

Introduction to Complex FSMs Design



Alfredo BENSO
Politecnico di Torino (Italy)
UCSD, CA

Alfredo.benso@polito.it
www.testgroup.polito.it

Goal

- This section presents a design methodology to perform Manual Synthesis of Complex Finite State Machines.

Prerequisites

- Simple FSM
- RT-level design

2.1

3

Outline

- Introduction
- The concept of RT-state
- RT-state based synthesis
- Some examples
- The concept of macro RT-state.

2.1

4

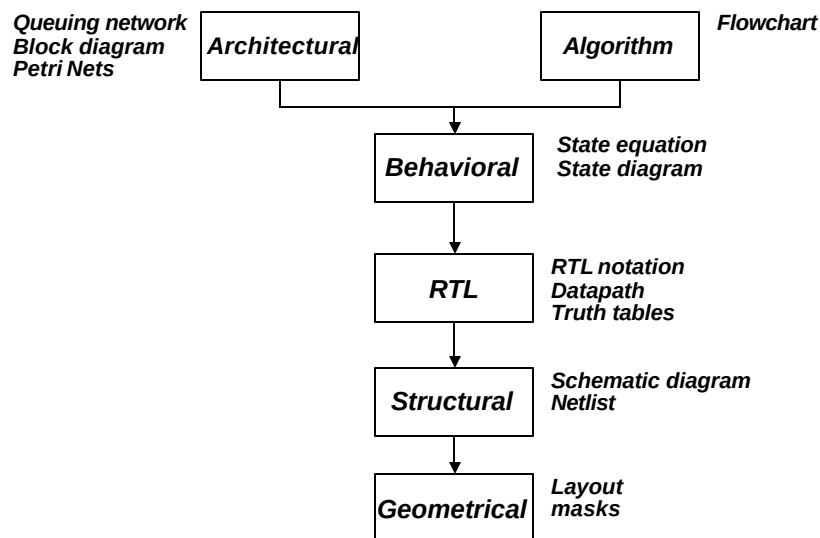
Levels of abstraction in VLSI design

- A design of a digital circuit requires a transformation from a "concept" to an "implementation", in a series of ordered levels.
- From the highest level to lower levels of design "abstraction", a design is iteratively refined.
- The design description is verified and validated at each level, often cycling between levels of abstraction.
- Design descriptions are described using one or more domain representations (Behavior, Structure, Physical).

2.1

5

Levels of abstraction in VLSI design



2.1

6

What's a Complex FSM?

There isn't a formal definition of Complex Finite State Machine.

We can say that we need a Complex FSM when the *simple FSM formalism* or the *RT-level approach* are not powerful enough to describe the circuit we want to design.

2.1

7

Simple FSM formalism

The behavior of a simple FSM is specified using states in order to implement a Mealy or a Moore machine.

This formalism is very powerful for small designs, but it becomes too complex when dealing with complex designs.

For example, imagine the complexity of designing a system requiring a memory, a counter, a comparator, and a timer, just using Moore or Mealy states.

2.1

8

RT-level approach

Designing an RT-level circuit, means to connect together standard components (memories, registers, counters, etc...) in order to obtain a system that behaves as desired.

This approach is useful for simple designs involving standard functionalities, but it is very limited if we want to design custom circuits with not standard functionalities.

2.1

9

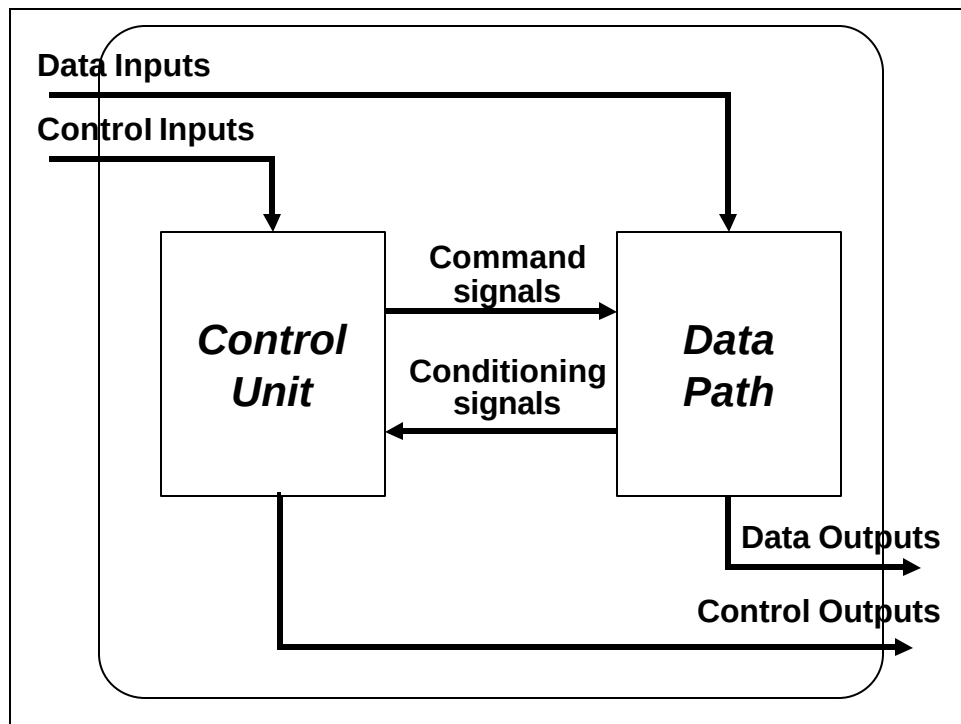
What's a Complex FSM?

A Complex FSM is a system including a Control Unit (designed as a FSM) and a Data Path (designed as an RT-level circuit) that are integrated together in order to implement complex functionalities.

The final description will include a structural (RTL) view of the Data Path + a logic-level specification of the Control Unit)

2.1

10



CU vs DP

Control Unit	Data Path
Modeled using FSM (finite state machine) model.	Modeled using RT-model.
Defines clock-based sequencing of actions in data path or external to the block	Defines both synch and async transformations of data moving through the block

Data Path (DP)

The Data Path is an interconnection of resources.

RTL resources include:

- Functional resources (ALU, adder, multiplier, ...)
- Memory resources (register, RAM, ROM, ...)
- Interface resources (bus, steering logic, I/O pad, ...)

2.1

13

Data Path (DP)

- The DP is in charge of executing all the operations listed in the State Transition Graph of the Control Unit.
- Executions inside the DP are enabled by the Control Unit via proper *Command Signals*.
- Via proper *Conditioning Signals*, the DP returns the Control Units the information needed to evolve through states.

2.1

14

Data Path (DP)

- Data Path and Control Unit are synchronized using clock signals
- Benefits of synchronization:
 - Eliminate unpredictability of output behavior due to timing skew.
 - Create signal stability, as they must have stable values for certain period of time.
 - Better isolate signals from noise transients.
- Controller sequences operations in the data path.

2.1

15

Control Unit

- On the basis of the values of
 - Present state
 - Control Inputs
 - Conditioning signal provided by the DPon each state it determines:
 - its next state
 - the set of *Command Signals* needed to enable the set of concurrent operations the DP has to perform on the next rising edge of the clock.

2.1

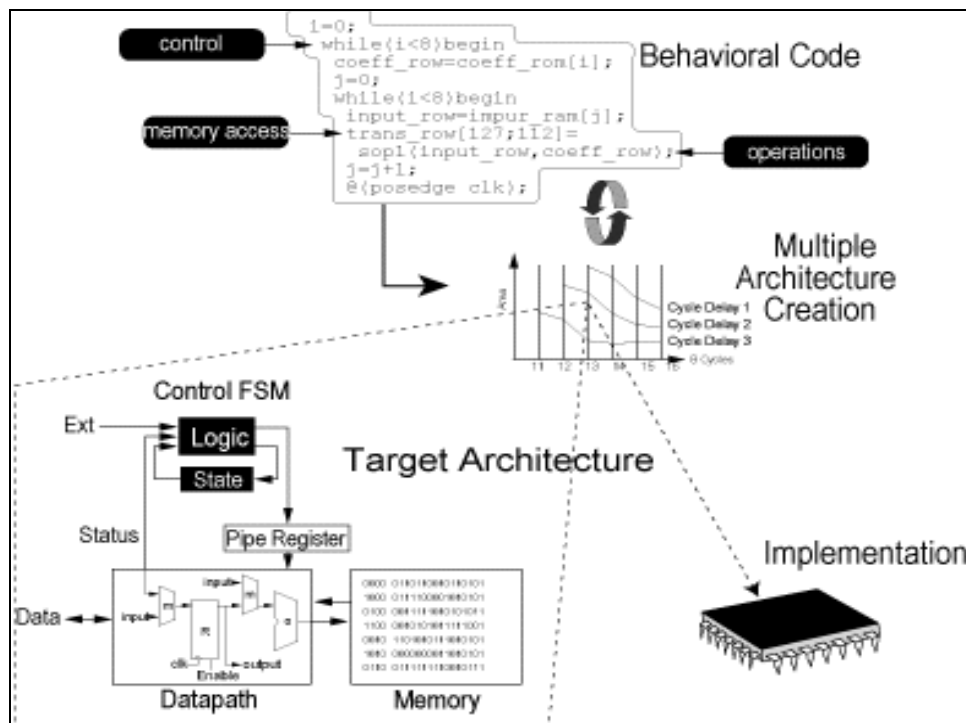
16

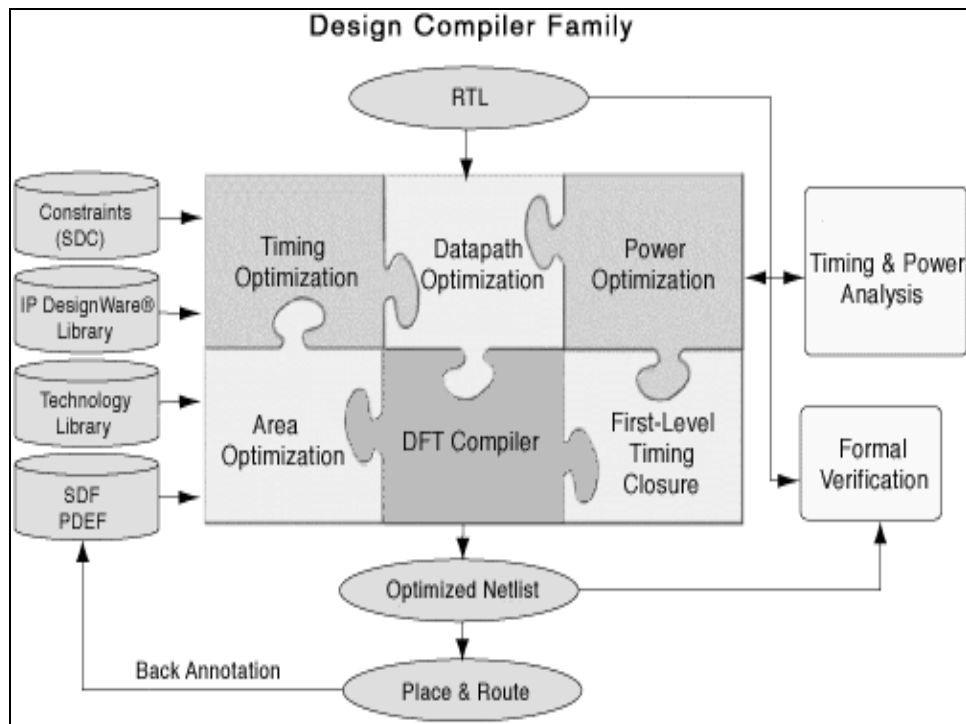
Automatic synthesis

- The system is described without partitioning CU and DP.
- The CAD software deals with the partitioning and the synthesis steps.
- Synthesis methodologies are different, according to the complexity of the target unit.
- Common tools: Synopsys, Cadence, Mentor, ...
- A license for a tool costs several tens of thousand of dollars

2.1

17





Manual synthesis

- Each Unit is synthesized independently.
- Synthesis methodologies are different, according to the complexity of the target unit.
- The Data Path is synthesized first.
- The Control Unit is synthesized later, to guarantee the Data Path performs the required operations.

Outline

- Introduction
- The concept of RT-state
- RT-state based synthesis
- Some examples
- The concept of macro RT-state.

2.1

21

Introduction

In targeting system level behavioral descriptions, it may be helpful resorting to the introduction of “States” the process evolves through.

2.1

22

Introduction (cont'd)

- In each state, hereinafter referred to as RT-state, the system can perform several complex concurrent RT level operations
- The evolution between states is triggered by the master clock signal.

2.1

23

RT-state

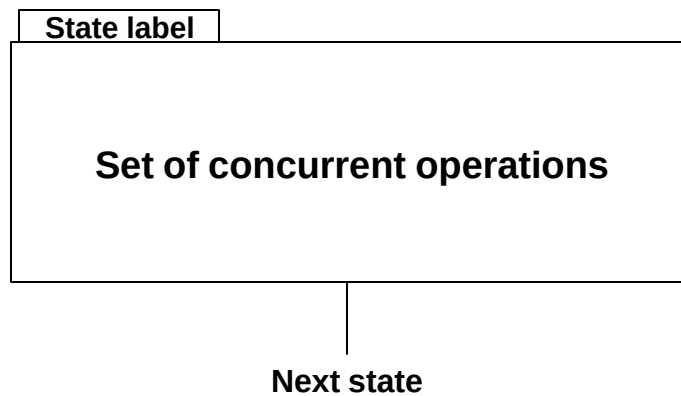
A new graphical representation formalism needs to be introduced.

2.1

24

Representation formalism

Each state looks like:



2.1

25

Representation semantic

The semantic is the following:

- when in the state, on the *next* rising edge of the master clock:
 - the set of concurrent operations will be executed concurrently
 - the next state entered.

2.1

26

Conditioning

The execution of some operations can be conditioned by the occurrence of some conditions:

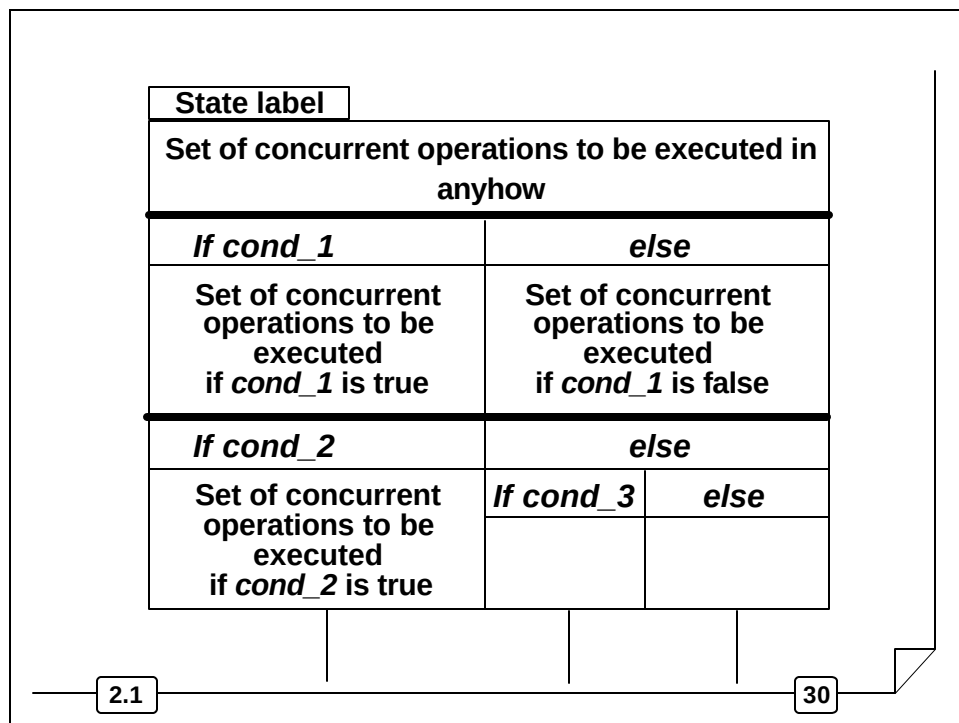
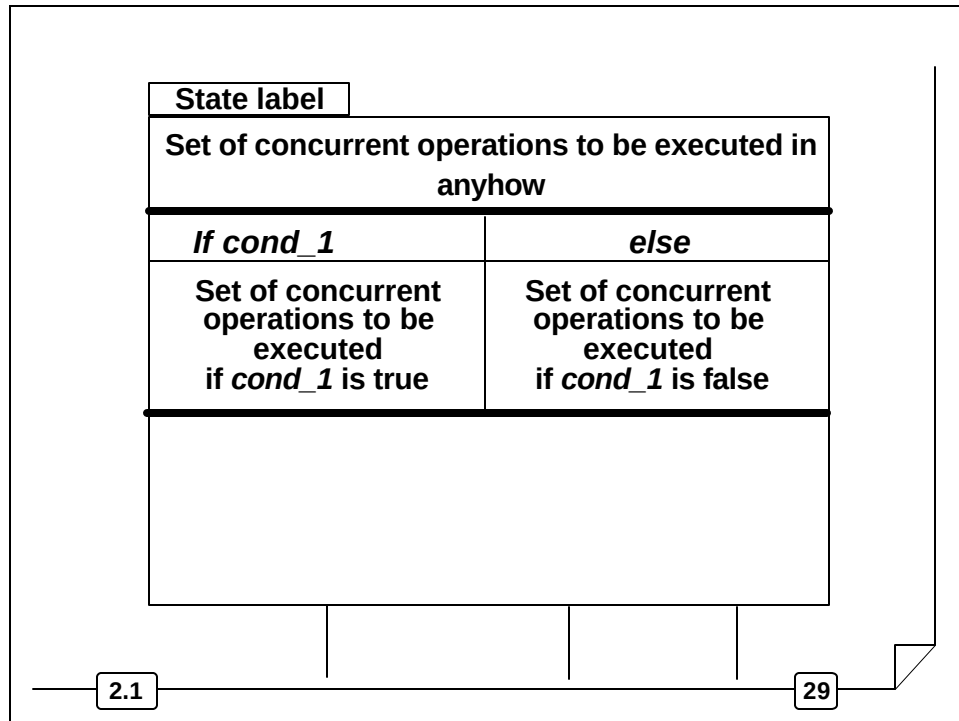
2.1

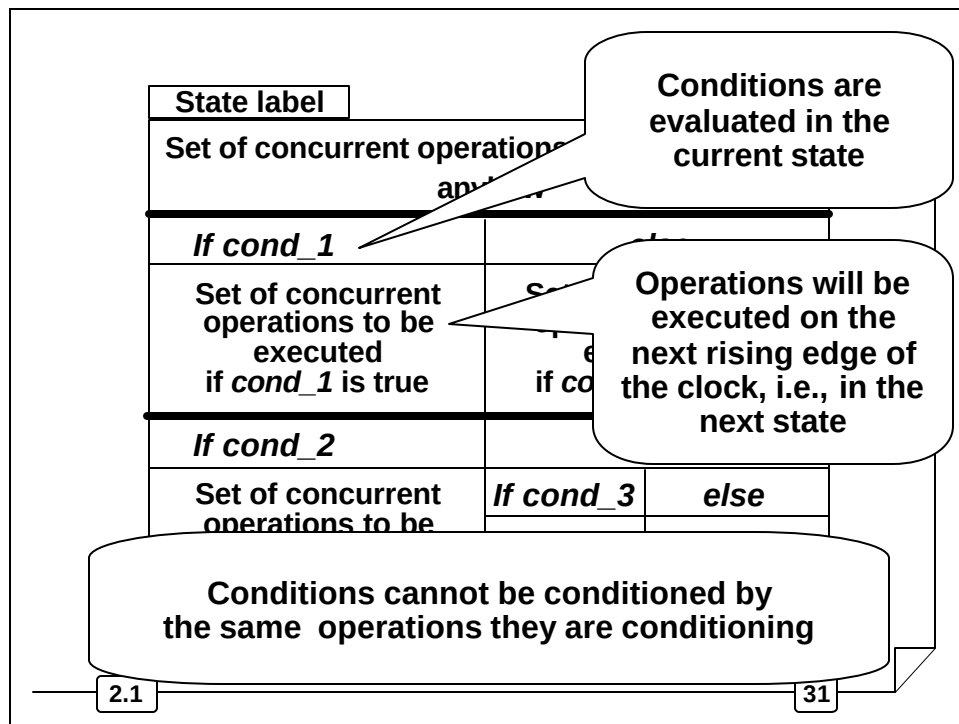
27

State label**Set of concurrent operations to be executed in
anyhow**

2.1

28



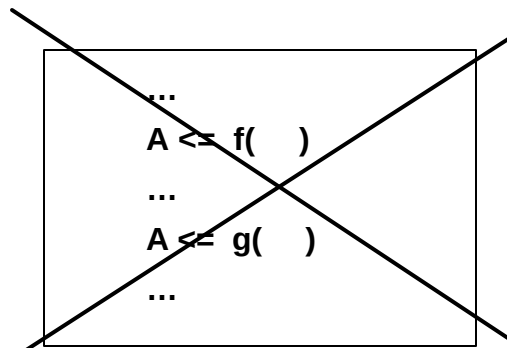


Some practical implications

- In a given set of operations, each “memory” device (flip-flops, registers, counters, RAM cells, etc) must be assigned a new value just once:

Some practical implications

- In a given set of operations, each “memory” device (flip-flops, registers, counters, RAM cells, etc) must be assigned a new value just once:



2.1

33

Warning !!

- All the operations listed in a given set must be executed concurrently and in just one clock cycle !!

2.1

34

Warning !!

- All the operations listed in a given set must be executed concurrently and in just one clock cycle !!



- The designer has to trade-off between the powerfulness of the operations and the complexity of the hardware actually needed to implement them

2.1

35

- Some operations may require complex hardware structures:

2.1

36

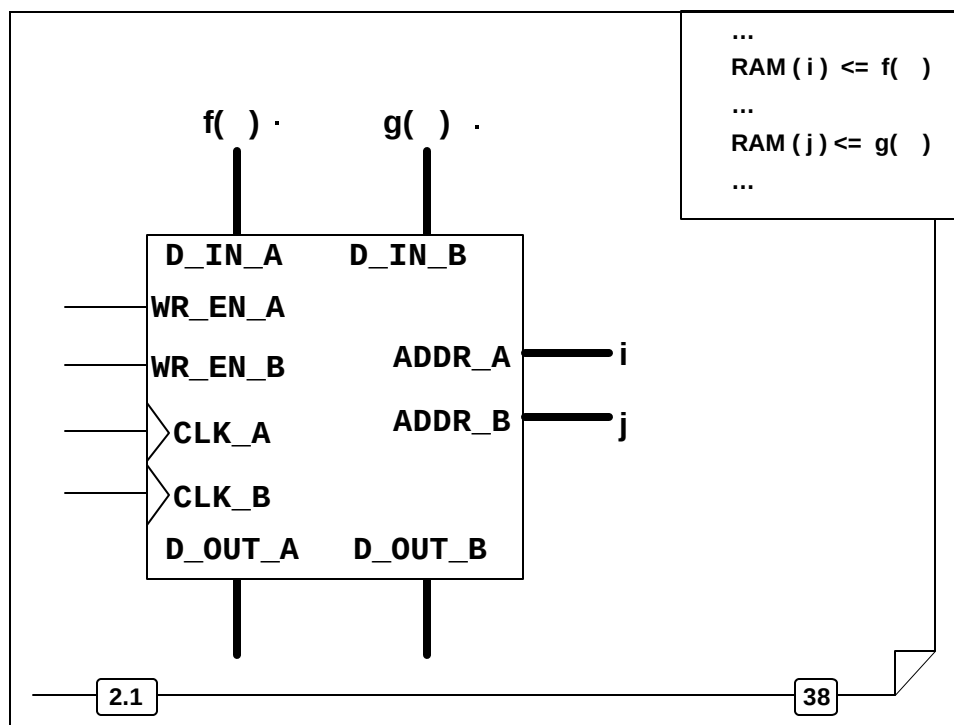
- Some operations may require complex hardware structures:

```
...  
RAM ( i ) <= f( )  
...  
RAM ( j ) <= g( )  
...
```

requires the adoption of a dual port memory:

2.1

37



2.1

38

- Some operations cannot be accomplished in just a single clock cycle:

2.1

39

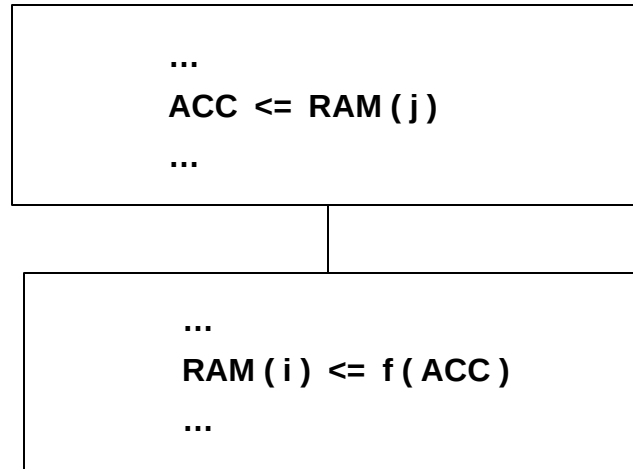
- Some operations cannot be accomplished in just a single clock cycle:

```
...  
RAM ( i ) <= f ( RAM ( j ) )  
...
```

2.1

40

It should be slit into 2 states:



2.1

41

Outline

- Introduction
- The concept of RT-state
- RT-state based synthesis
- Some examples
- The concept of macro RT-state.

2.1

42

RT-state based synthesis

Several design methodologies exist for RT-state based system descriptions.

There are two main approaches:

- **VHDL-based :**
 - it relies on the availability of a VHDL synthesis tool. It will be not be covered in this course
- **Purely manual:**
 - it will be covered in this course

2.1

43

Outline

- **Introduction**
- **The concept of RT-state**
- **RT-state based synthesis**
- **Some examples**
- **The concept of macro RT-state.**

2.1

44

Example: Mouse controller

A “Mouse controller” is to be designed.

It has:

- A 1-bit data input KEY, asserted to ‘1’ by the mouse whenever the left key of the mouse itself is pressed
- A clock signal CLK, which acts as a proper sampling signal of KEY, i.e., the frequency of CLK is such that it never happens that two transitions of KEY occur within a same CLK cycle

2.1

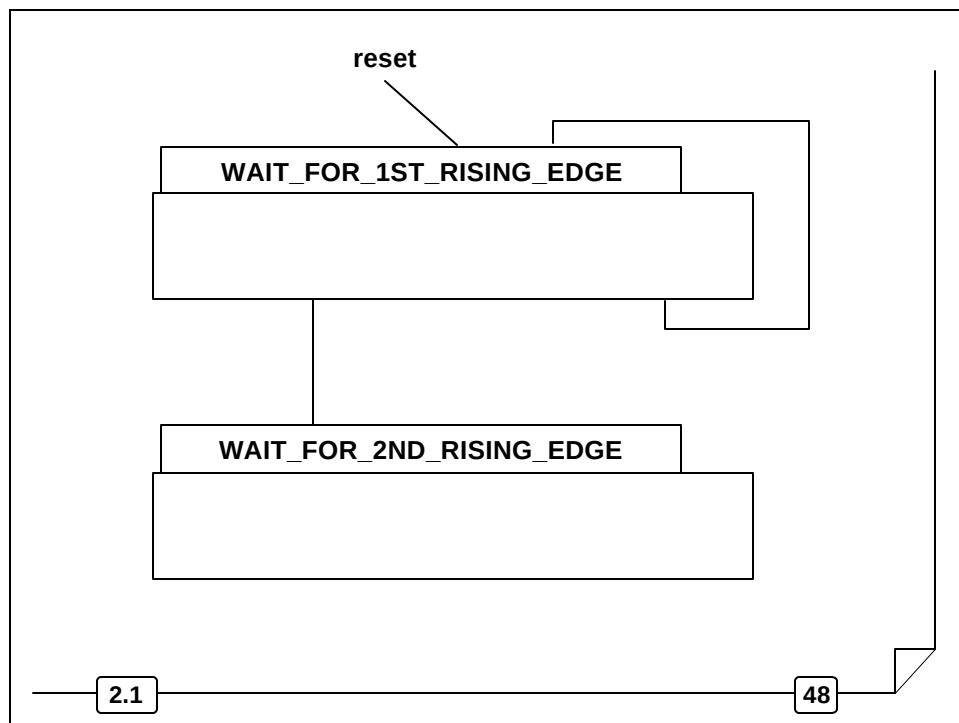
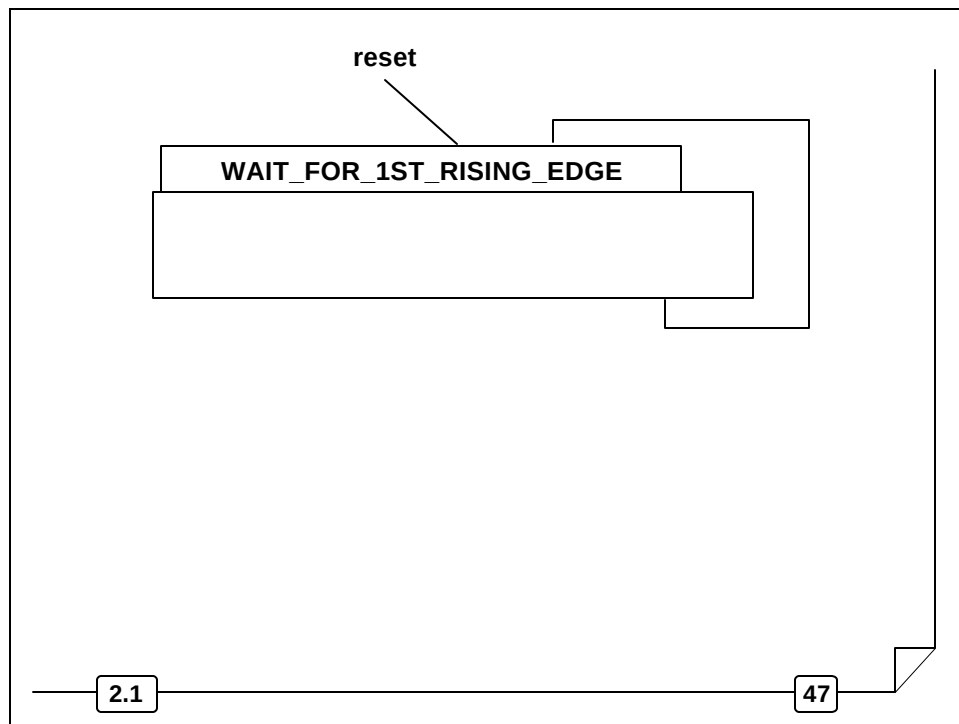
45

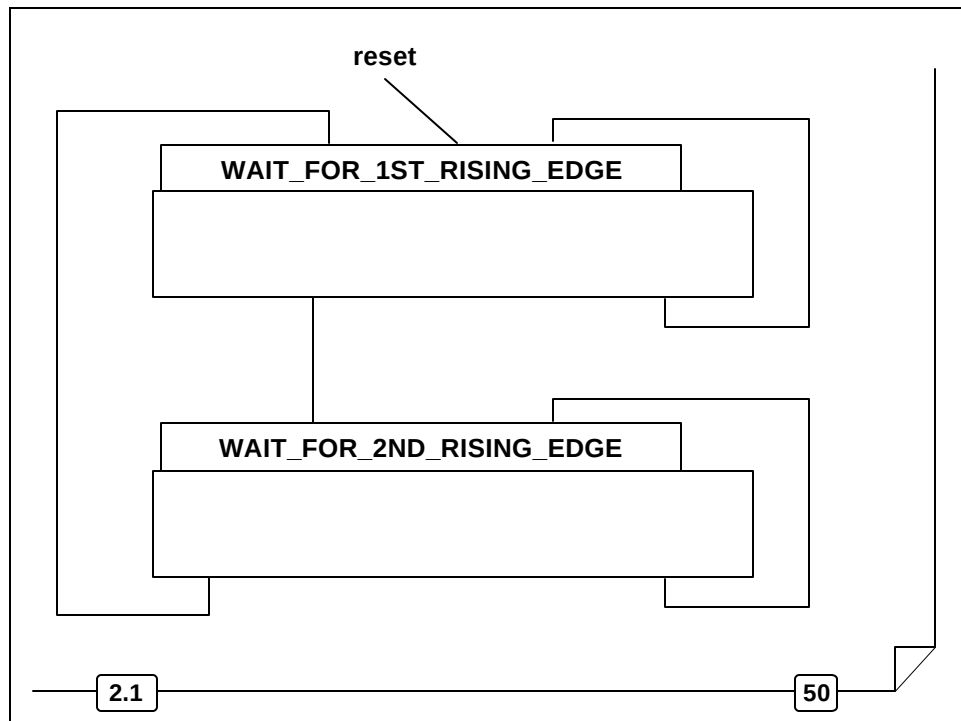
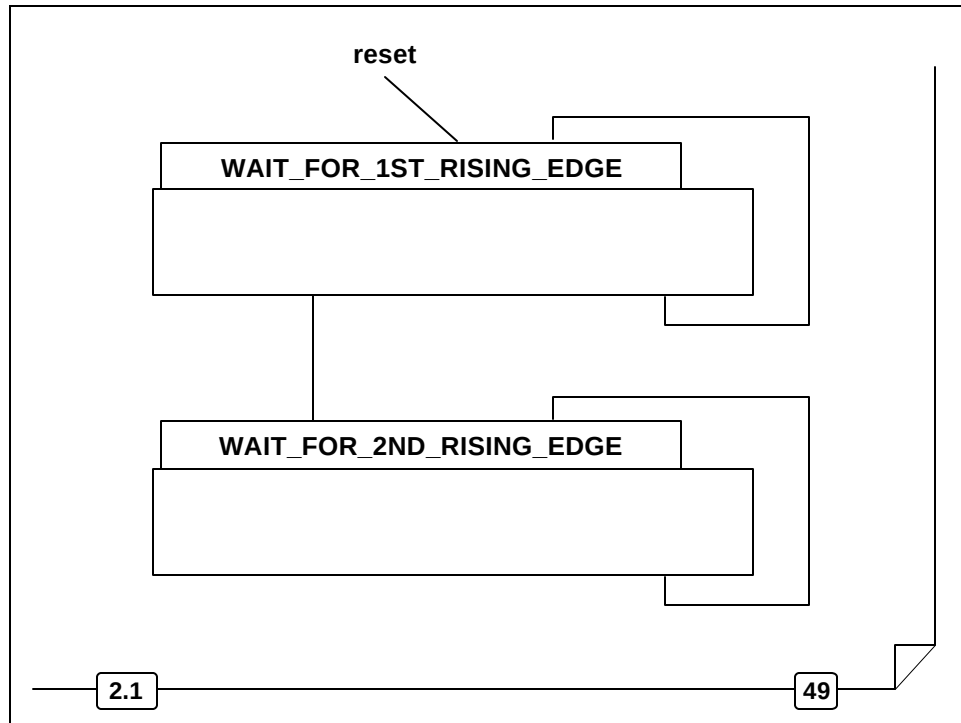
Example: Mouse controller (cont'd)

- An output DOUBLE_CLICK, to be asserted to ‘1’ for 1 clock cycle whenever less than 6 clock cycles occur between two consecutive rising edges of the input KEY
- An output TWO_CLICKS, to be asserted to ‘1’ for 1 clock cycle whenever more than 6 and less than 12 clock cycles occur between two consecutive rising edges of the input KEY.

2.1

46





Operations to be performed when the Reset is asserted

```
timer <= 0  
last_key <= '0'  
two_clicks <= '0'  
double_click <= '0'
```

WAIT_FOR_1ST_RISING_EDGE

2.1

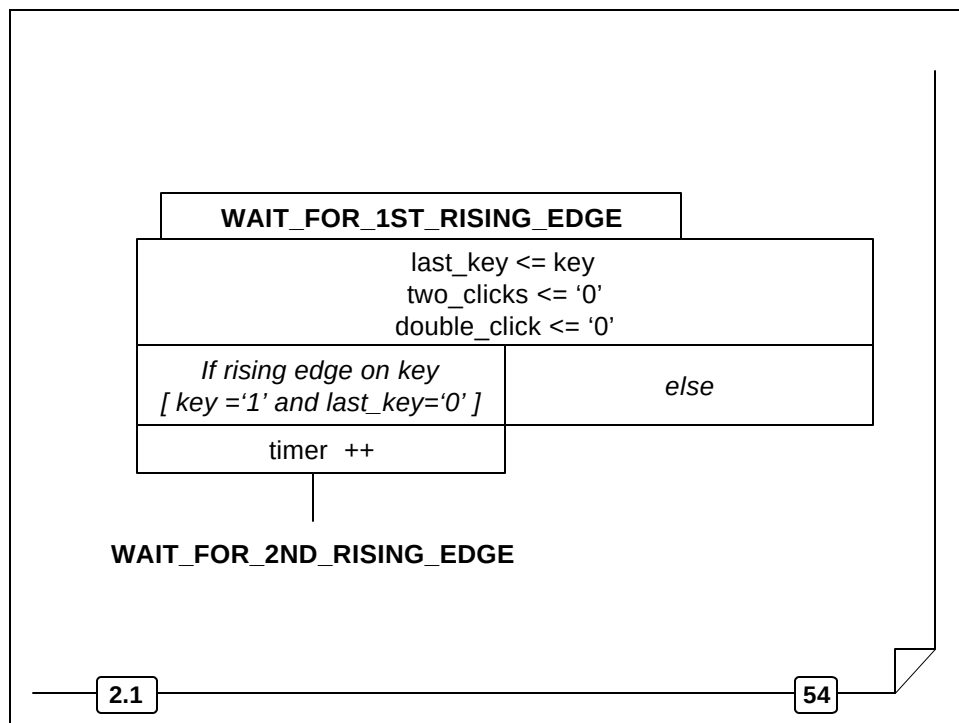
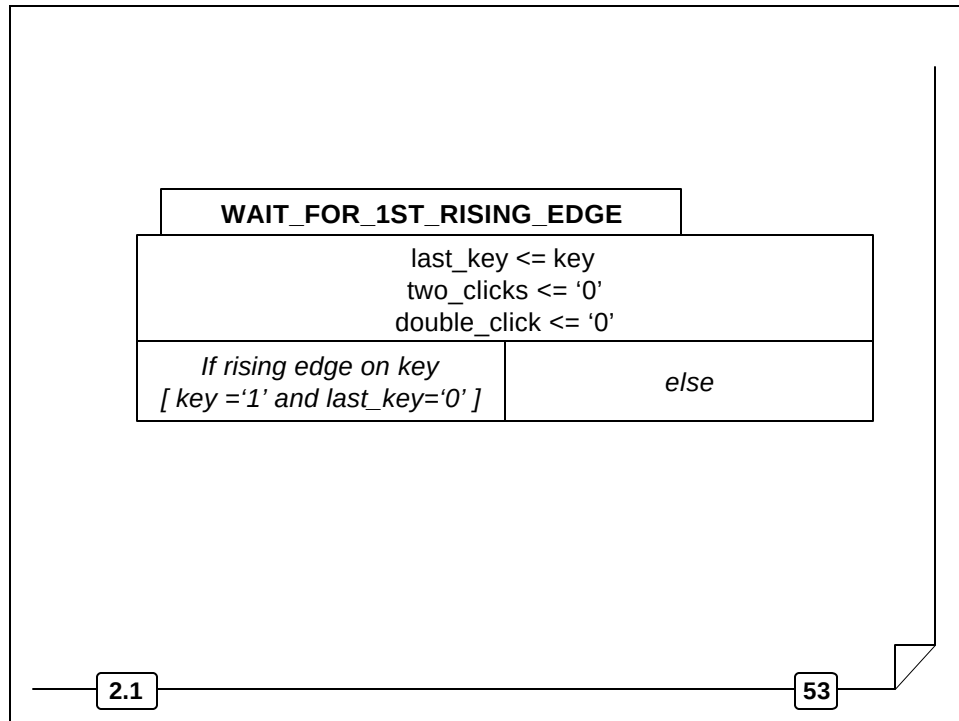
51

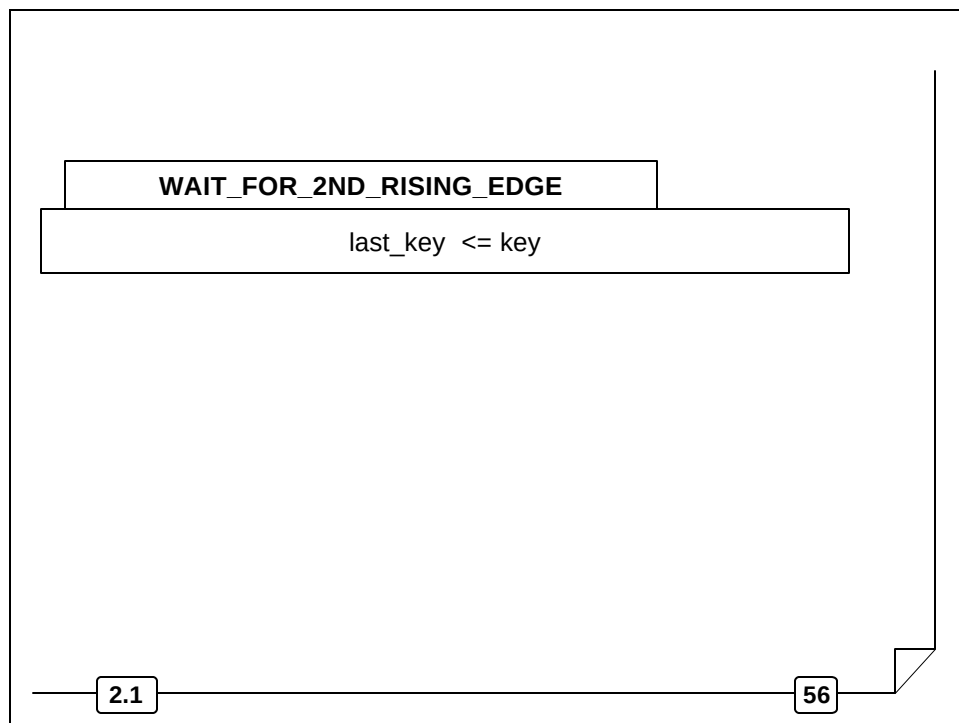
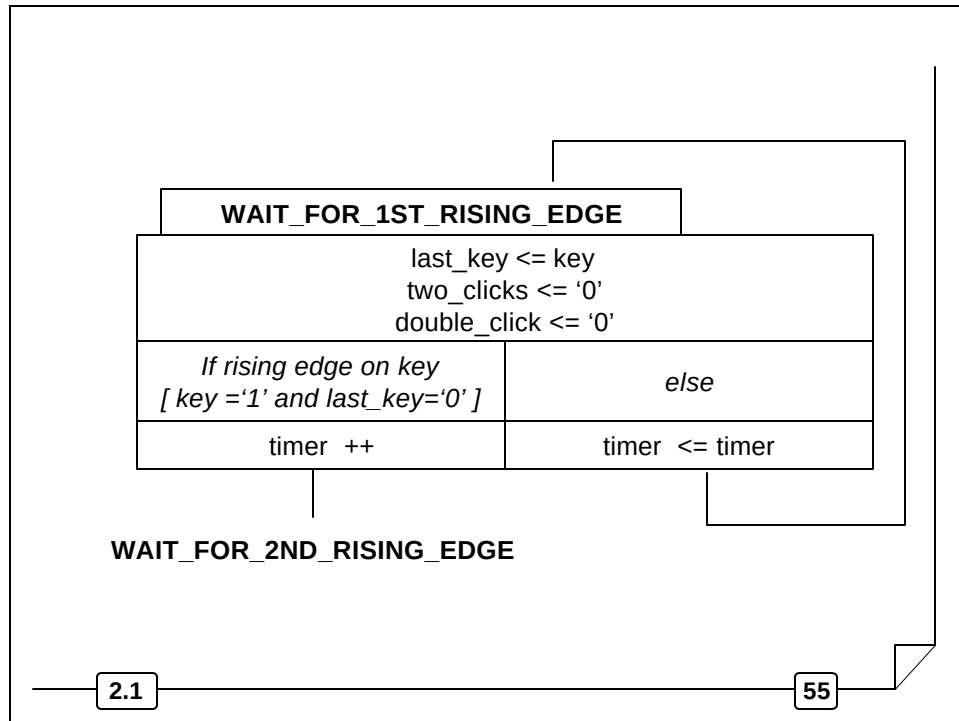
WAIT_FOR_1ST_RISING_EDGE

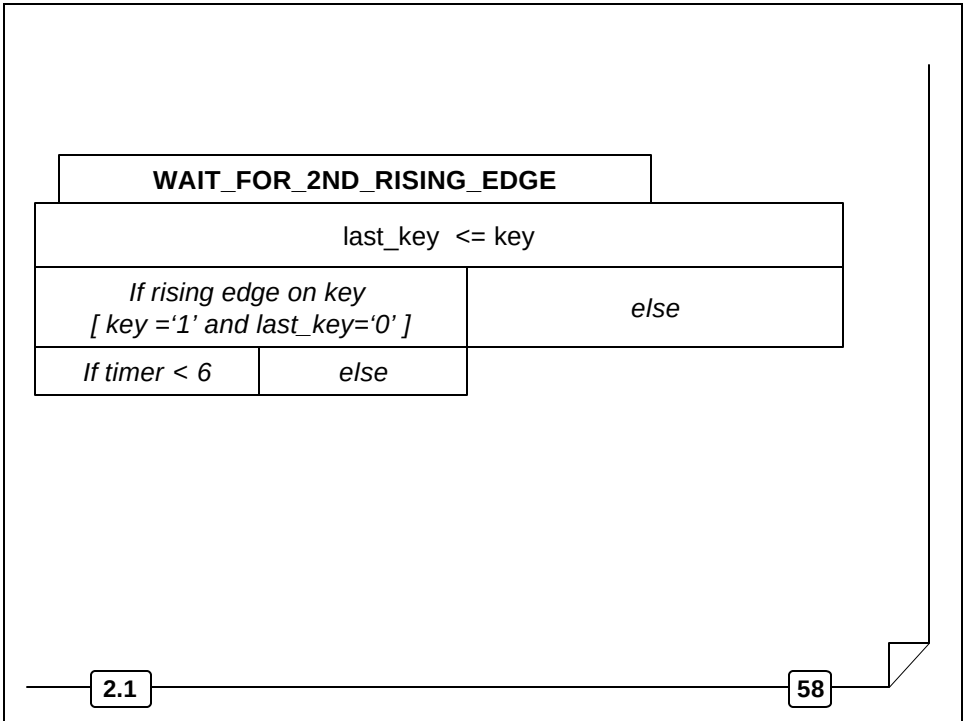
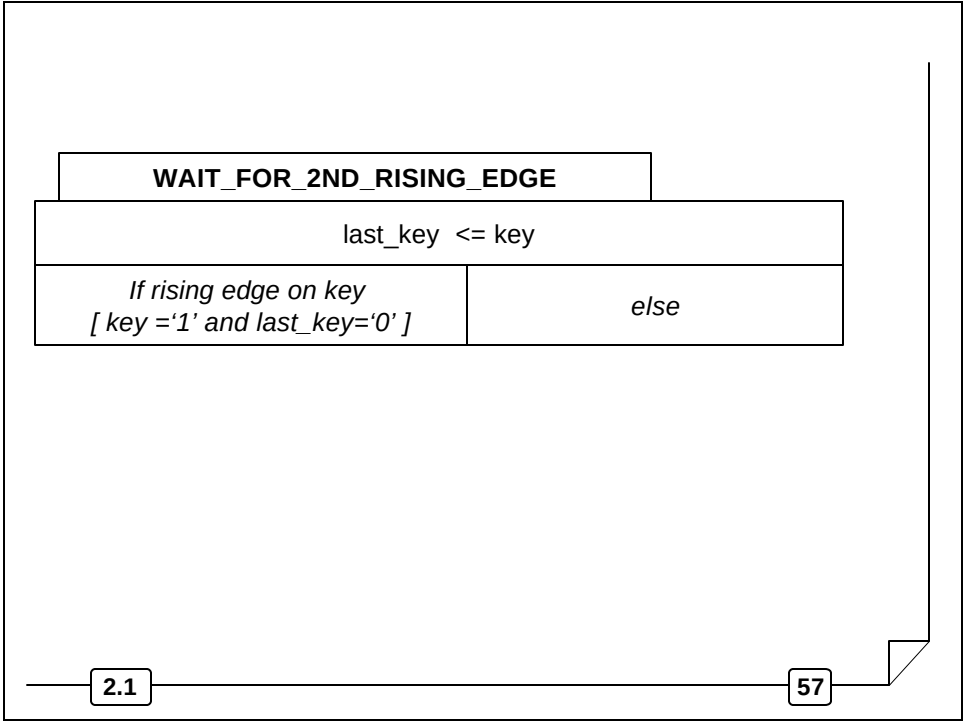
```
last_key <= key  
two_clicks <= '0'  
double_click <= '0'
```

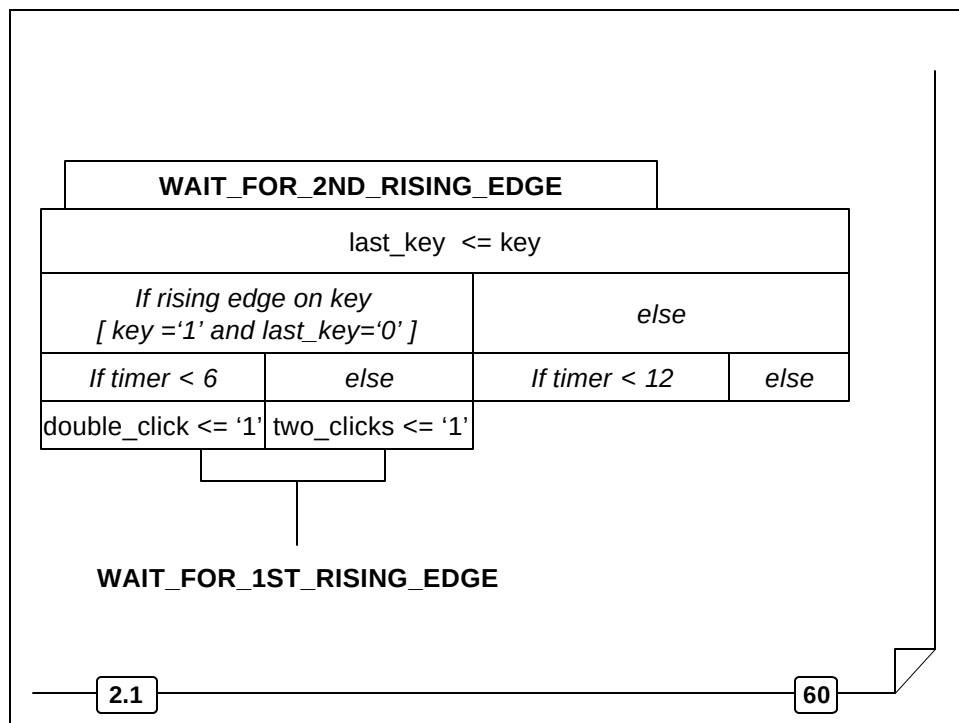
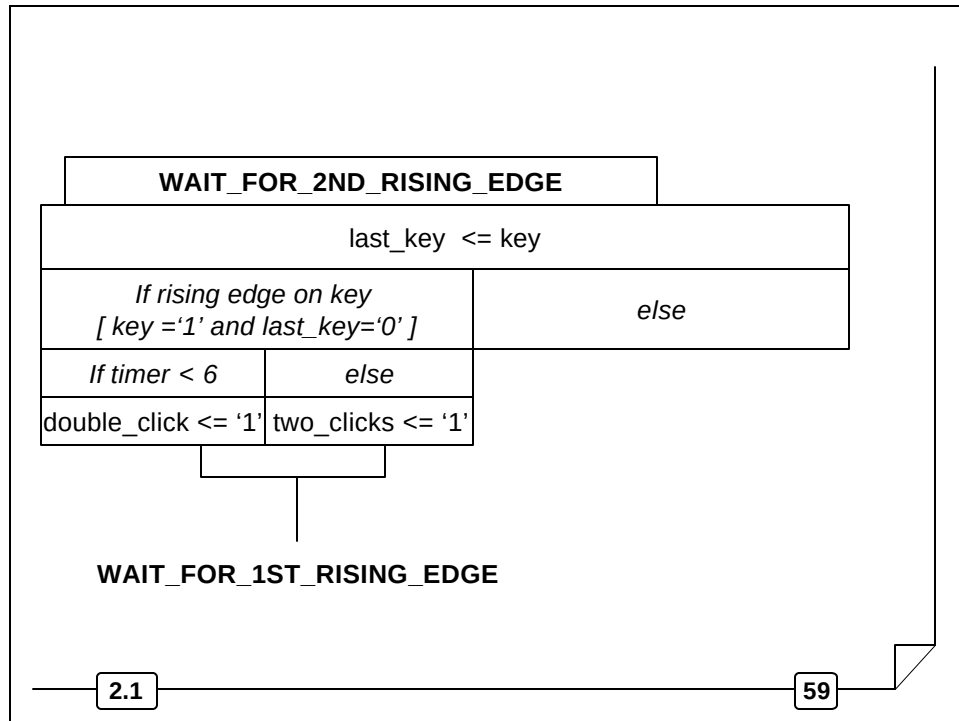
2.1

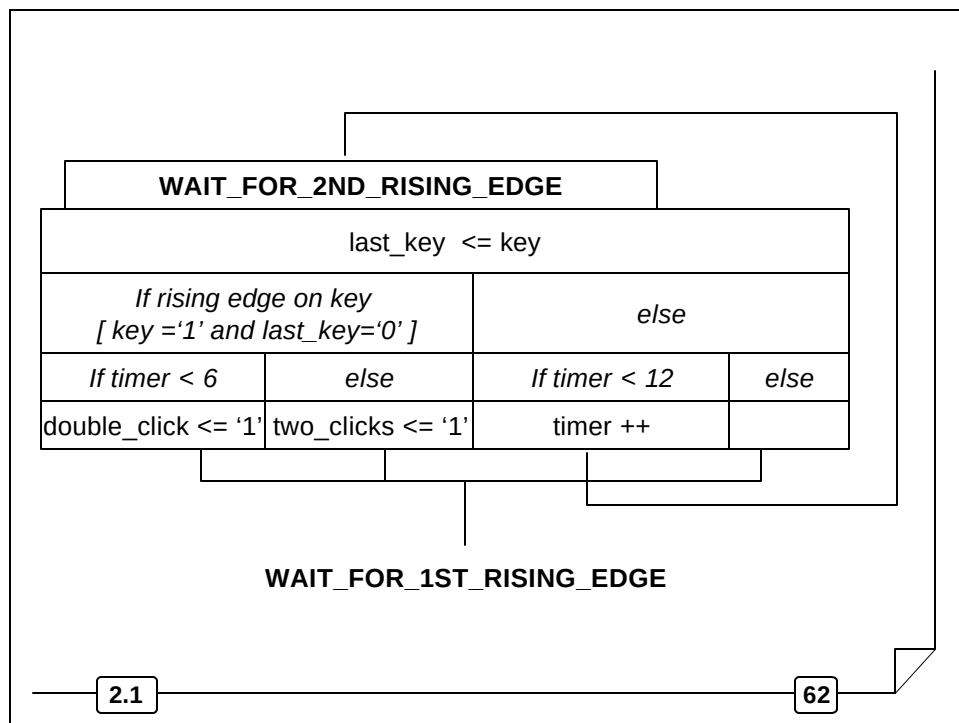
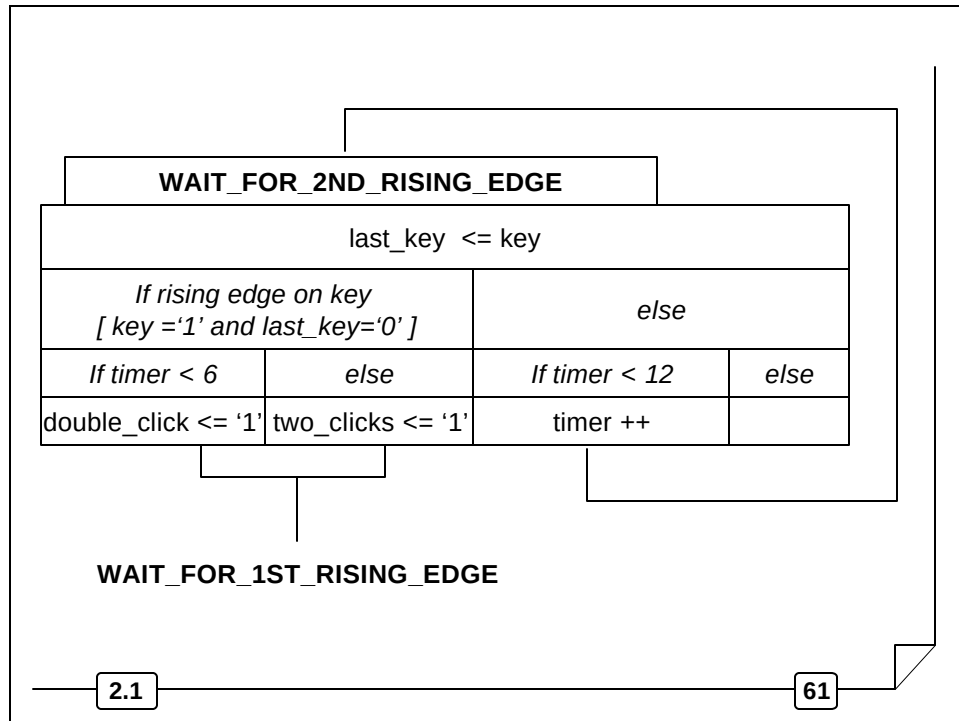
52











Remarks

- The proposed solution requires much less states than a simple FSM
- The reason is that RT-states are more “powerful” in terms of expressiveness than STG-states (e.g., they can include conditions, counters, etc)

2.1

63

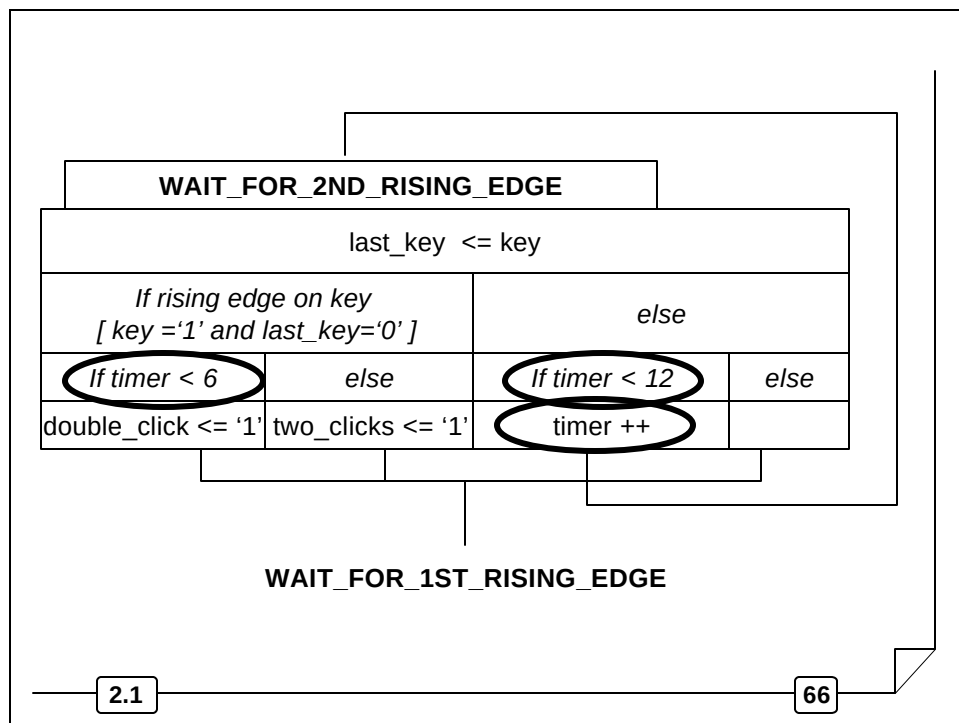
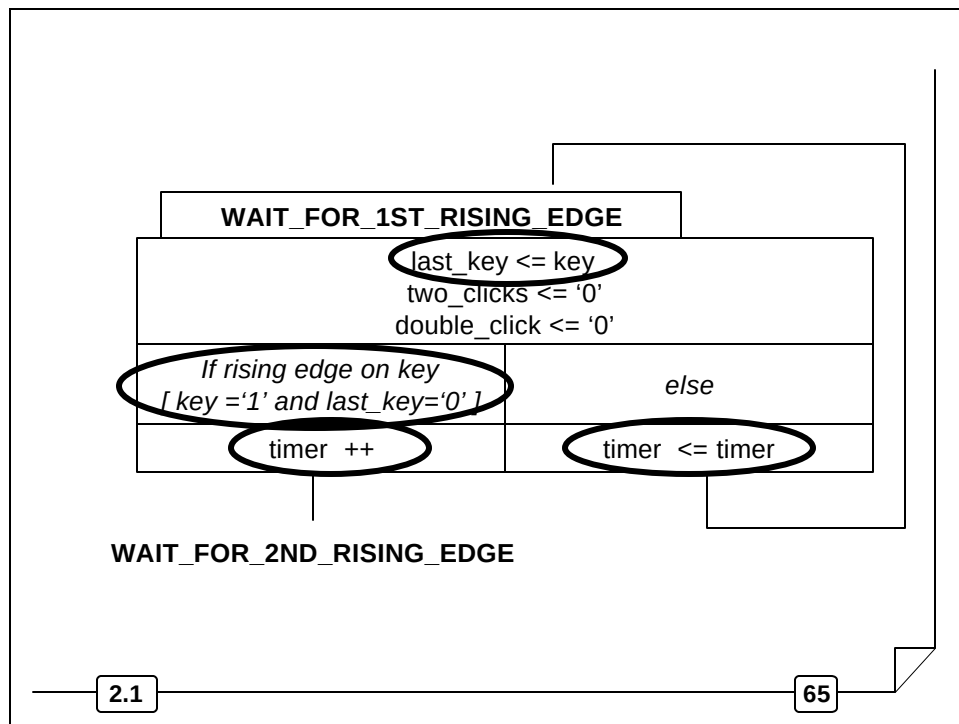
And the Data Path?

The Data Path has to be “extracted” from the RT-states

From the operations performed on the memory elements, we can choose the best RT component to use in the DP

2.1

64



And the Data Path?

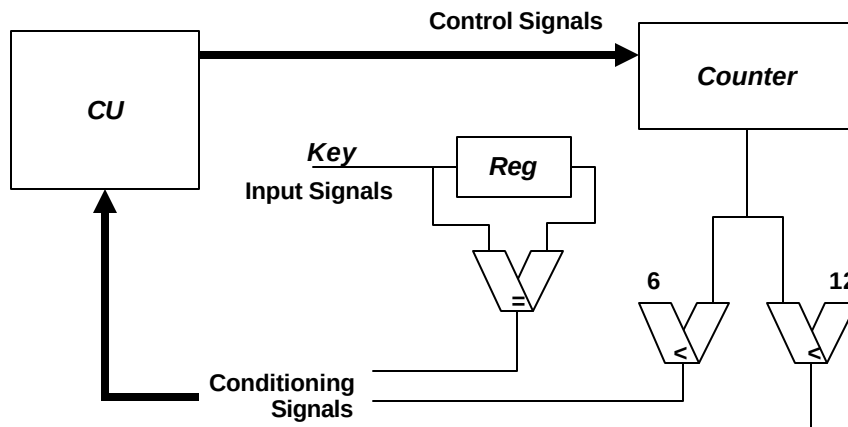
In this case we will need:

- A counter
- Two comparators (<6 and <12)
- A register for the key signal
- A comparator between *key* and *last_key*

2.1

67

The Data Path



2.1

68

Example: sequence checker

odd
even

On a serial transmission line X, bits are transmitted synchronously w.r.t. a clock signal CLK, one bit per clock cycle. The line is used to transmit sequences of 1's and sequences of 0's.

In particular, sequences may have any length, but all the sequences of 1's must contain an odd # of 1's, whereas all the sequences of 0's must contain an even # of 0's.

A circuit to be connected to the serial line is to be designed, such that its output ERR is asserted for one clock cycle whenever a transmission error has been detected.

2.1

69

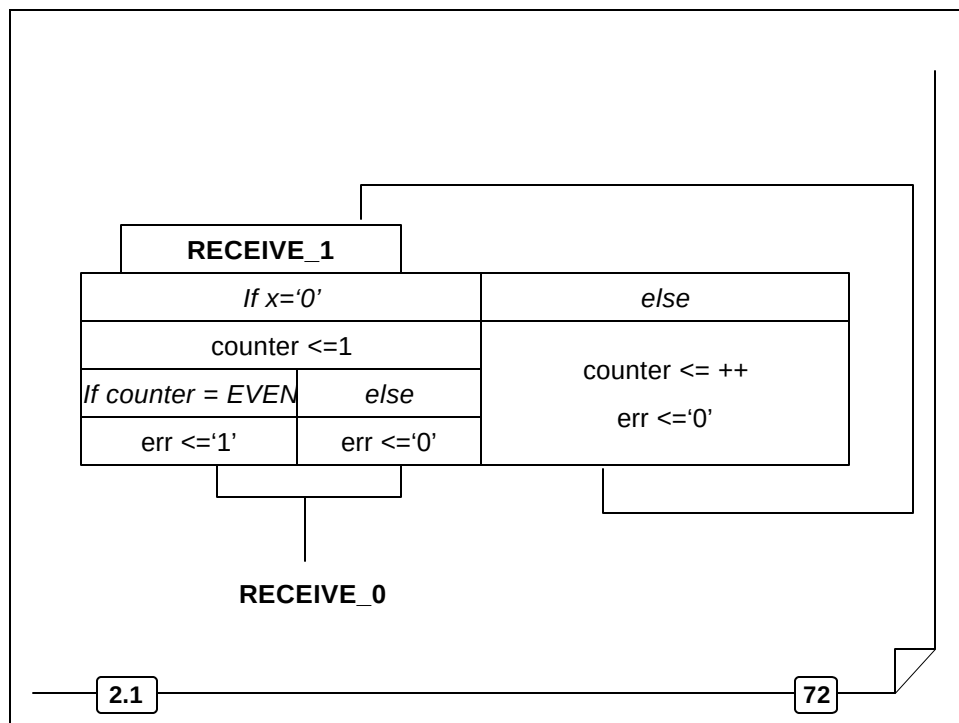
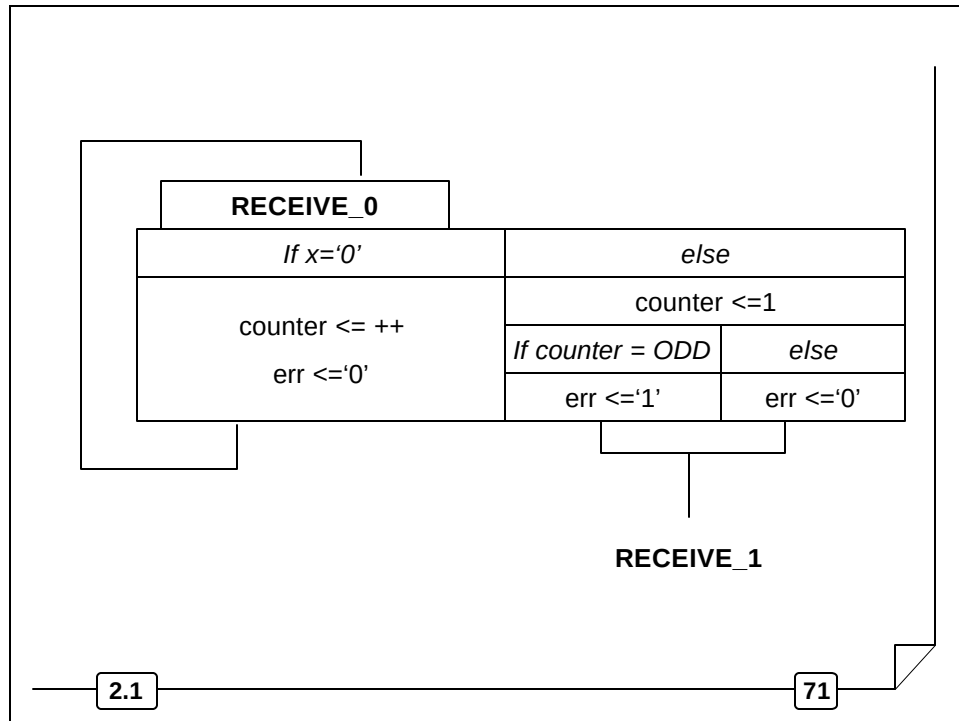
Operations to be performed when the Reset is asserted

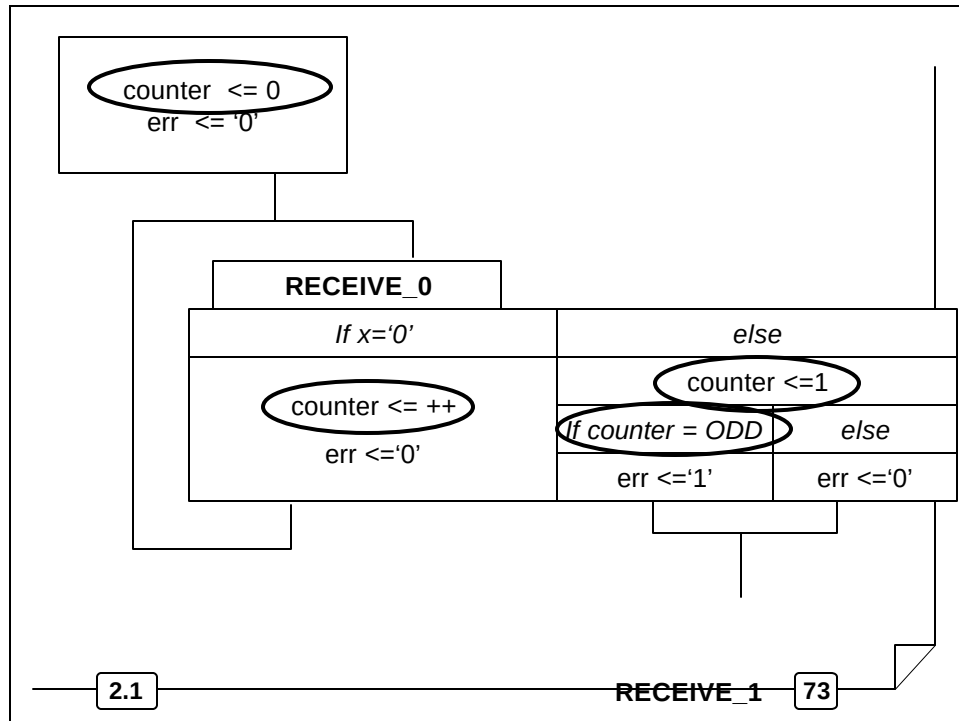
```
counter <= 0  
err <= '0'
```

RECEIVE_0

2.1

70





Data Path

We need the following resources:

- A counter

