

Embedded System Architecture - CSEN 701

Module 4: ES Modeling & Design **Lecture 08: System Modeling using FSMs**

Dr. Eng. Catherine M. Elias

catherine.elias@guc.edu.eg

***Lecturer, Computer Science and Engineering,
Faculty of Media Engineering and Technology, German University in Cairo***

- System Modeling
- Event-driven Discrete Model
- Finite State Automata
- Example
- Conclusion



The Real-time Embedded System

ES Modeling & Design (2 Modules)

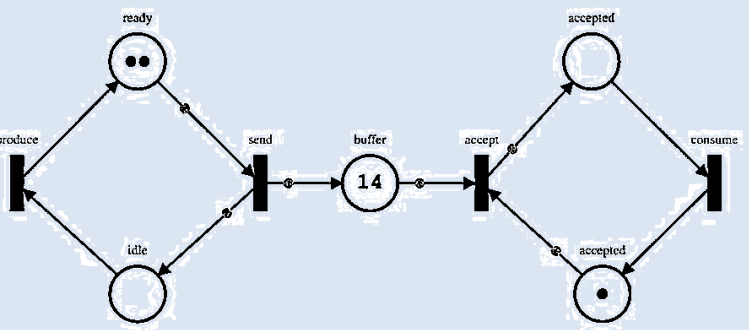
System Modeling

Let's go

Design Considerations

Power management and optimization

Reliability and fault tolerance



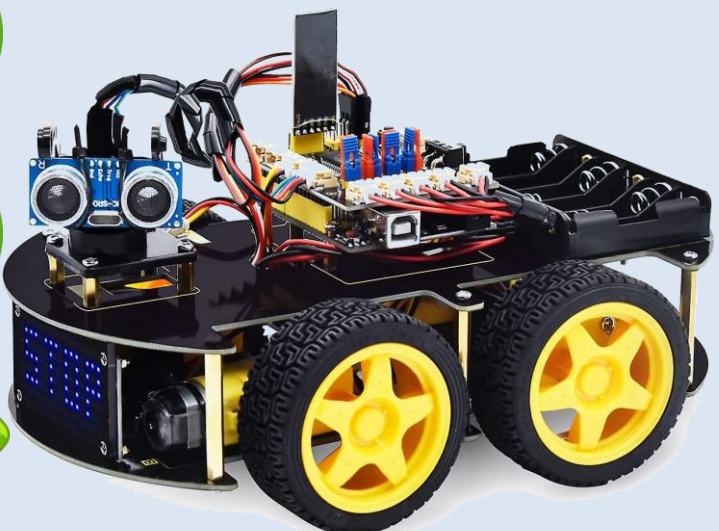
ES Hardware Components (4 Modules)

Microcontroller
Fundamentals

Embedded programming
languages

Embedded Hardware

Communication and
Networking



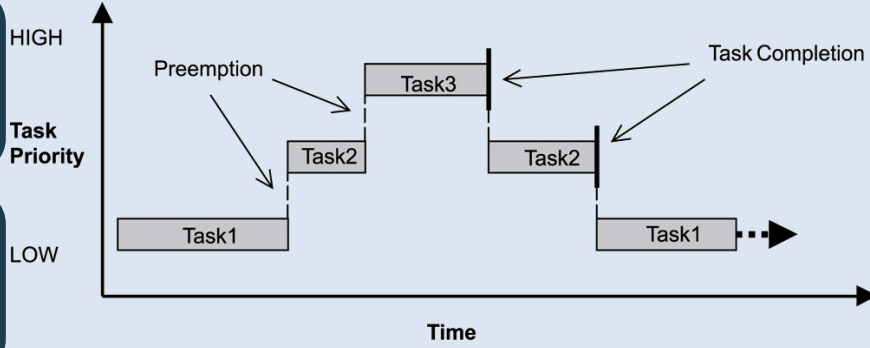
ES Software Components (1 Module)

Real-Time Systems

Multi-tasking

Scheduling

Resource Management



Embedded System Tools & Software Development (2 Modules)

Debugging techniques

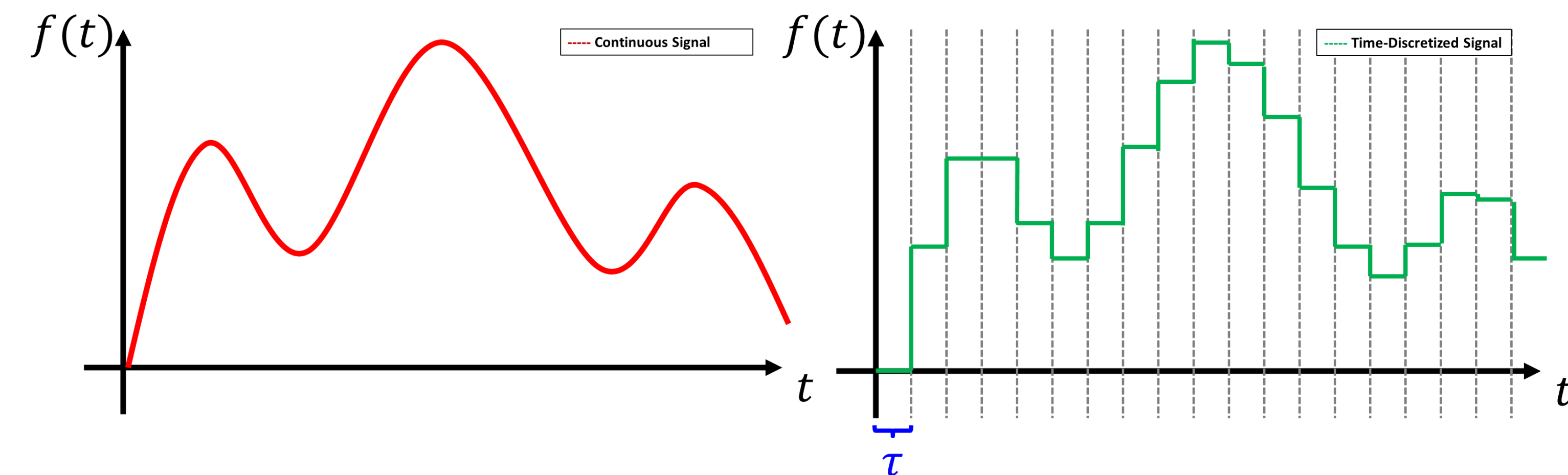
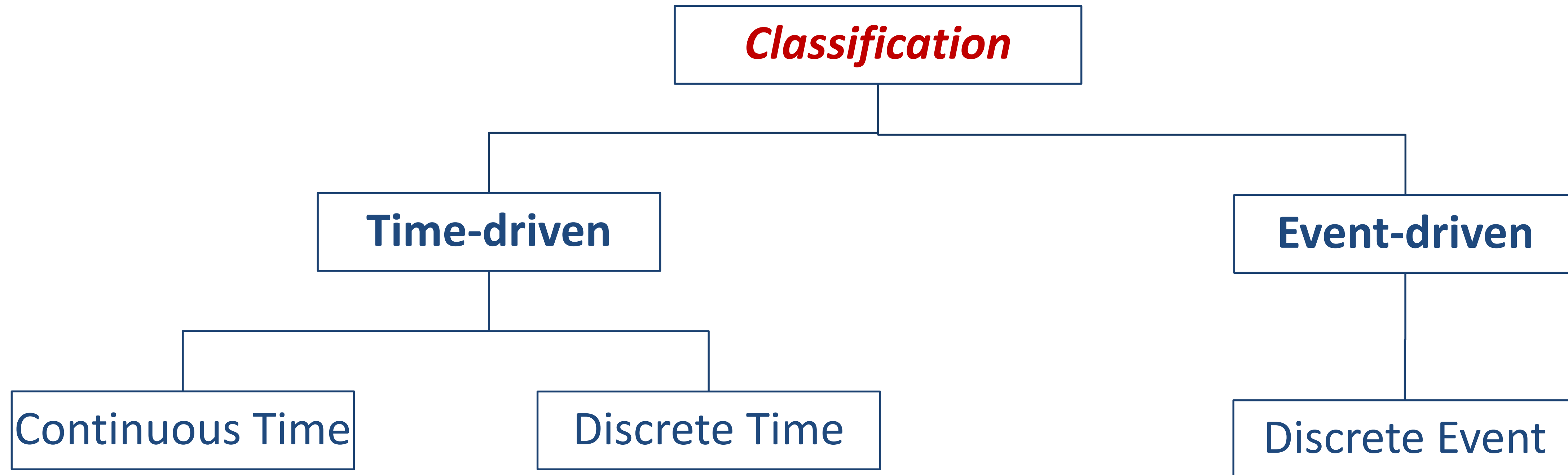
Interrupts and exception
handling

Memory management



Definition

- System modeling refers to the process of creating a representation or model of a real-world system or physical phenomenon using mathematical, graphical, or computational techniques.
- This type of modeling is typically used in engineering, physics, and other scientific disciplines to understand, analyze, and predict the behavior of physical systems.



Definition

- An event-driven discrete model of a physical system is a **modeling approach** that represents a **real-world physical system** as a discrete event-driven simulation.
- In this type of modeling, the **behavior** of the physical system is **discretized in time**, and **changes in the system's state** occur in **response** to **specific discrete events**.
- These events may include:
 - External stimuli,
 - Sensor readings,
 - Control signals, or
 - other significant occurrences that influence the system's behavior.

How to model?

1. Define the System:

- Clearly define the physical system you want to model, including its **behaviors**, **inputs**, **outputs**, and **key parameters**. → *System Description*

2. Identify Events:

- Identify the **events** (triggers/inputs) that will drive changes in the system's behavior.

Techniques

Finite State Automata / Machine (FSA/FSM)

Petri-Nets

Statecharts

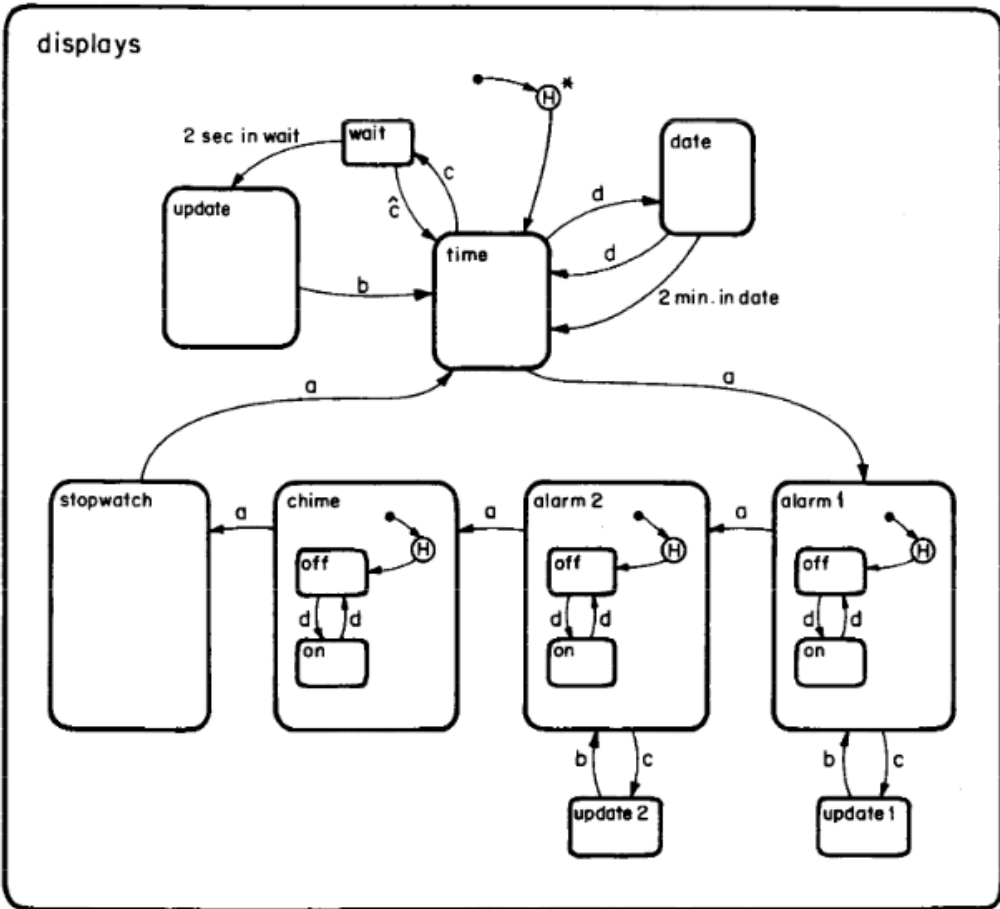
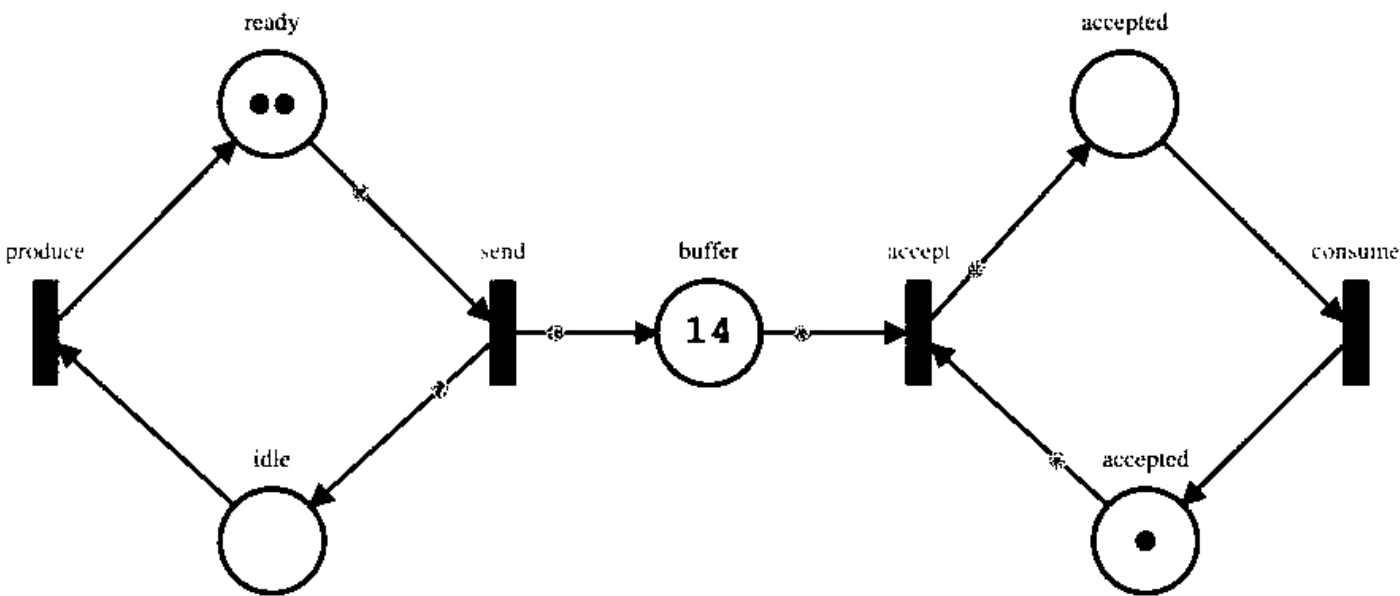
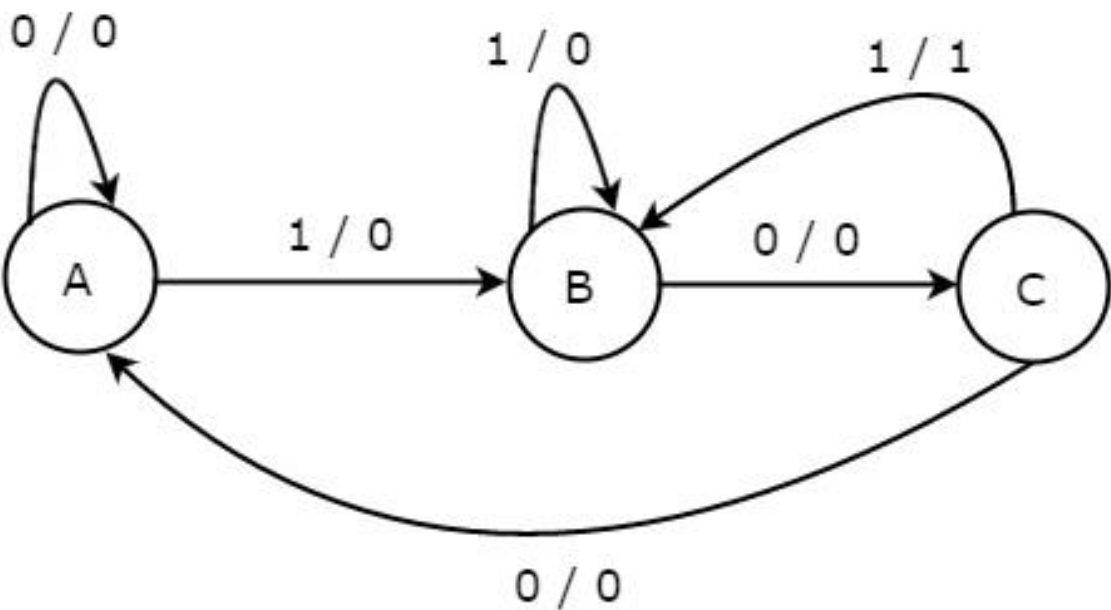


Fig. 13.

Definition

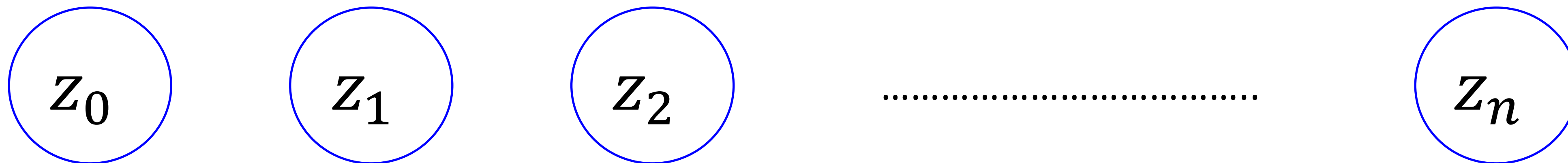
- A finite state automaton, also known as a finite state machine (FSM) or finite automaton, is a **mathematical model** and **an abstract computational device** used to represent and analyze the behavior of systems that can be in a limited number of distinct states and undergo transitions between those states in response to inputs.

Key Characteristics

- 1. States:** An FSM consists of a finite set of states. Each state represents a distinct condition or configuration of the system.

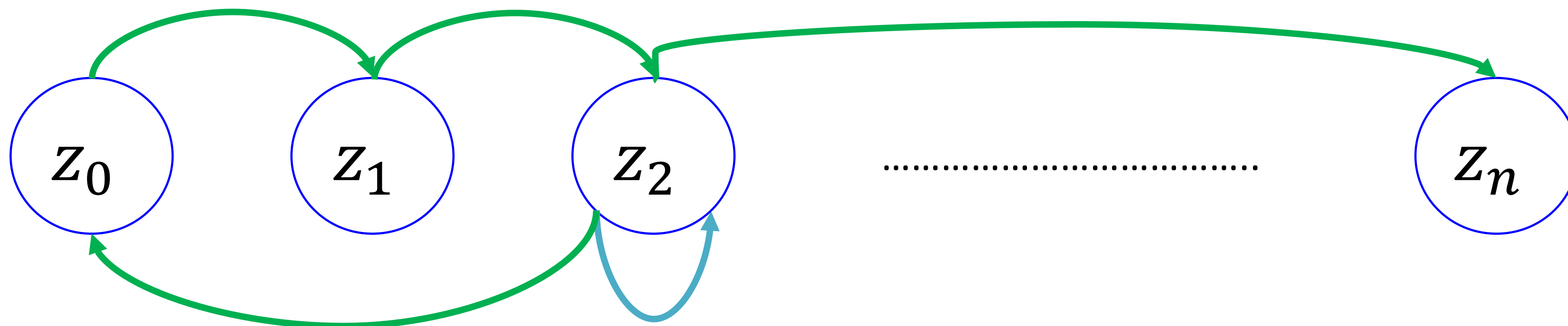
$$Z = \{z_0, z_1, z_2, \dots, z_n\} \text{ where } n \text{ is the number of states}$$

It is important to note that the state is represented by a **circle** in the graphical FSM model



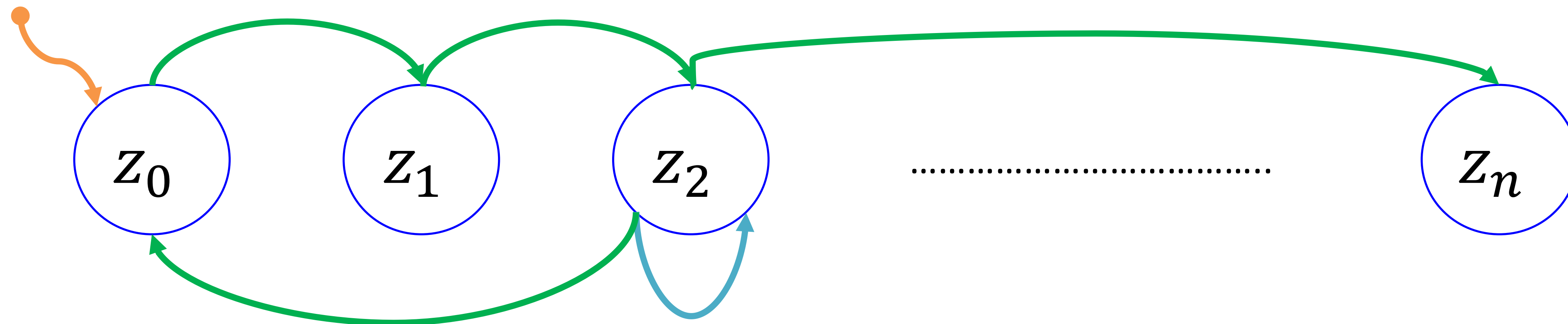
Key Characteristics

- 2. Transitions:** Transitions define how the system can move from one state to another.
- Transitions are often triggered by specific inputs or events.
 - Each transition is labeled with the input or event that causes the state change.
 - It is important to note that the transition is represented by an **arrow** that connects 2 states
 - One important type of transitions is the **self-transition** that allows the system to idle in a desired state.



Key Characteristics

- 3. Initial State z_0 :** One state is designated as the initial state, which represents the system's starting point.
- In order to identify the initial state on the graph, we add an **initial transition**.

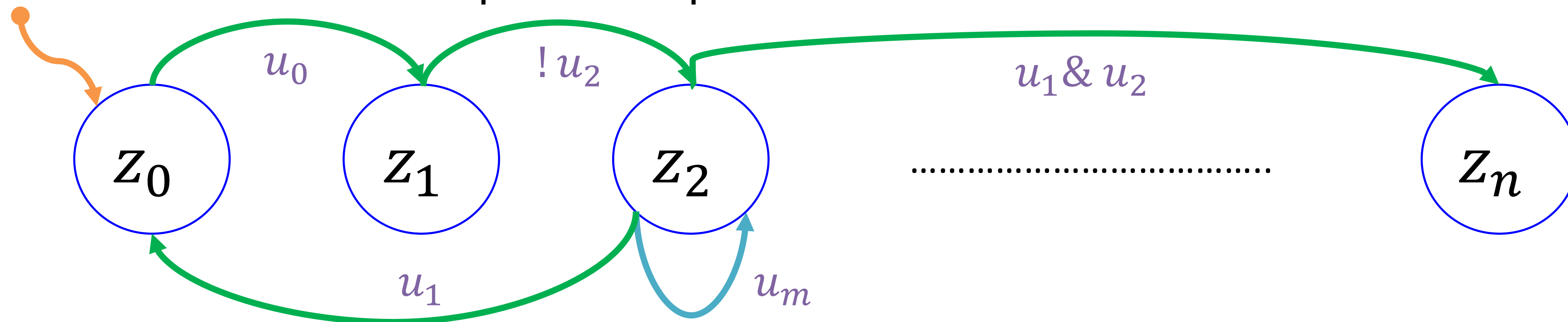


Key Characteristics

- 4. Inputs/ triggers/ guard set:** The system receives a sequence of inputs defined in a guard set.

$$U = \{u_0, u_1, u_2, \dots, u_m\} \text{ where } m \text{ is the number of inputs}$$

- Each input may lead to a **transition** from one state to another.
- Note that these inputs are considered a binary value $u_i \in \{0,1\}$.
- It is important to make sure that the machine is **well-defined** that means you already added a transition that cover all possible inputs for each state.

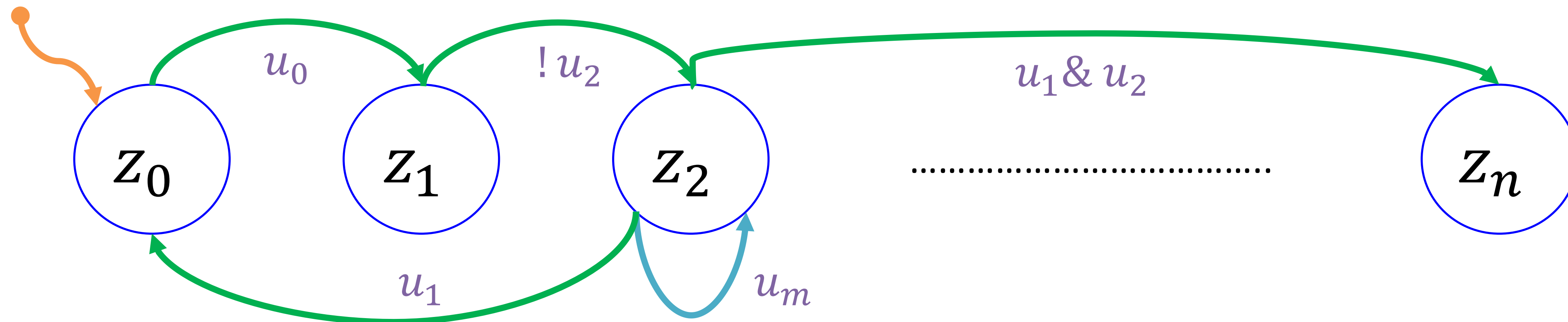


Key Characteristics

5. Outputs/ actions: Regardless of the FSM type, an action should be made based on the previous state and the input trigger out of the output set

$$Y = \{y_0, y_1, y_2, \dots, y_l\} \text{ where } l \text{ is the number of outputs}$$

- Note that these outputs are also considered a binary value $y_i \in \{0,1\}$.
- The placing of the output on the machine identifies its **type** and also identifies the **sequence of execution** in each time sample k .



Mealy Machine

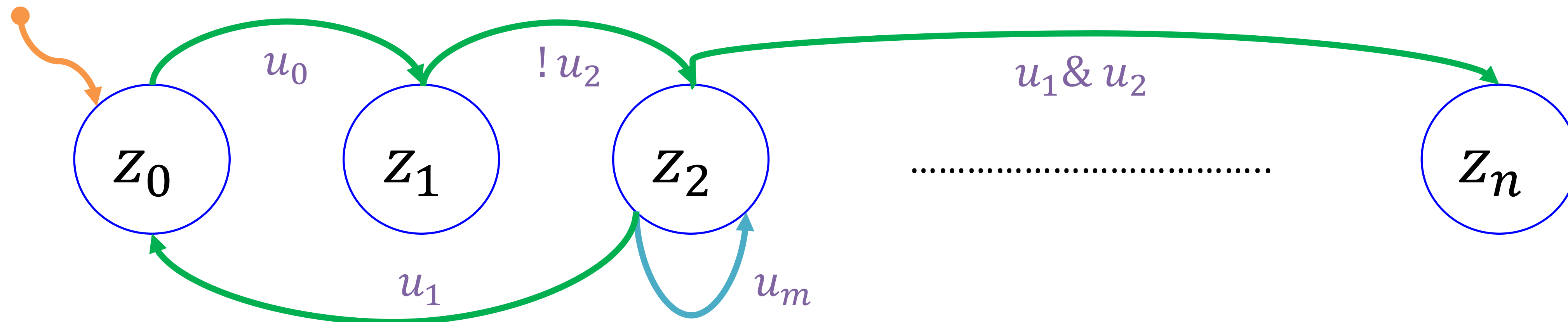
- Named after George H. Mealy, who presented the concept in a 1955 paper, A Method for Synthesizing Sequential Circuits.
- A Mealy machine A_{mealy} is a tuple (ordered set):

$$A_{mealy} = (Z, U, Y, f, g, z_0)$$

f is the **transition function** $f: Z \times U \rightarrow Z$ or in other format $Z_{k+1} = f(Z_k, u_k)$

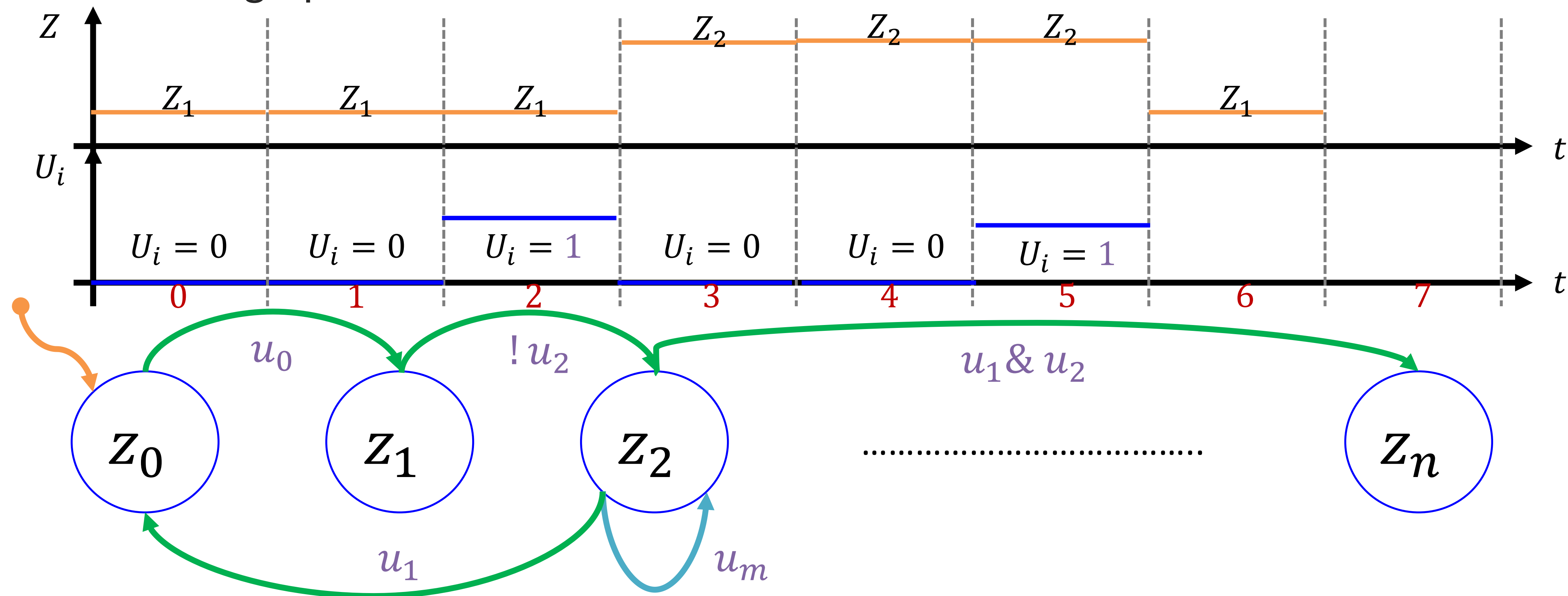
g is the **output function** $g: Z \times U \rightarrow Y$ or in other format $Y_k = g(Z_k, u_k)$

Note: the suffix denotes the time sample which indicates the sequence of execution.



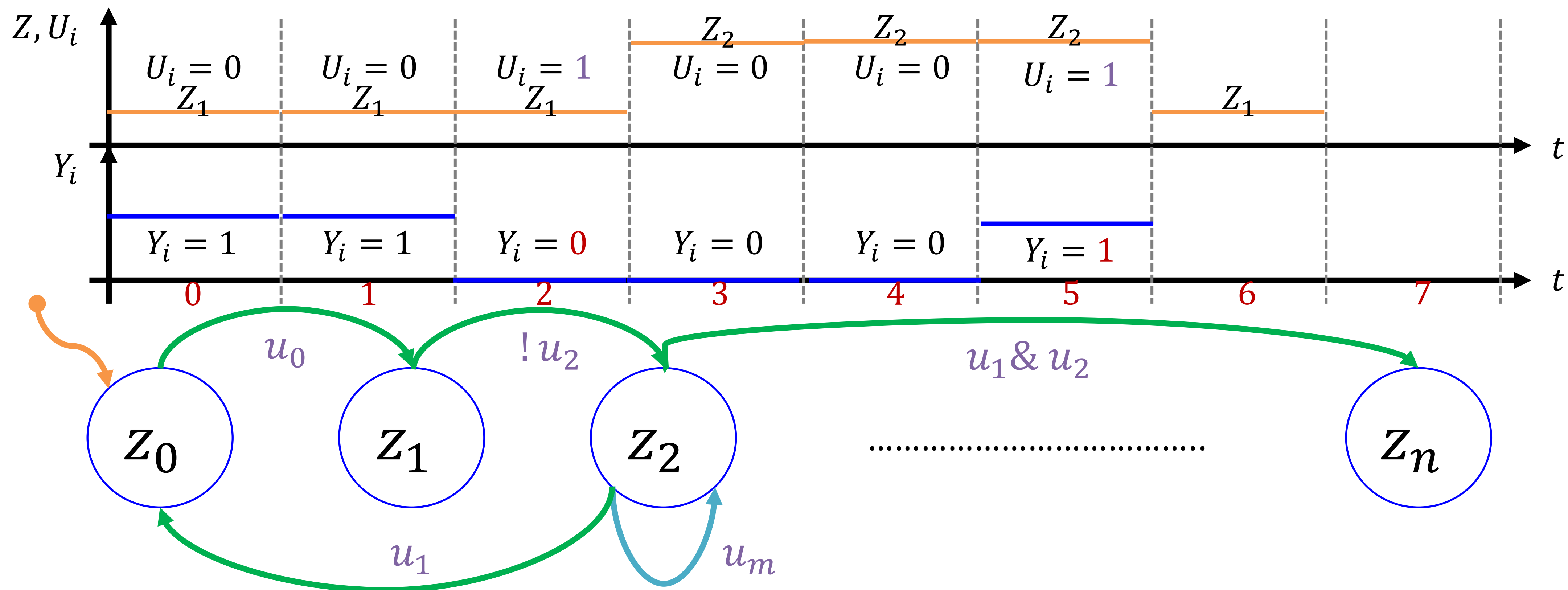
Mealy Machine

- From the **transition function** $Z_{k+1} = f(Z_k, u_k)$, we can understand that the new state at sample $k + 1$ is dependent on the previous state and input at time sample k as shown in the execution graph



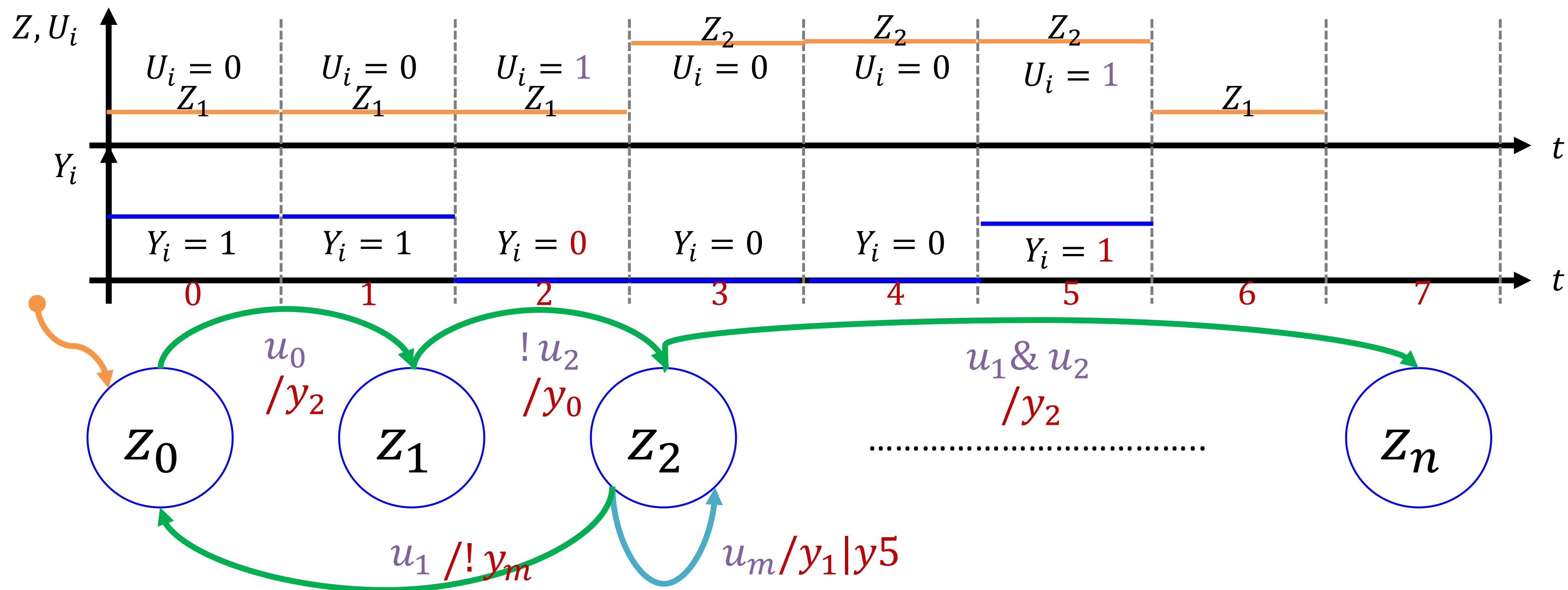
Mealy Machine

- From the **output function** $Y_k = g(Z_k, u_k)$, we can understand that the current output at sample k is dependent on the current state and input as shown in the execution graph.



Mealy Machine

- From the **output** execution graph, we can observe that the input and output are executed in the same clock cycle. That means that they are generated together. Accordingly, they are on the **same transition**.



Moore Machine

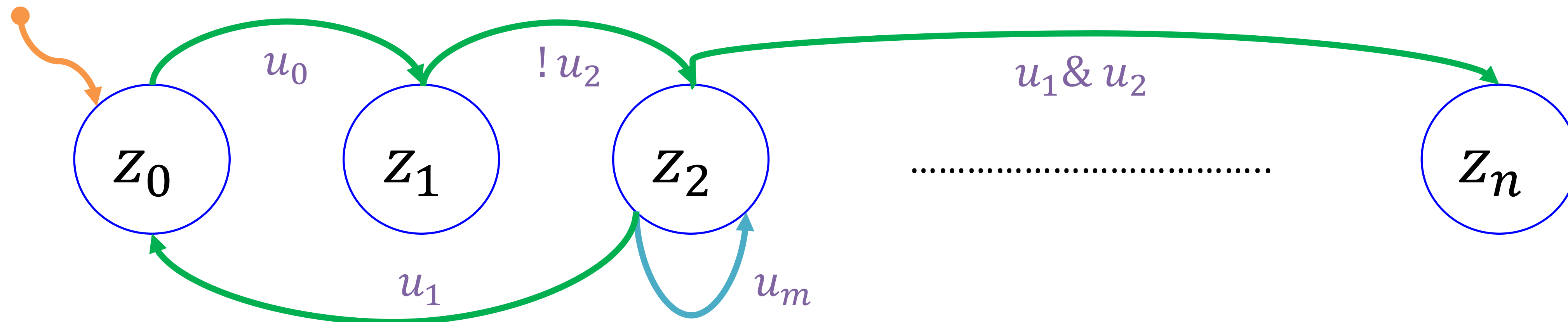
- Named after Edward F. Moore, who presented the concept in a 1956 paper, Gedanken-experiments on Sequential Machines.

- A Moore machine A_{moore} is a tuple (ordered set):

$$A_{moore} = (Z, U, Y, f, g, z_0)$$

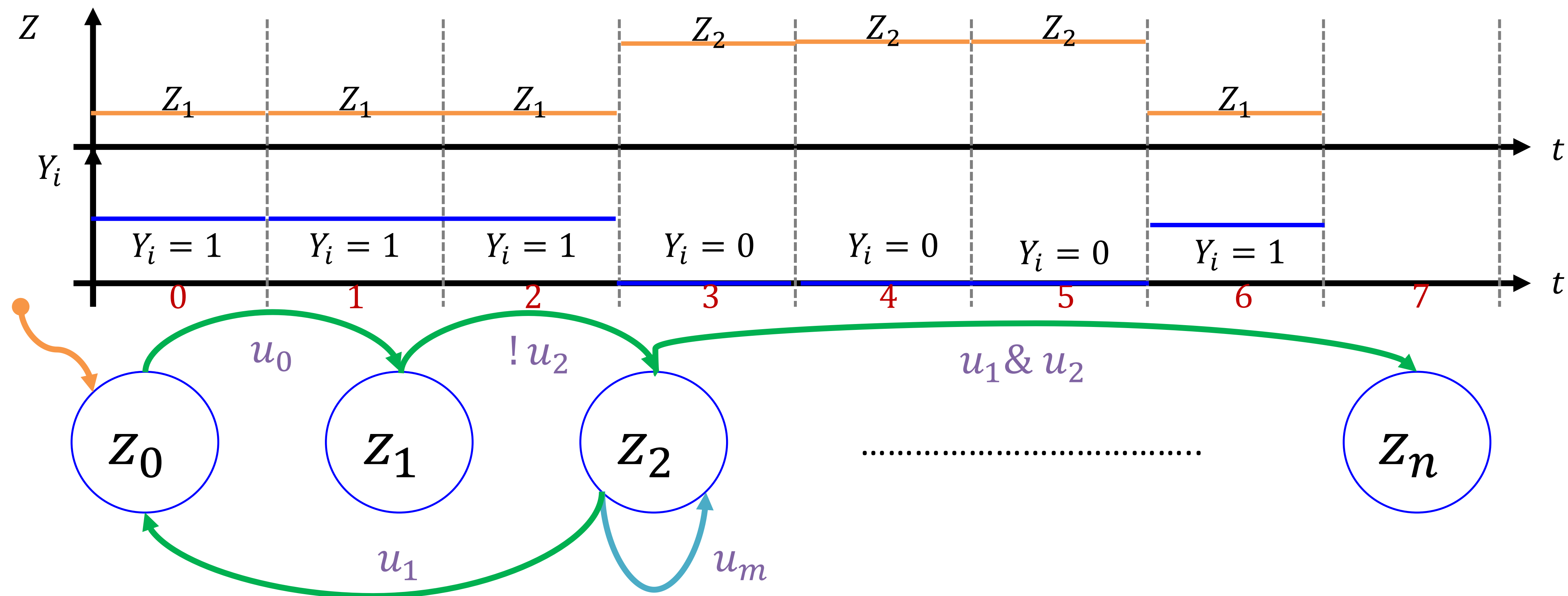
- The machine is identical to a Mealy, except that the output **only depends on the state**:

g is the **output function** $g: Z \rightarrow Y$ or in other format $Y_k = g(Z_k)$



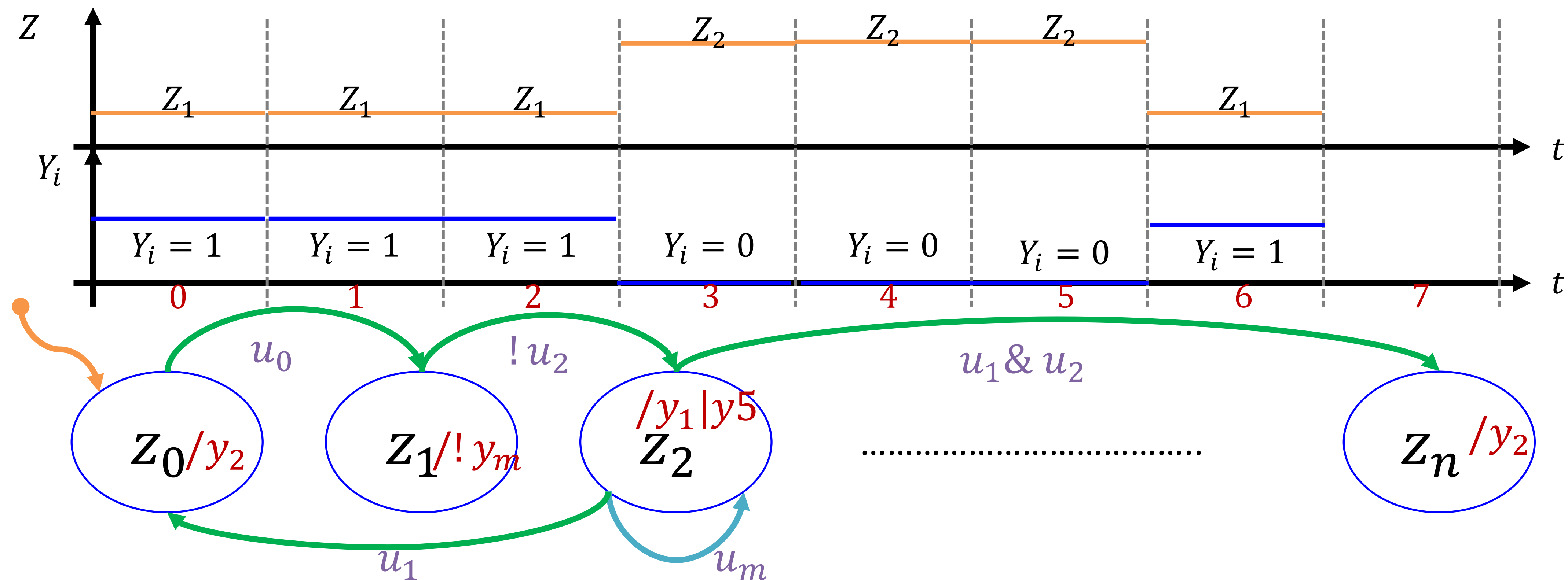
Moore Machine

- From the **output function** $Y_k = g(Z_k)$, we can understand that the current output at sample k is dependent on the current state and input as shown in the execution graph.



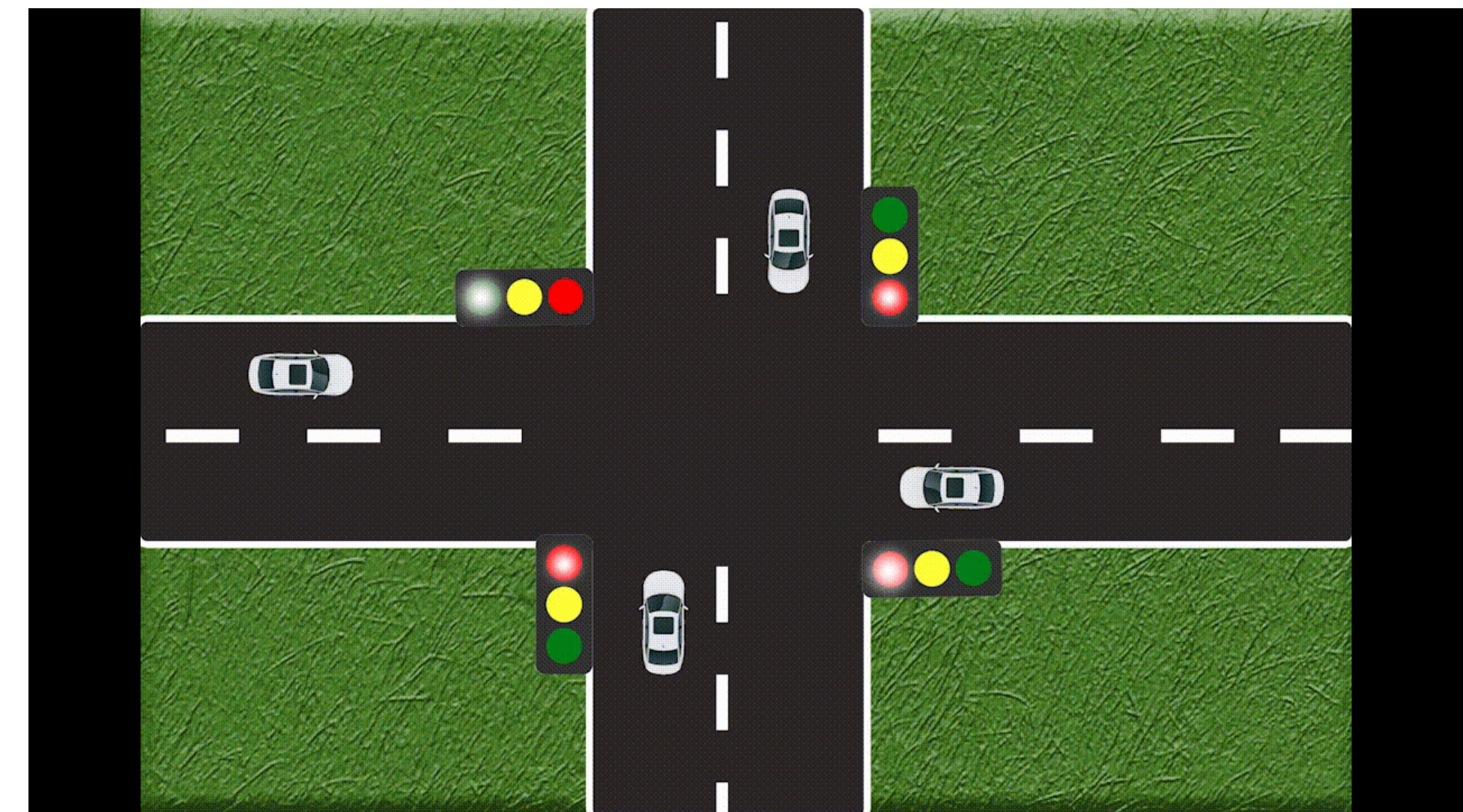
Moore Machine

- From the **output** execution graph, we can observe that the state and output are executed in the same clock cycle. That means that they are generated together. Accordingly, they are on the **same state**.



Traffic Light Controller: *Description*

- In this example, we will explore how Finite State Machines (FSMs) can be used to model a Traffic Light Controller. Traffic light systems are a ubiquitous part of our daily lives, and they provide an excellent context for understanding how FSMs can be applied to real-world embedded systems.
- Imagine a standard traffic intersection with traffic lights controlling the flow of vehicles and pedestrians. The system's goal is to manage the traffic efficiently and safely by providing the right-of-way to various directions of traffic. The traffic light controller's task is to switch between red, green, and yellow lights for each direction, following a specific sequence.



Traffic Light Controller: *Solution*

- States:

1. Idle $\rightarrow z_0$
2. North-South **Green**, East-West **Red** $\rightarrow z_1$
3. North-South **Yellow**, East-West **Red** $\rightarrow z_2$
4. North-South **Red**, East-West **Green** $\rightarrow z_3$
5. North-South **Red**, East-West **Yellow** $\rightarrow z_4$

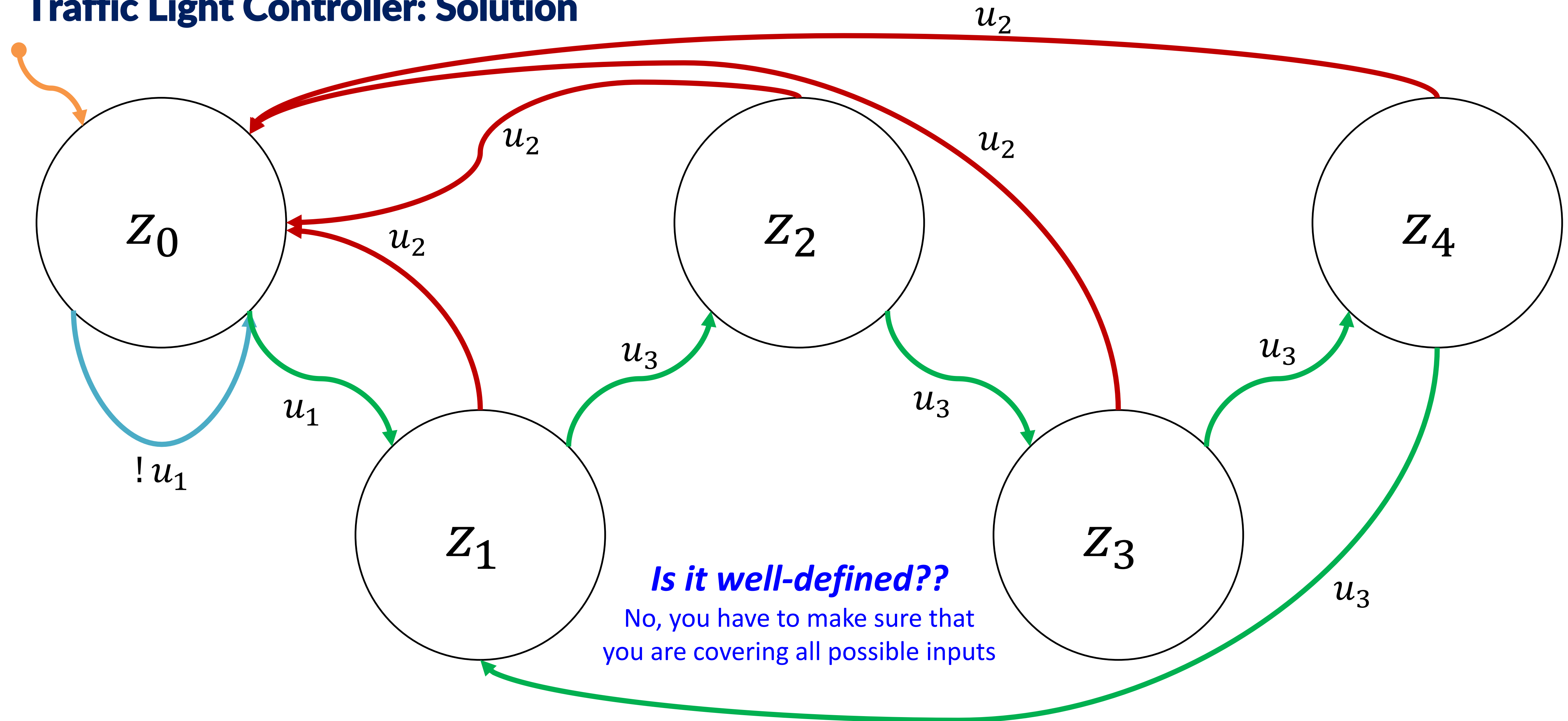
- Inputs:

1. Start Button $\rightarrow u_1$
2. Reset Button $\rightarrow u_2$
3. Timer $\rightarrow u_3$

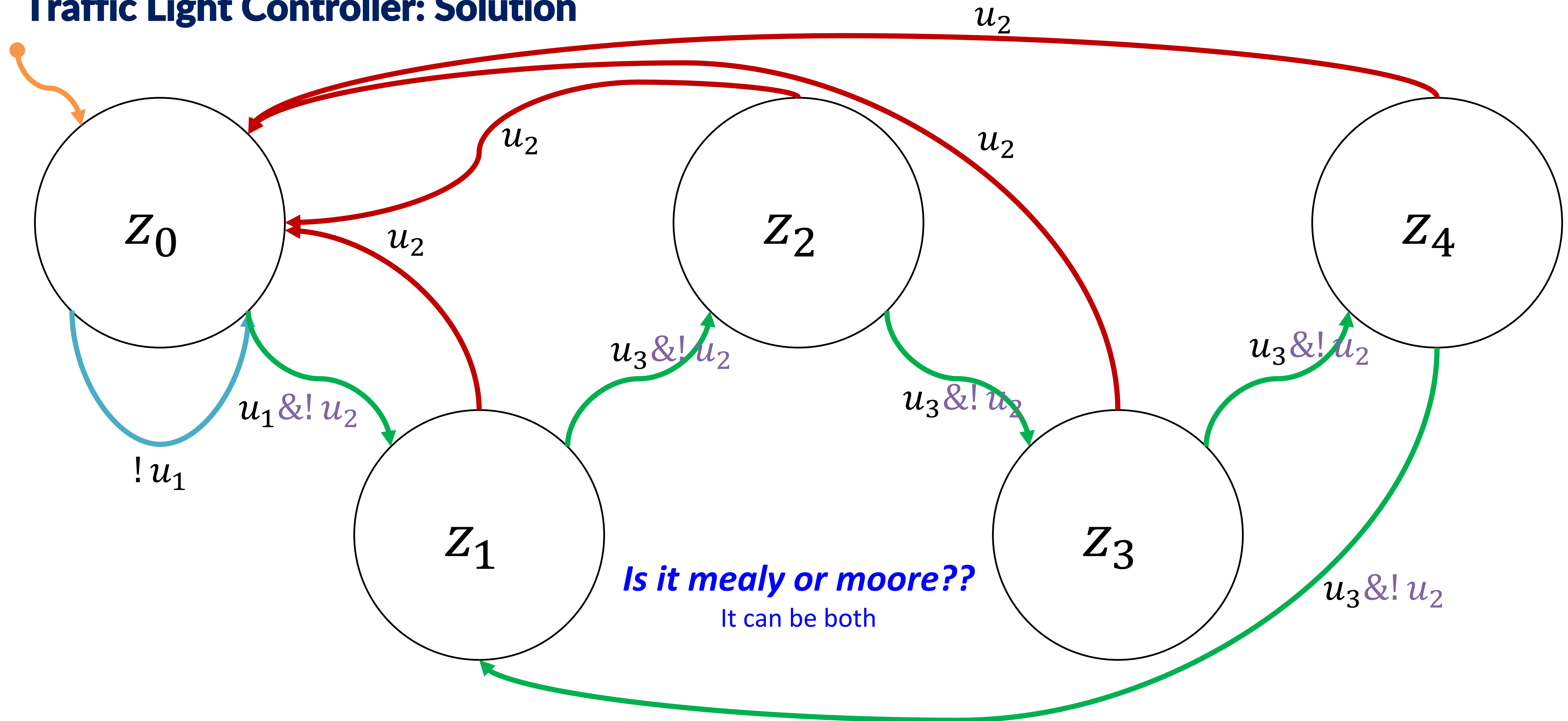
- Outputs:

1. 3 RGY lambs for North-South traffic light
 $\rightarrow y_1, y_2, y_3$
2. 3 RGY lambs for , East-West traffic light $\rightarrow y_4, y_5, y_6$

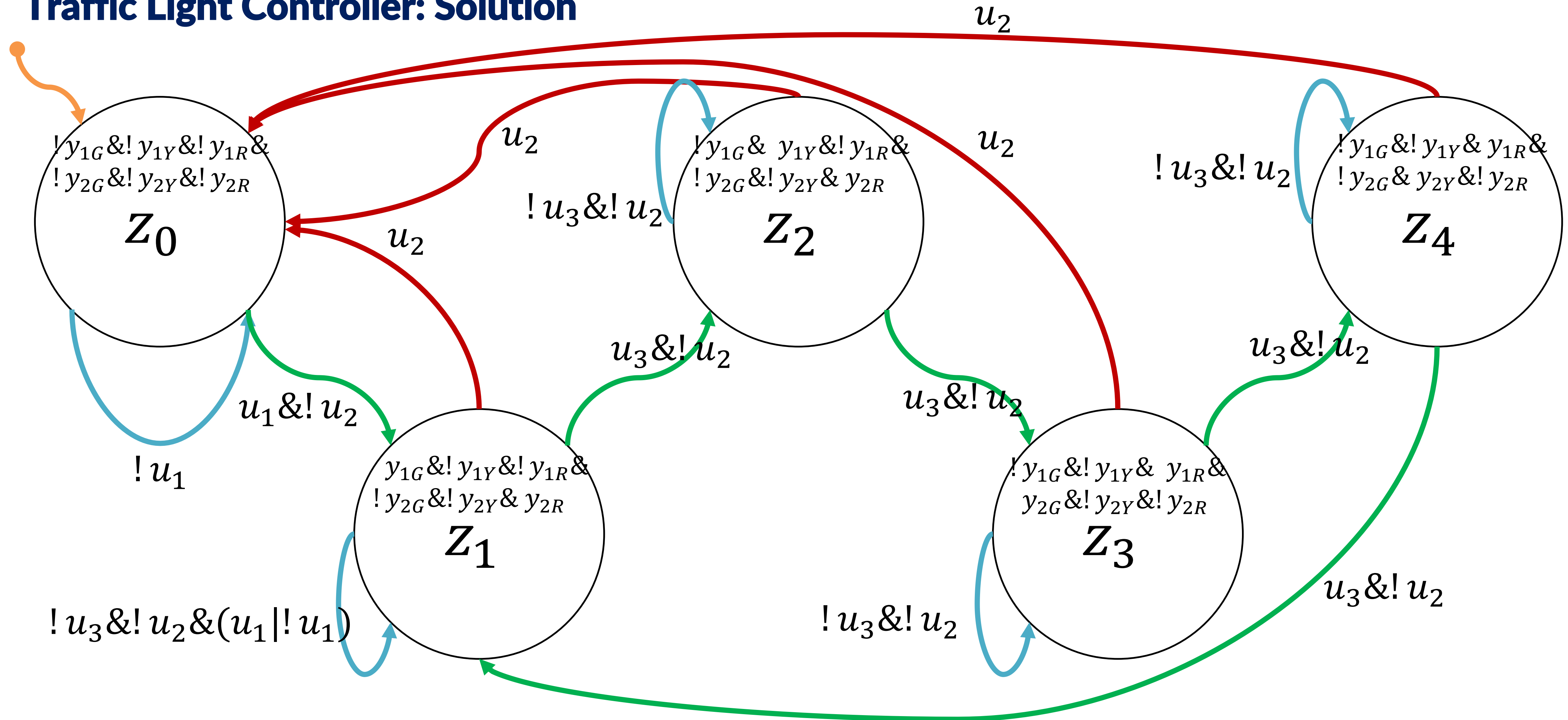
Traffic Light Controller: Solution



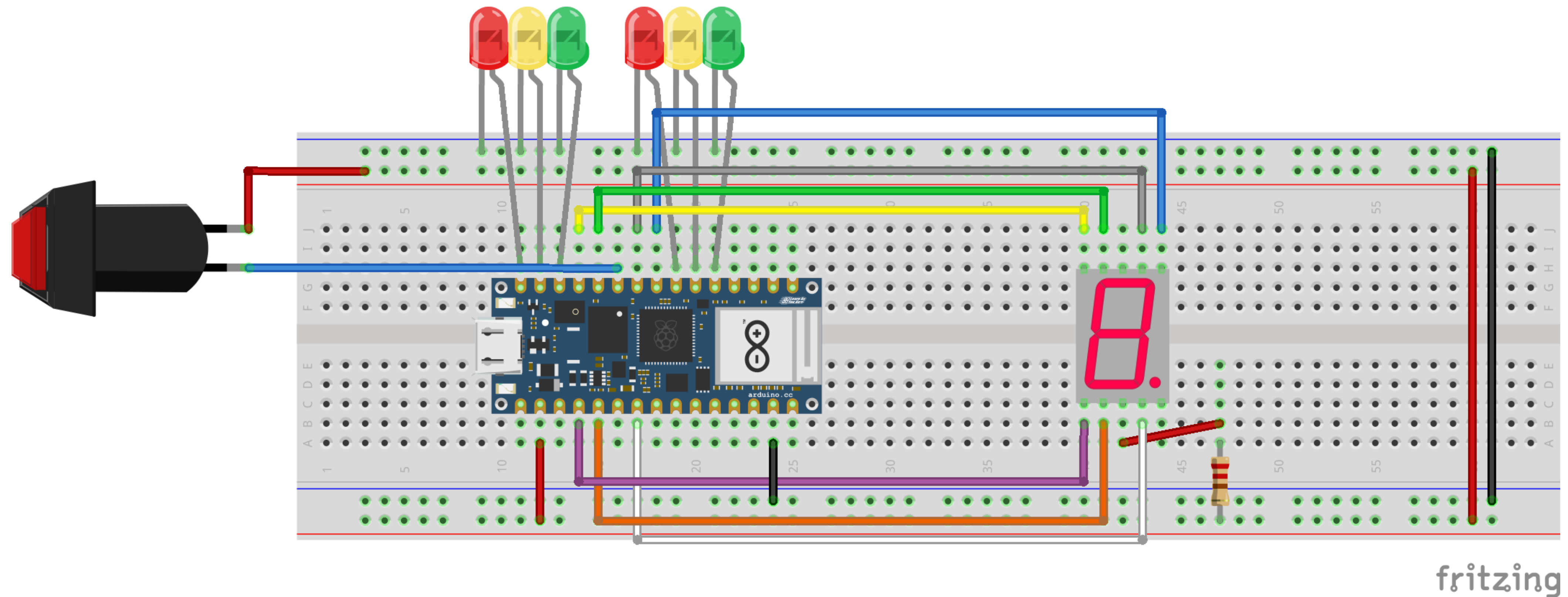
Traffic Light Controller: Solution



Traffic Light Controller: Solution



Traffic Light Controller: Wiring Diagram



Mealy Machine

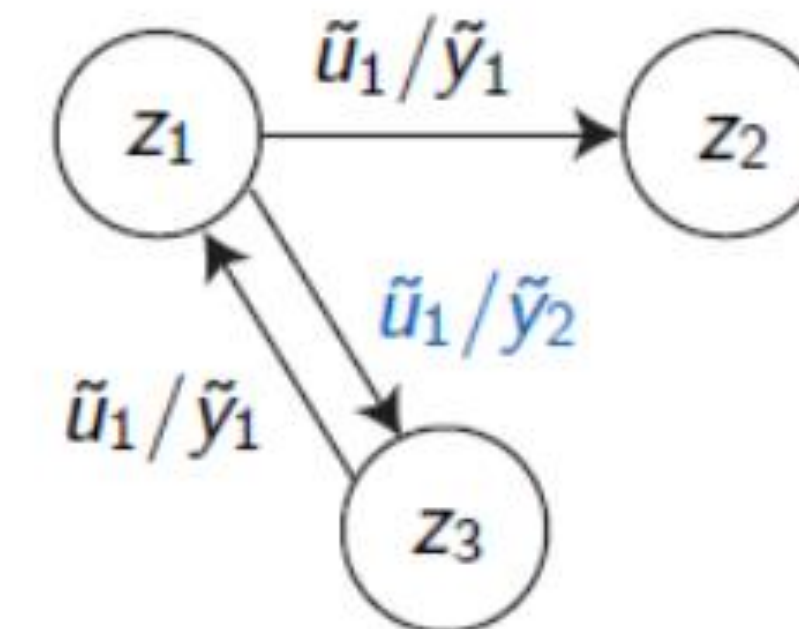
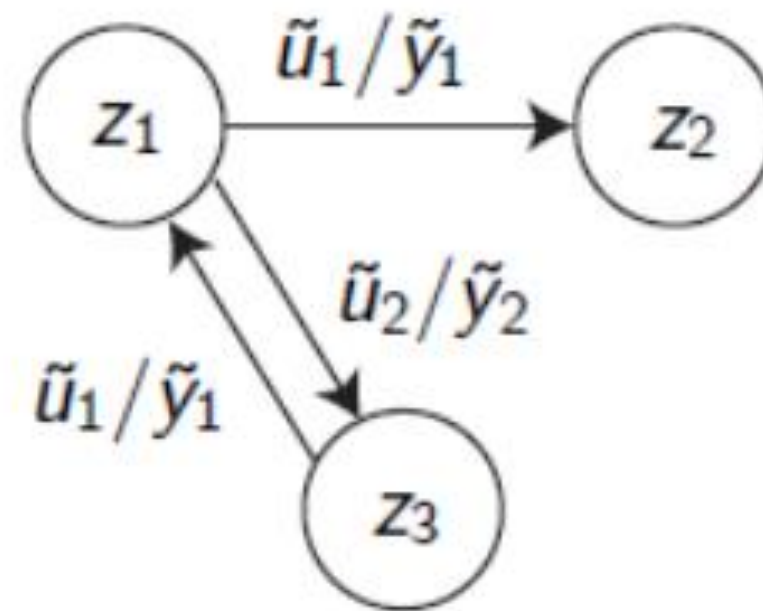
- Less number of states required
- Less hardware requirements
- Asynchronous output generation
- Harder to model DES

Moore Machine


- More number of states required
- More hardware requirements
- Synchronous output generation
- Easier to model DES

Limitations

- A Finite State Automata (FSA) is said to be **deterministic** if for each state there is at most one transition enabled by each input value (not one-to-many mapping), a state machine will be deterministic only if the guards leaving each state are non-overlapping.



- The drawback of the Finite State Automata (FSA) is that only one state is reached at a time as it is impossible for more than one condition to happen at the same time so it does not represent **concurrent systems**, as in the parking station if car entered and at the same time a car exits or traffic lights in cross roads depend on each other as they are not allowed to be read or green at the same time, so they have to be synchronized with each other.

For Further Inquiries, Please
 *send an email*

Catherine.elias@guc.edu.eg,
Catherine.elias@ieee.org

Thank you for your attention!

See you next time 😊