

# Finite Statemachines

Specify and Implement dynamic Behaviour

---

Stefan Dennig

November 23, 2025

SkillHub © NewTec GmbH

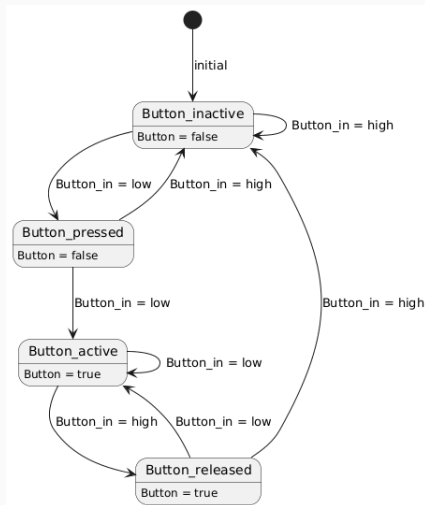
## Example

---

# Example Button Debounce

## Button Debounce Statemachine

- Button Inactive - Button is registered as released
- Button pressed - latch event after first pressed
- Button active - Button is registered as pressed
- Button released - latch event after first release



# Theory

---

## Definition of a Statemachine

A finite-state machine (FSM) is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition.

Parameters of a Statemachine:

- states  $s$
- transitions  $t$
- event  $\sigma$

$$s \in \mathbf{S}$$

## Definition of State

The state of a system is defined by the smallest possible number of its changeable internal parameters.

e.g.:

- velocity
- position
- temperature

Parameters of a Statemachine:

- states  $s$
- transitions  $t$
- event  $\sigma$

$$t \in \mathbf{T}$$

## Definition of Transition

A Transition is a change from one set of parameters of a system to another one. e.g.:

- acceleration
- heat

Parameters of a Statemachine:

- states  $s$
- transitions  $t$
- event  $\sigma$

$$\sigma \in \Sigma$$

## Definition of Event

A Event is a change of inputs of a system. e.g.:

- push of gas pedal
- expiration of time



# Definition

## Finite

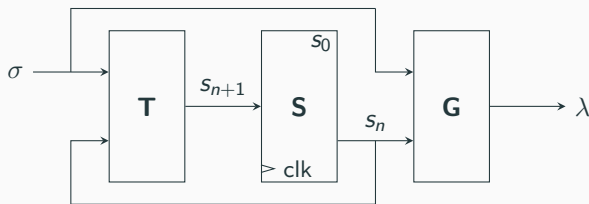
A finite Statemachine only implements a limited set of states **S**. This implicates a limited set of transitions **T** and a limited set of Events **E**

A Statemachine can be described as a set functions  $t$  operating on its internal state  $s$ . The output  $y$  depends on the internal state  $s$ , additionally it may depend on the input  $\sigma$ .

$$s_{n+1} = t(s_n, \sigma) \quad (1)$$

$$\lambda = g(s_n, \sigma) \quad (2)$$

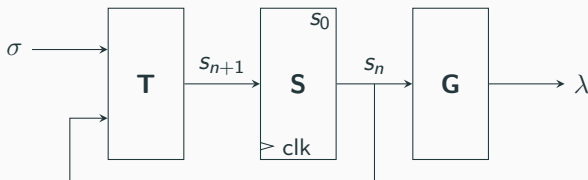
# Mealy Machine



$$s_{n+1} = t(s_n, \sigma) \quad (3)$$

$$\lambda = g(s_n, \sigma) \quad (4)$$

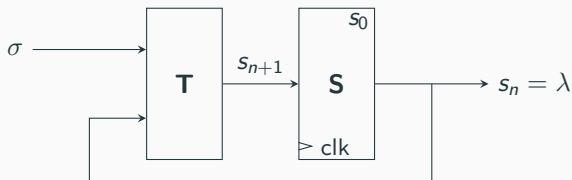
# Moore Machine



$$s_{n+1} = t(s_n, \sigma) \quad (5)$$

$$\lambda = g(s_n) \quad (6)$$

# Medvedev Machine



$$s_{n+1} = t(s_n, \sigma) \quad (7)$$

$$\lambda = s_n \quad (8)$$

- Statemachines, as per Definition, do not imply any timing behaviour.
- Every Memory implicitly holds state, thus implements a statemachine.
- The amount of states is proportional to the complexity of the statemachine.
- Every slightly complex component implements a statemachine.
- There can be multiple statemachines in a system.

# Design of Statemachines

When to use a statemachine?

- Simple Behaviour that can be clustered
- Strict Deterministic control
- Ressource/Device Control

When not to use a statemachine?

- If a combinational logic is an alternative
- If a mathematical formula can be used instead (filters)
- Multiple Statemachines should never implement the same state

## Senior Experience

State should be avoided.

- Decide about timing (cyclic, synchronous, asynchronous)
- Number of states shall be reduced
- Hierarchical Statemachines are a good way to reduce states
- Statemachines that are directly or indirectly linked shall be avoided
- Make states explicit (no rotation direction as variable, no global variables)
- If a component implies state implement it as a statemachine (init, running, error)
- shall invalid transitions be neglected or treated as errors?

- How many states does a Integer(8-Bit) hold?
- How many states can a microcontroller with 2 KBytes of RAM and 16KBytes of ROM have?
- What is the maximum amount of transitions in a state machine with 4 States?

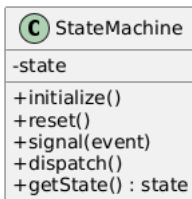


- How many states does a Integer(8-Bit) hold? **256**
- How many states can a microcontroller with 2KBytes of RAM and 16KBytes of ROM have?  **$2^{18000}$**
- What is the maximum amount of transitions in a state machine with 4 States? **16**

# Software Specification

---

# Interface of Statemachine Component



Interface	Description	Implements
initialize()	Initializes a statemachine	$s_n = s_0$
reset()	Sets a Statemachine back to Initial State	$s_n = s_0$
signal(e)	sets the event of the state machine	$\sigma = e$
dispatch()	executes the state transition (clock)	$s_{n+1} = t(s_n, \sigma)$
getState() : $\lambda$	accesses the output	$\lambda = g(s_n, \sigma)$

# Features of Statemachine Component

- synchronous asynchronous dispatch (clock)
- use guard-function
- no reaction or transition or entry-function/exit-function
- use do-function
- use use output logic
- treat non-transition events

The more features you want the more complex the solution will be.

## Senior Experience

Complexity increases Risk.

# Synchronous/Asynchronous Dispatch

## Synchronous Dispatch

When dispatching synchronously, the state transition is executed at the moment the event is created. This implicitly means the transition occurs at state transition.

### Challenges:

- transition can occur at any time
- rapid state changes possible
- nested state changes possible

## Asynchronous Dispatch

When dispatching asynchronously, the state transition is executed delayed to the moment of event creation. This implicitly means the transition occurs at state transition.

### Challenges:

- events need to be memorized until dispatch
- decision necessary: queued events or dumped events
- event queue overrun or lost events possible

# Guard Function

## Guard Function

A Guard Function is used to verify if a transition shall be performed. It overwrites the behaviour of a state machine.

Guard functions overwrite the defined behaviour of a state machine.  
Thus they implicate a Exception.

## Senior Experience

Exceptions shall be avoided.

# Reaction to state change

## No Reaction

If only state needs to be tracked it may not be needed to act on state changes.

## Transition Function

A transition function can be used create behaviour that changes the state of the system.

## Entry/Exit Function

It may be possible to split a transition function. The exit-function specifies the change out of a state. The exit-function specifies the change into of a state. Thus, entry- and exit functions are state related.

Challenge: entry and exit functions imply a state that is between all the states. This state is entered by the exit function and left by the entry function.

## **Do Functions**

Do Functions define Behaviour to be done in a state.

Challenge: Using do functions usually conflict with the scheduler. I prefer changing a schedule table in a entry/exit function.



## **Do Functions**

Do Functions define Behaviour to be done in a state.

Challenge: Using do functions usually conflict with the scheduler. I prefer changing a schedule table in a entry/exit function.

## Output Logic

The Output logic can be implemented inside the statemachine, using hooks (function pointers). For generic implementations this increases complexity.

Challenge: I prefer implementing the Output logic as a wrapper around the statemachine.

## Non-Transition Events

Not every Event must lead to transition in any state. There must be a decision whether such events need to be treated (Error-Handling).

This decision is strictly use-case dependent. Personally i try to avoid error-handling in Non-Transition Events. Alternatively, if all defined events have defined transitions in every state, this issue can be avoided by design completely.

# Software Implementation

---

# Implementation of Statemachine

- At the core a statemachine is basically a nested switch statement, thats a good way for dealing with small non-generic implementations.
- For handling more complex or generic implementations a table based approach may be better.

## Senior Experience

(Nested) Switch statements are difficult to maintain.

# Implementation of a generic StateMachine

