;	20	cat pat <- cat + 1a	
	begin	21	end if:	
		22	end processy	
	state yes ; processicik, yes as	23		
	beain	25	output logic (process(all)	
10	if yes p = '0' then	25	variable comp a std plogica	
	cos - coshera -> 10131	25	begin	
12	elsif rising edge tolkt them	27	comp in fif when out moles for	
13	cost <- cast_sasty	28	tick <= en and compy	
14	end 15;	- 20	and process;	
10	end process;	30	end architectures	

Next, using these signals, we can implement the state register via a synchronous process. For that we simply follow the code pattern presented in the lecture about sequential circuit elements. During the asynchronous reset, the counter is set to zero, which defines the initial state of our FSM. On rising clock edges, we simply assign the next signal to the counter to update the state register.

Simple Timer - Implementation

HWMod **WS24**

Implementation

2

3

4

5

7

8

9

10

12

14

15

```
2^{N} - 1
                                                            tick
                                   ont nxt
1 architecture arch of simple_timer is
                                             16
                                                  next_state_logic : process(all)
    subtype cnt_t is
                                             17
                                                  begin
      unsigned (N-1 downto 0);
                                             18
                                                    cnt_nxt <= cnt;</pre>
    signal cnt, cnt_nxt : cnt_t;
                                                    if en = '1' then
                                             19
    constant M : natural := 2**N-1;
                                             20
                                                      cnt_nxt <= cnt + 1;</pre>
                                                    end if;
6 begin
                                             21
                                                  end process;
                                             22
    state_reg : process(clk, res_n)
                                             23
                                                  output_logic : process(all)
                                             24
      if res_n = '0' then
                                                    variable comp : std_ulogic;
                                             25
        cnt <= (others => '0');
                                                  begin
                                             26
                                                    comp := '1' when cnt=M else '0';
      elsif rising_edge(clk) then
                                             27
        cnt <= cnt_nxt;</pre>
                                             28
                                                    tick <= en and comp;
      end if;
                                             29
                                                  end process;
                                             30 end architecture;
    end process;
```

—Finite-State Machine Basics Example: Timer Simple Timer - Implementation



Let us now turn our attention to the next-state logic. Recall that this function is purely combinational and is hence also implemented in a purely combinational process. The required multiplexer is described using an if-condition which checks the en signal. Notice, that we only write to the "next" signal, as the task of the next state logic is to prepare the next value that will be captured into the state register.

Simple Timer - Implementation

HWMod WS24

FSM Basics Introduction FSM Circuits Example: Timer Specification Circuit Implementation

```
2^{N} - 1
                                                               tick
                                     ont nxt
                                          clk → RST
                                         res_n
1 architecture arch of simple_timer is
                                               16
                                                    next_state_logic : process(all)
     subtype cnt_t is
                                               17
2
3
       unsigned (N-1 downto 0);
                                               18
                                                       cnt_nxt <= cnt;</pre>
     signal cnt, cnt_nxt : cnt_t;
                                                       if en = '1' then
4
                                               19
5
     constant M : natural := 2**N-1;
                                               20
                                                         cnt_nxt <= cnt + 1;</pre>
6 begin
                                               21
                                                       end if;
7
                                               22
                                                    end process;
     state_reg : process(clk, res_n)
                                               23
8
                                                    output_logic : process(all)
9
    begin
                                               24
       if res_n = '0' then
                                               25
                                                       variable comp : std_ulogic;
10
         cnt <= (others => '0');
                                               26
                                                    begin
       elsif rising_edge(clk) then
                                                       comp := '1' when cnt=M else '0';
12
                                               27
         cnt <= cnt_nxt;</pre>
                                                       tick <= en and comp;
                                               28
14
       end if;
                                               29
                                                    end process;
                                               30 end architecture;
15
     end process;
```

Finite-State Machine Basics -Example: Timer **Simple Timer - Implementation**



Finally, the output logic is also implemented using its own process. As with the next state logic, this process must also be purely combinational. The constant M, representing the maximum value of the counter, is declared in the declaration part of the architecture.

Simple Timer - Implementation

HWMod **WS24**

Implementation

2

3

4

5

7

8

9

10

12

14

15

```
tick
                                   ont nxt
                                        clk - RST
                                        res_n
1 architecture arch of simple_timer is
                                             16
                                                  next_state_logic : process(all)
    subtype cnt_t is
                                             17
                                                  begin
      unsigned (N-1 downto 0);
                                             18
                                                    cnt_nxt <= cnt;</pre>
    signal cnt, cnt_nxt : cnt_t;
                                                    if en = '1' then
                                             19
    constant M : natural := 2**N-1;
                                             20
                                                       cnt_nxt <= cnt + 1;</pre>
                                                    end if;
6 begin
                                             21
                                                  end process;
                                             22
    state_reg : process(clk, res_n)
                                             23
                                                  output_logic : process(all)
    begin
                                             24
      if res_n = '0' then
                                             25
                                                     variable comp : std_ulogic;
        cnt <= (others => '0');
                                             26
      elsif rising_edge(clk) then
                                                     comp := '1' when cnt=M else '0';
                                             27
        cnt <= cnt_nxt;</pre>
                                             28
                                                     tick <= en and comp;
      end if;
                                             29
                                                  end process;
    end process;
```

```
30 end architecture;
```

Finite-State Machine Basics -Example: Timer Simple Timer - Implementation



The FSM implementation style we used here is referred to as the 3-process-method, as it uses three separate processes for each of the three state machine components. As we will see in an upcoming lecture, there also exists a 2- and 1-processmethod.

Simple Timer - Implementation

HWMod **WS24**

Implementation

```
tick
                                    ont nxt
                                          clk → RST
                                         res_n
1 architecture arch of simple_timer is
                                              16
                                                   next_state_logic : process(all)
     subtype cnt_t is
                                              17
                                                   begin
2
3
       unsigned (N-1 downto 0);
                                              18
                                                      cnt_nxt <= cnt;</pre>
     signal cnt, cnt_nxt : cnt_t;
                                                      if en = '1' then
4
                                              19
5
     constant M : natural := 2**N-1;
                                              20
                                                        cnt_nxt <= cnt + 1;</pre>
6 begin
                                              21
                                                      end if;
7
                                                   end process;
                                              22
8
     state_reg : process(clk, res_n)
                                              23
                                                   output_logic : process(all)
9
    begin
                                              24
      if res_n = '0' then
                                              25
                                                      variable comp : std_ulogic;
10
         cnt <= (others => '0');
                                              26
                                                   begin
       elsif rising_edge(clk) then
                                                      comp := '1' when cnt=M else '0';
12
                                              27
         cnt <= cnt_nxt;</pre>
                                                      tick <= en and comp;
                                              28
14
       end if;
                                              29
                                                   end process;
15
     end process;
```

```
30 end architecture;
```

 $2^{N} - 1$

Thank you for listening! We recommend you to immediately take the self-check test in TUWEL, to see if you understood the material presented in this lecture.



FSM Basics Introduction FSM Circuits Example: Timer Specification Circuit Implementation

Lecture Complete!

Modified: 2025-03-10, 16:07 (b25118c)