HLSM & RTL Design

(High-Level State Machine & Register Transfer Level)

Jia Chen
jiac@ucr.edu

HLSM — High-Level State Machine

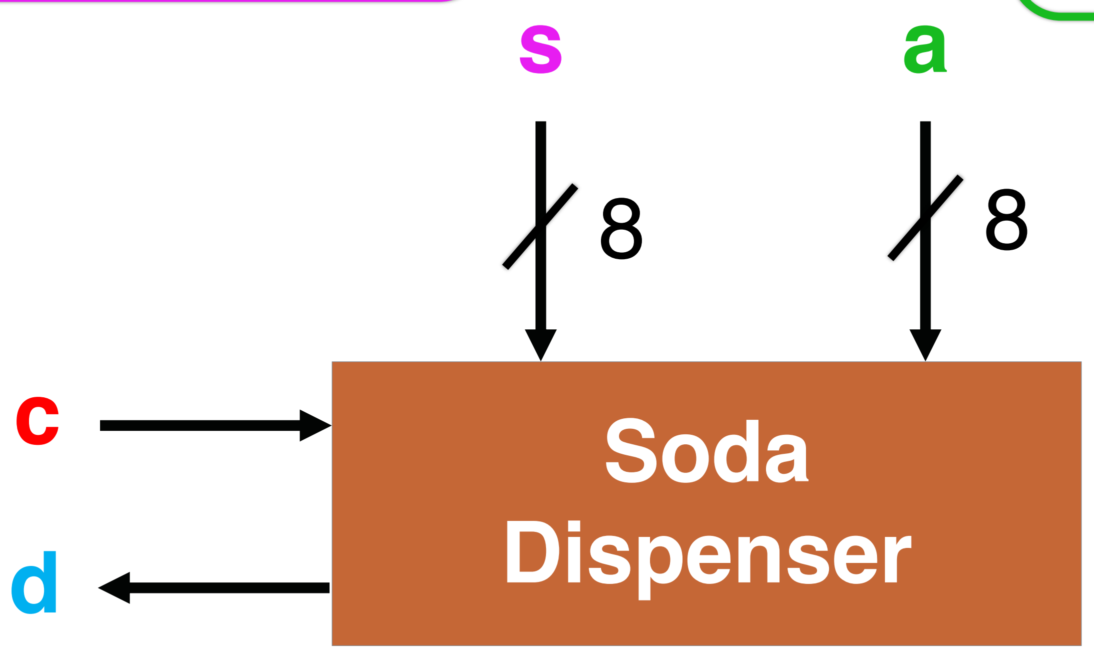
Soda dispenser

- Soda dispenser: dispenses a soda when enough money has been inserted

s: 8-bit input
cost of a soda

a: 8-bit input
value of the deposited coin

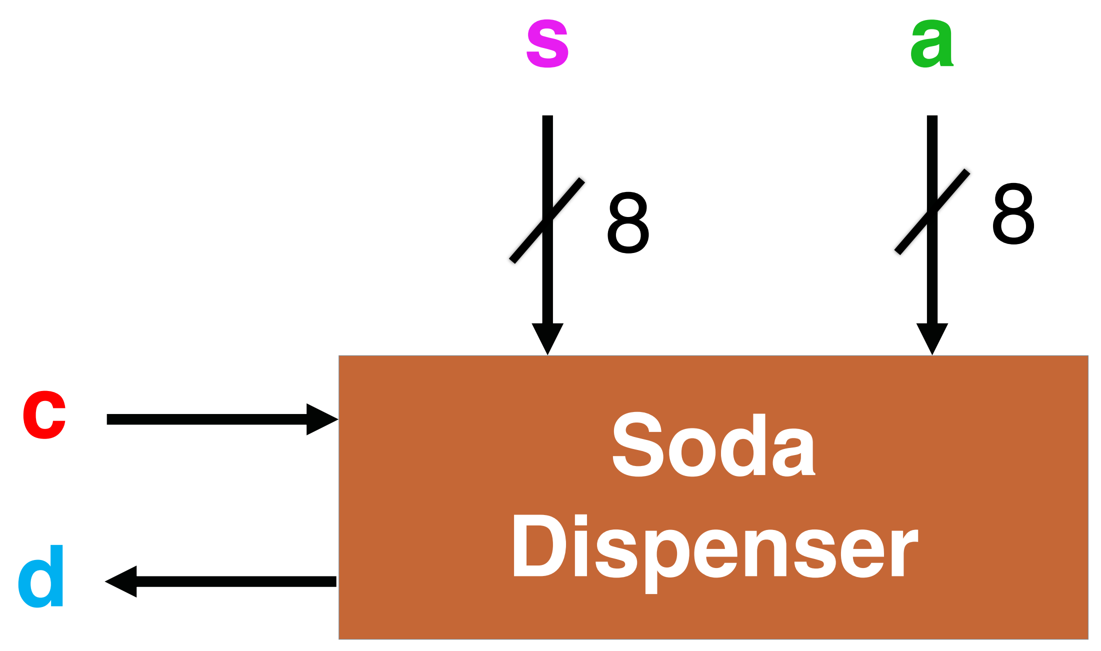
c: 1-bit input
c=1 coin is deposited
c=0 coin is not deposited



d: 1-bit output
d=1 machine dispenses a soda
d=0 machine does not dispenses a soda

High-Level State Machine

- An FSM has single-bit inputs and outputs
- Some behaviors may be too complex to describe by using classical FSMs
  - E.g., a sequential system has multibit data input and output, such as a system with 8-bit data input, e.g., **a** and **s** in the soda dispenser machine



- A ***high-level state machine*** **(HLSM)** is a form of extended FSM that supports multi-bit data input/output and data operations.

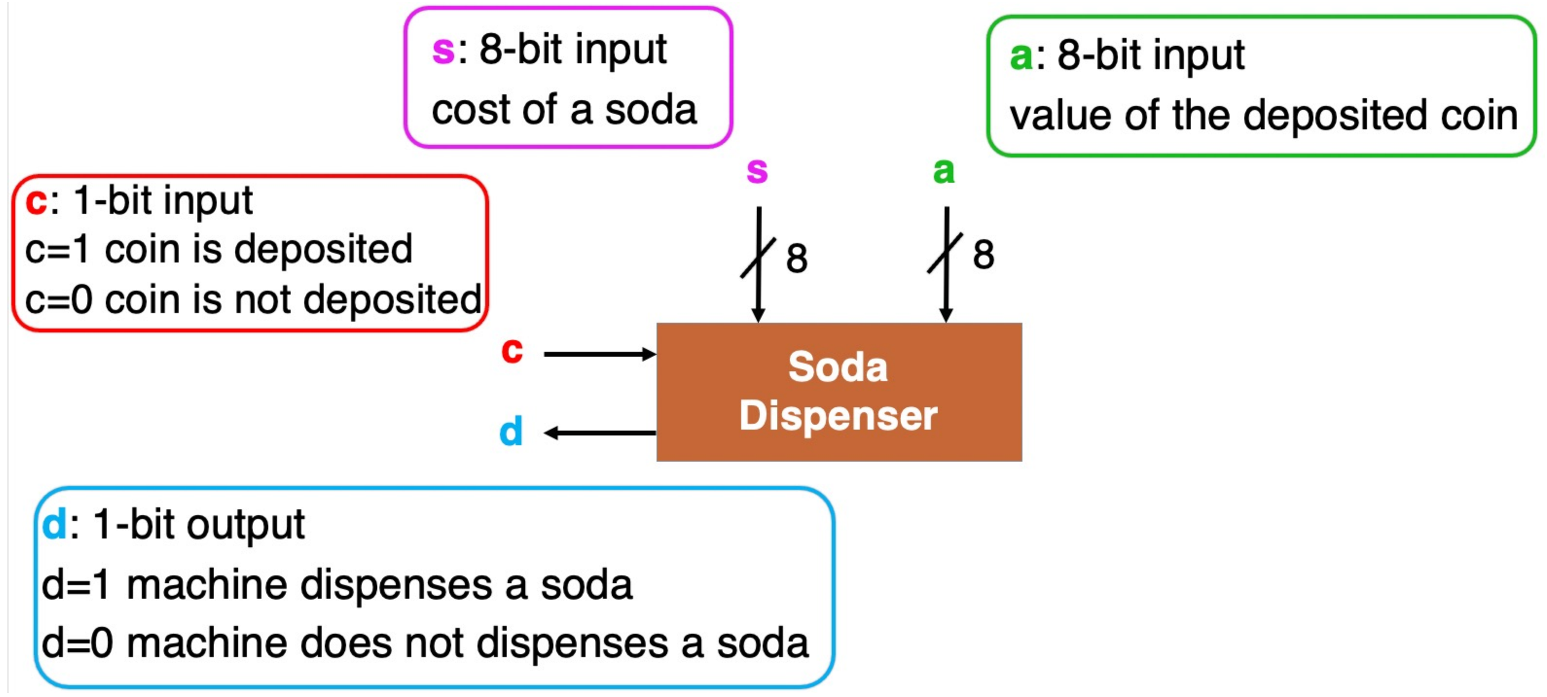
HLSMs v.s. FSMs

- Similarity
  - Sequential circuit
  - Transitions happen at the edge of a clock
  - Store states
  - Need registers

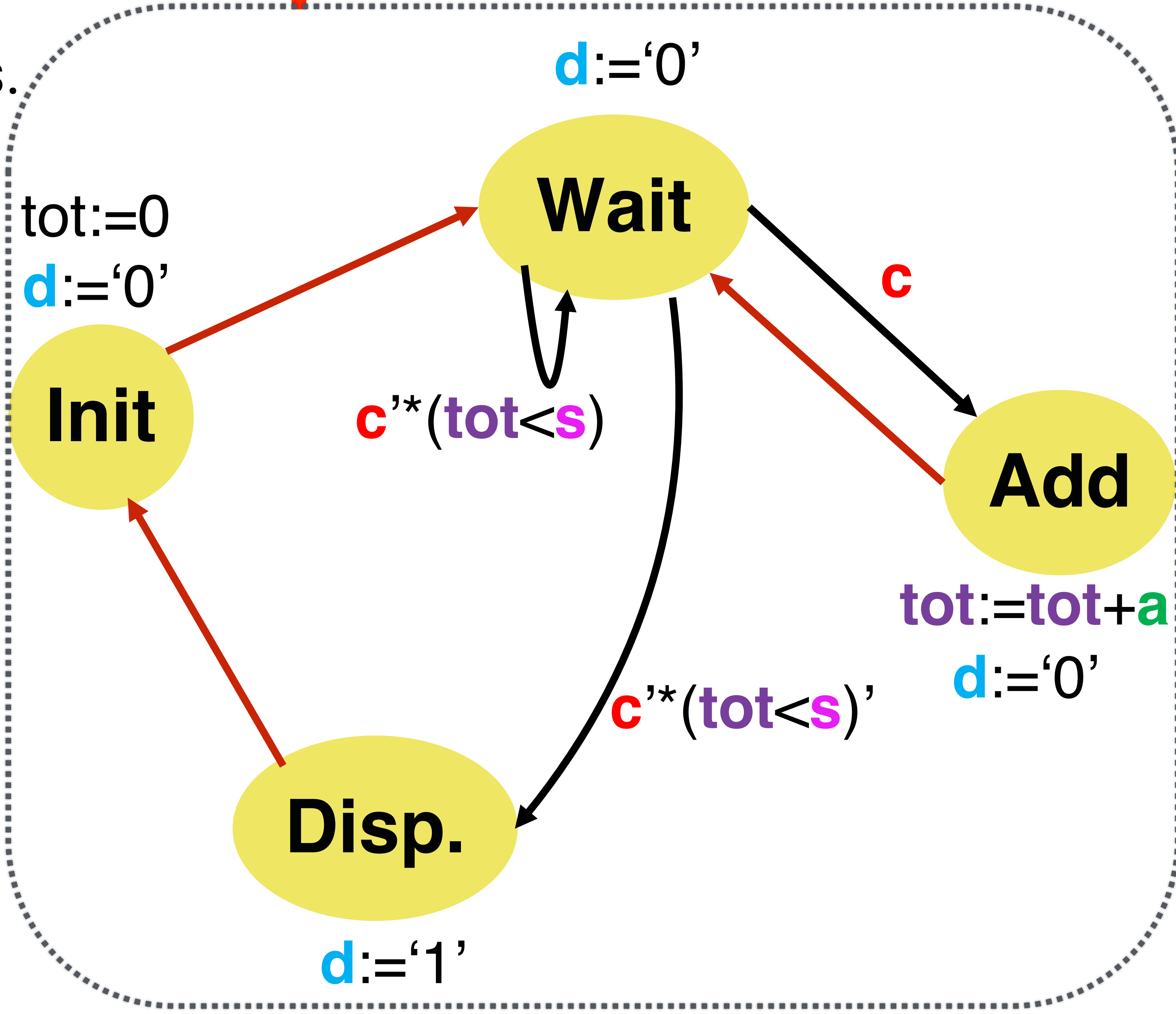
- Dissimilarity
  - HLSM stores multibit data, but FSM stores single bit data

HLSM for soda dispenser

HLSM supports: inputs, outputs, & variables.
A ***variable*** is a data item that maintains a value, and can be read or assigned.
E.g., "**tot**": the total money deposited



s: 8-bit input
cost of a soda
a: 8-bit input
value of the deposited coin
c: 1-bit input
c=1 coin is deposited
c=0 coin is not deposited
s
a
8
8
c
d
Soda Dispenser
d: 1-bit output
d=1 machine dispenses a soda
d=0 machine does not dispenses a soda

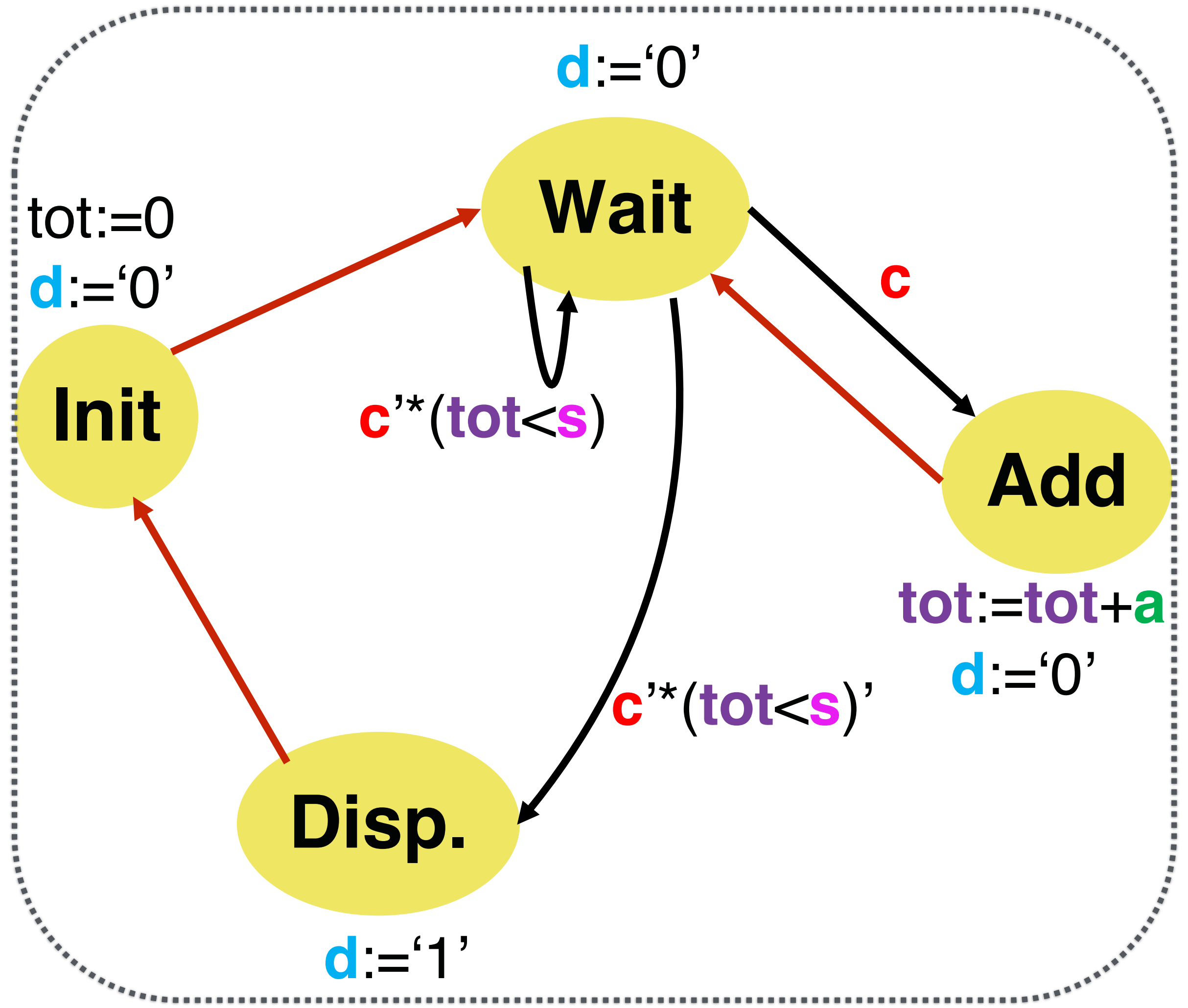


Move to next state regardless of inputs

Benefits of HLSMs

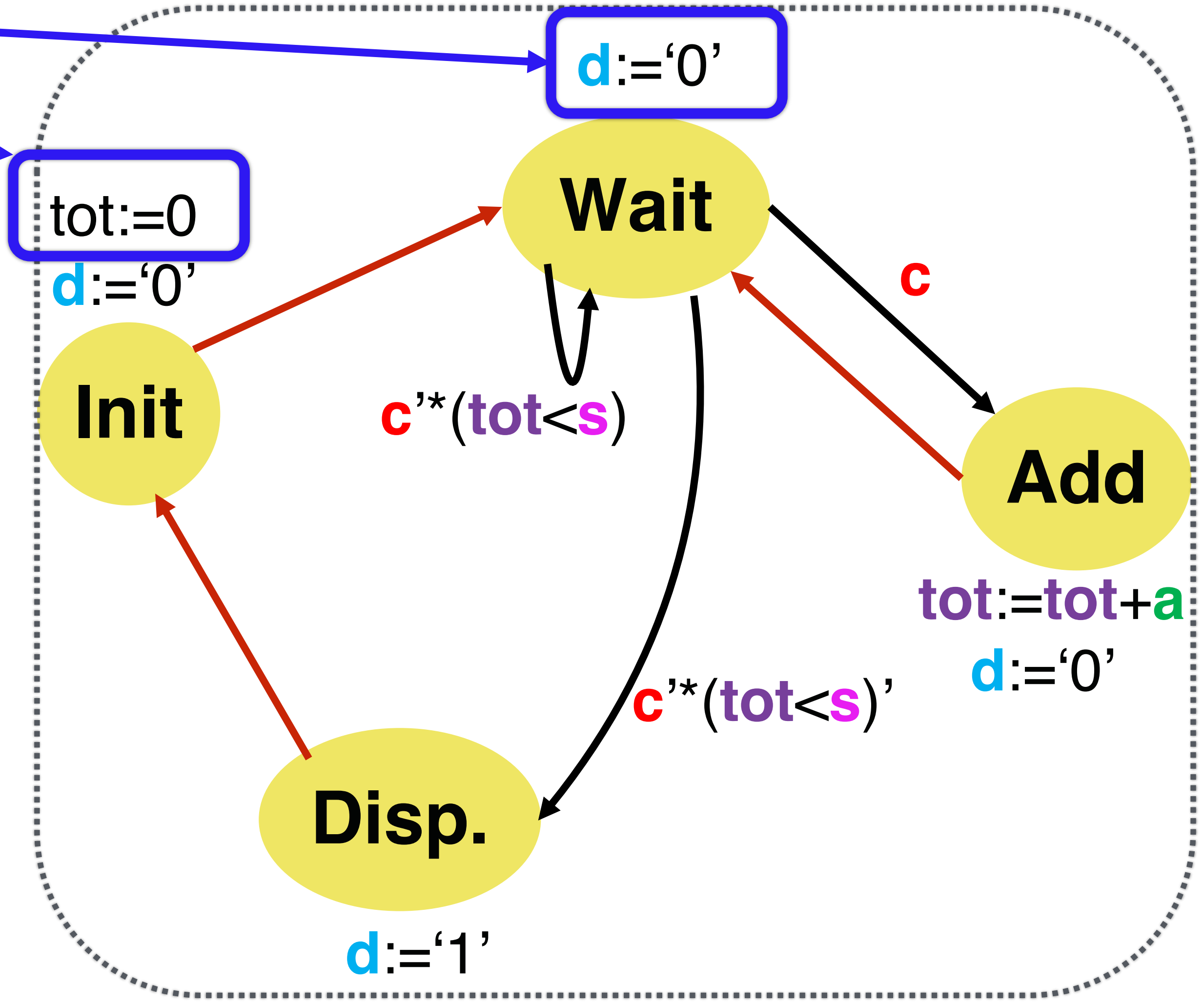
- High-level state machine (HLSM) extends FSM with:
  - Multi-bit input/output (e.g., a, s)
  - Local storage (e.g., store tot)
  - Arithmetic operations (e.g, tot < s and tot + a)
- Conventions
  - Each transition is implicitly ANDed with a rising edge of the clock
  - Any bit output not explicitly assigned a value in a state is implicitly assigned to 0. This convention does not apply for multibit outputs

    (e.g., d:='0' can be omitted in "Wait" and "Add")
  - Every HLSM multibit output is registered (stored)



Conventions

- Numbers:
  - Single-bit: '0' (single quotes)
  - Integer: 0 (no quotes)
  - Multi-bit: “0000” (double quotes)



- == for comparison equal
- // precedes a comment

Challenges in High-Level State Machines

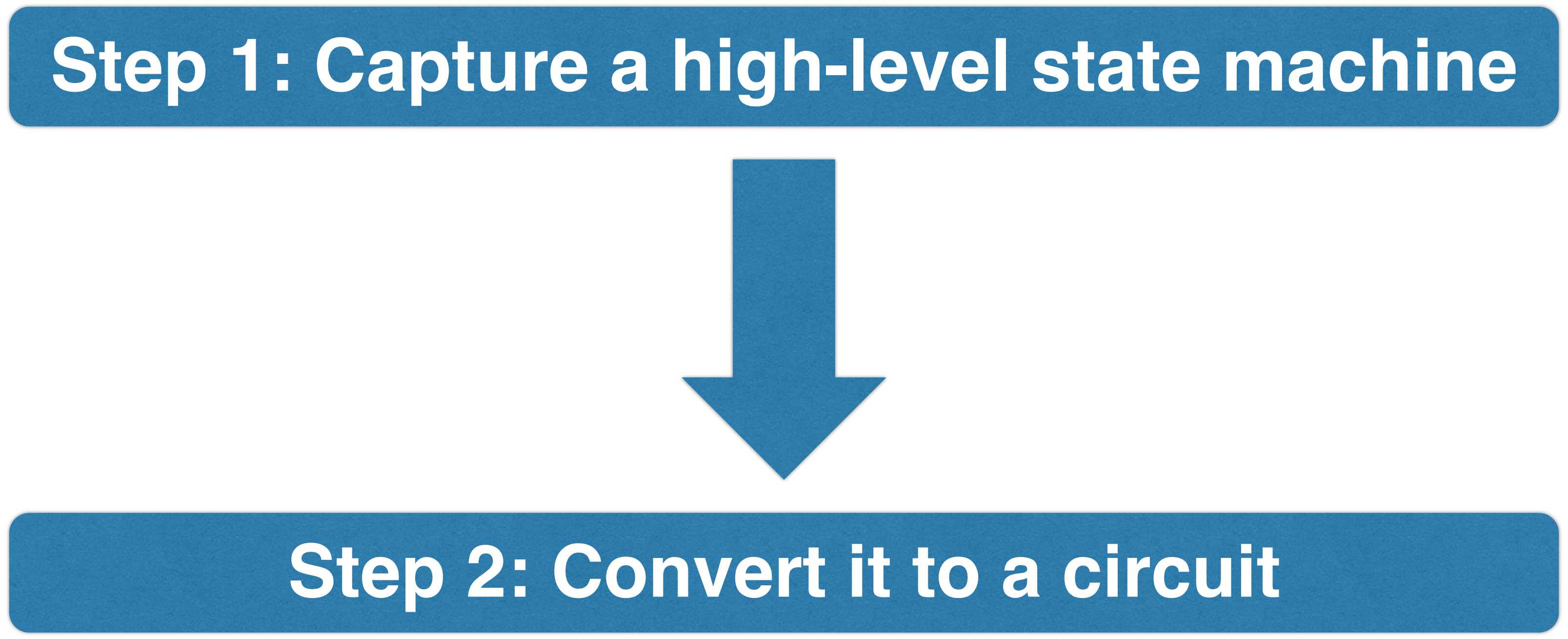
- ❑ Factors that make the FSM design of this problem difficult
  - ➢ 8-bit input/output
  - ➢ Tracking the current total
  - ➢ Multibit comparison

RTL
(Register Transfer Level) Design

Slides adopted from Hung-Wei Tseng

RTL Design Process

- ❑ The process of capturing behavior as an HLSM and converting to a circuit is known as ***register-transfer-level* (RTL) *design***


Step 1: Capture a high-level state machine
Step 2: Convert it to a circuit

RTL Design

- ❑ Controllers
  - Control input/output: single bit (or just a few) representing event or state
  - FSM describes behavior; implementation

- ❑ Datapath components
  - Data input/output: Multiple bits collectively representing single entity
  - Datapath components included registers, adders, comparators, multipliers, etc.

- ❑ This Lecture: custom **processors**
  - Processor: Controller and datapath components working together to implement an algorithm

RTL Design Process

- Step 1: Capture a high-level state machine
  - Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is high level because the transition conditions and the state actions are more than just Boolean operations on single-bit input and outputs
  - Recommendations:
    - ✓ Always list all inputs, outputs and local registers on top of your HLSM diagram
    - ✓ Clearly specify the size in bits of each of them
    - ✓ On states: update the value of registers, update of outputs
    - ✓ On transitions: express conditions in terms of the HLSM inputs or state of the internal values and arithmetic operations between them.

RTL Design Process

- Step 2: Convert it to a circuit
  - Create a datapath
    - Create a datapath to carry out the data operations of the high level state machine
    - Elements of your datapaths can be registers, adders, comparators, multipliers, dividers, etc.
  - Connect the datapath to a controller
    - Connect the datapath to a controller block.
    - Connect the external control inputs and outputs to the controller block.
    - Clearly label all control signals that are exchanged between the datapath and the controller
  - Derive the controller’s FSM
    - Convert the high-level state machine to a finite state machine (FSM) for the controller, by replacing data operations with setting and reading of control signals to and from the datapath
  - Implement the controller

RTL Design Summary

- Recall
  - ✓ Combinational Logic Design
    - First step: Capture behavior (using equation or truth table)
    - Remaining steps: Convert to circuit
  - ✓ Sequential Logic Design
    - First step: Capture behavior (using FSM)
    - Remaining steps: Convert to circuit
- RTL Design - the method for creating custom processors
  - ➢ First step: Capture behavior (using high-level state machine)
  - ➢ Remaining steps: Convert to circuit
    - High-level architecture (datapath and control path)
    - Datapath capable of HLSM's data operations
    - Design controller to control the datapath

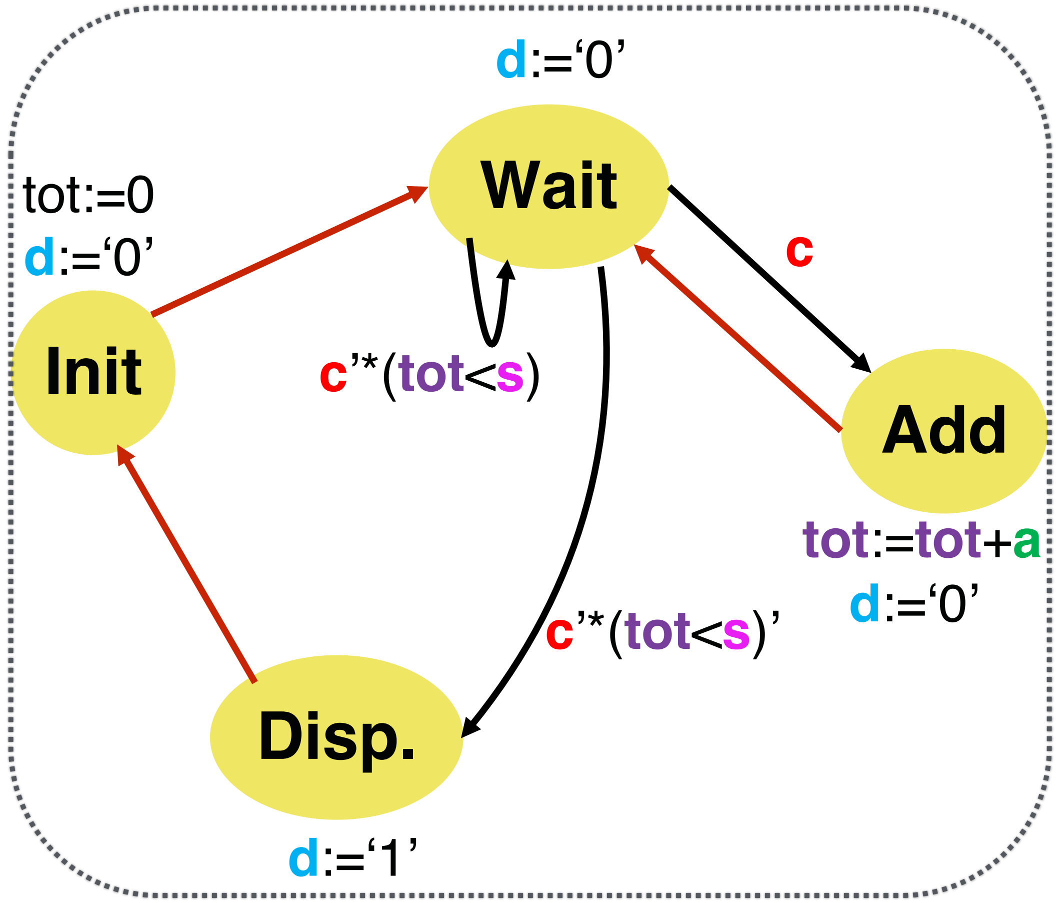
Create Datapath for Soda Dispenser

s: 8-bit input cost of a soda;

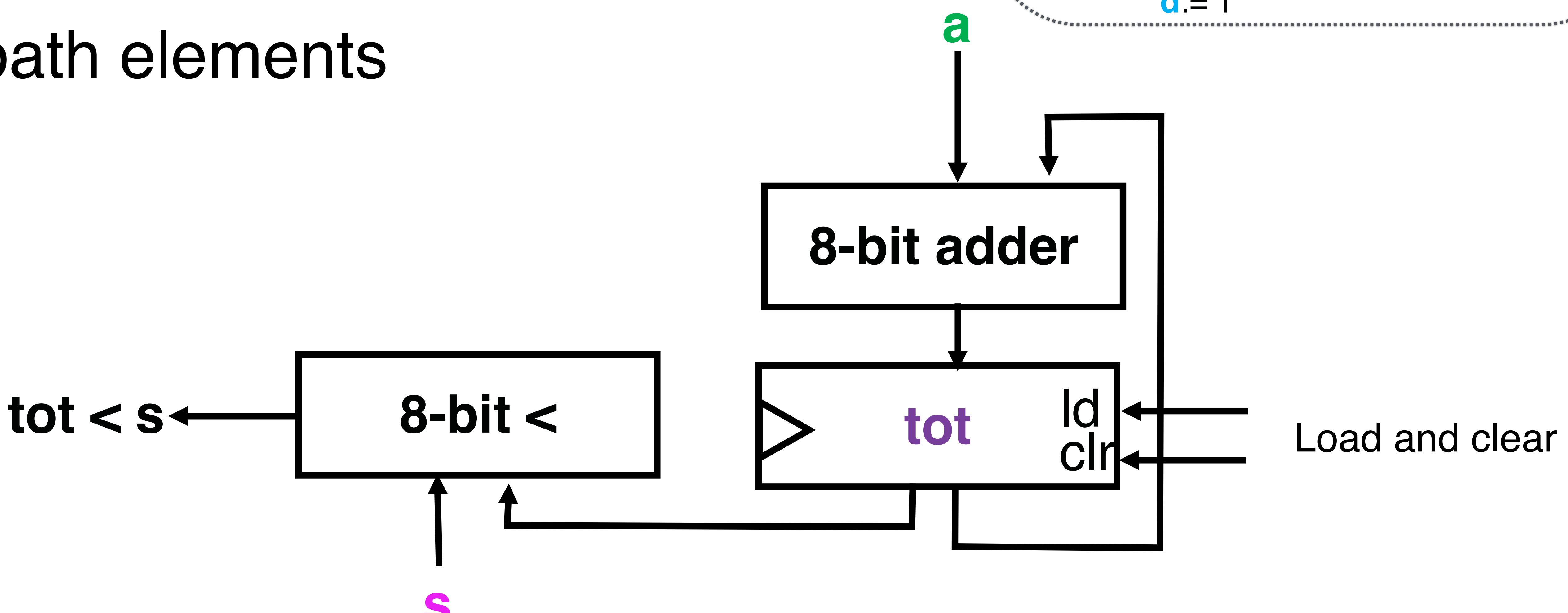
a: 8-bit input value of the deposited coin;

c: 1-bit input;

d: 1-bit output

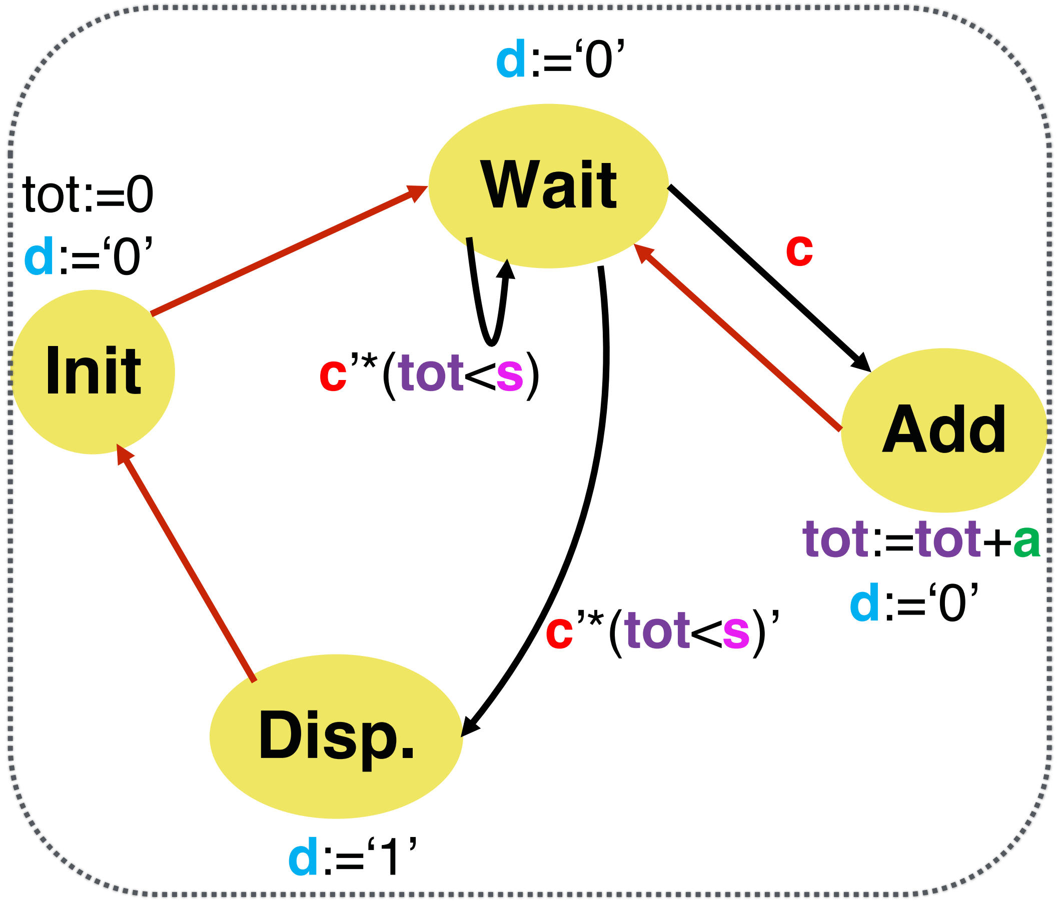


- Register: tot
- Comparator: to compare tot and s
- Adder: to update tot = tot + a
- Connect datapath elements
- I/O interface

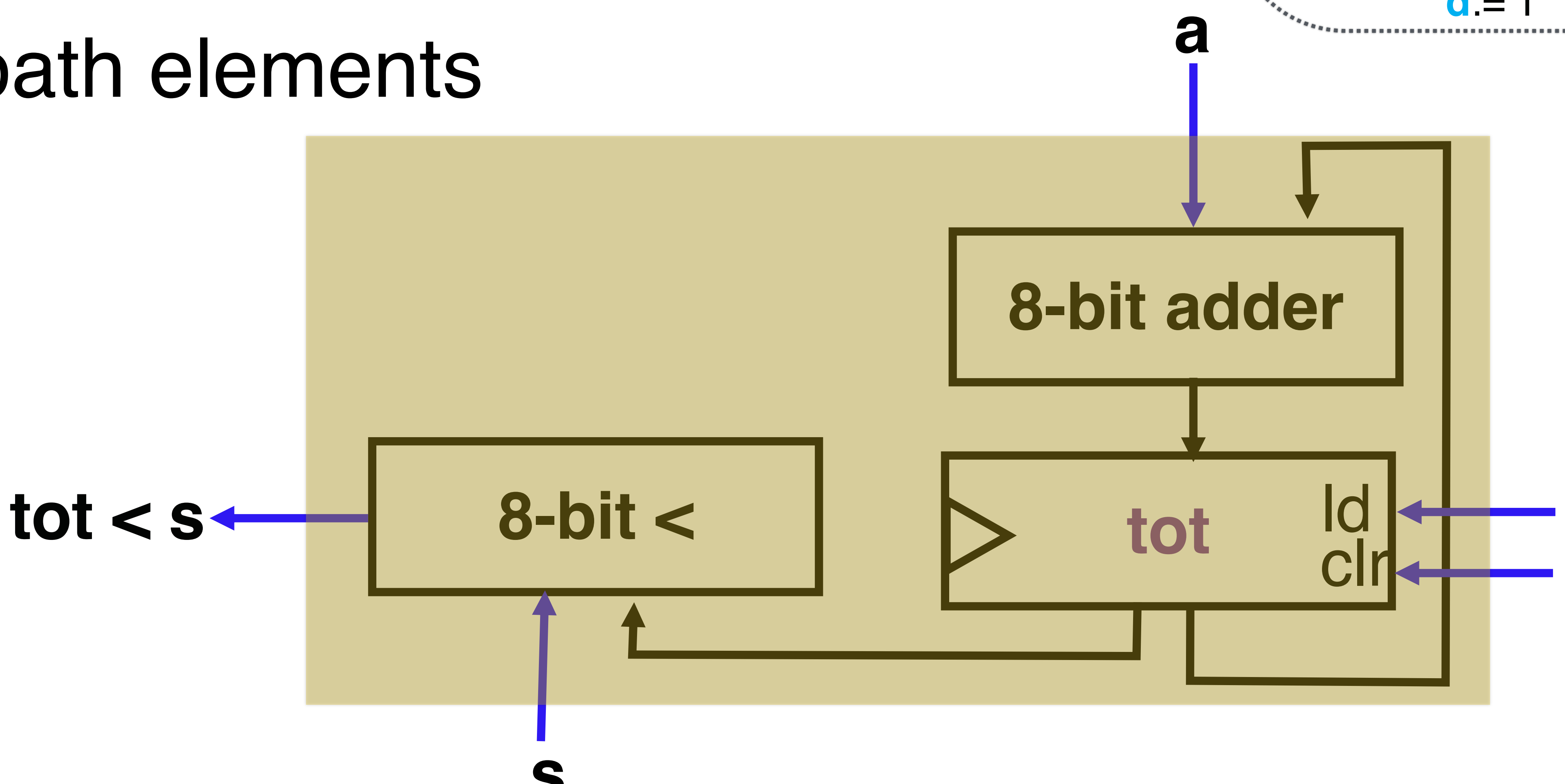


Create Datapath for Soda Dispenser

s: 8-bit input cost of a soda;
a: 8-bit input value of the deposited coin;
c: 1-bit input;
d: 1-bit output

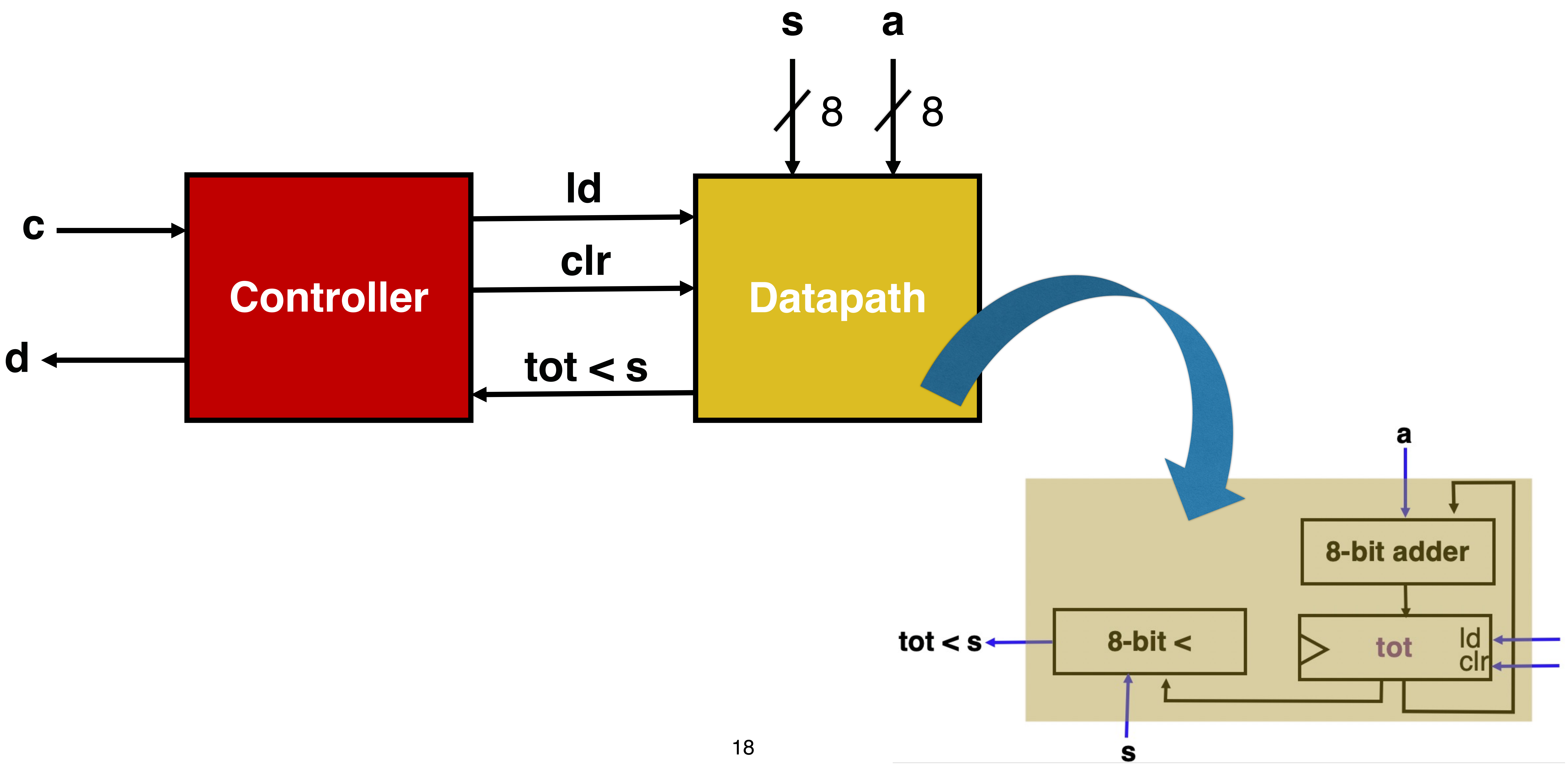


- Register: tot
- Comparator: to compare tot and s
- Adder: to update tot = tot + a
- Connect datapath elements
- I/O interface



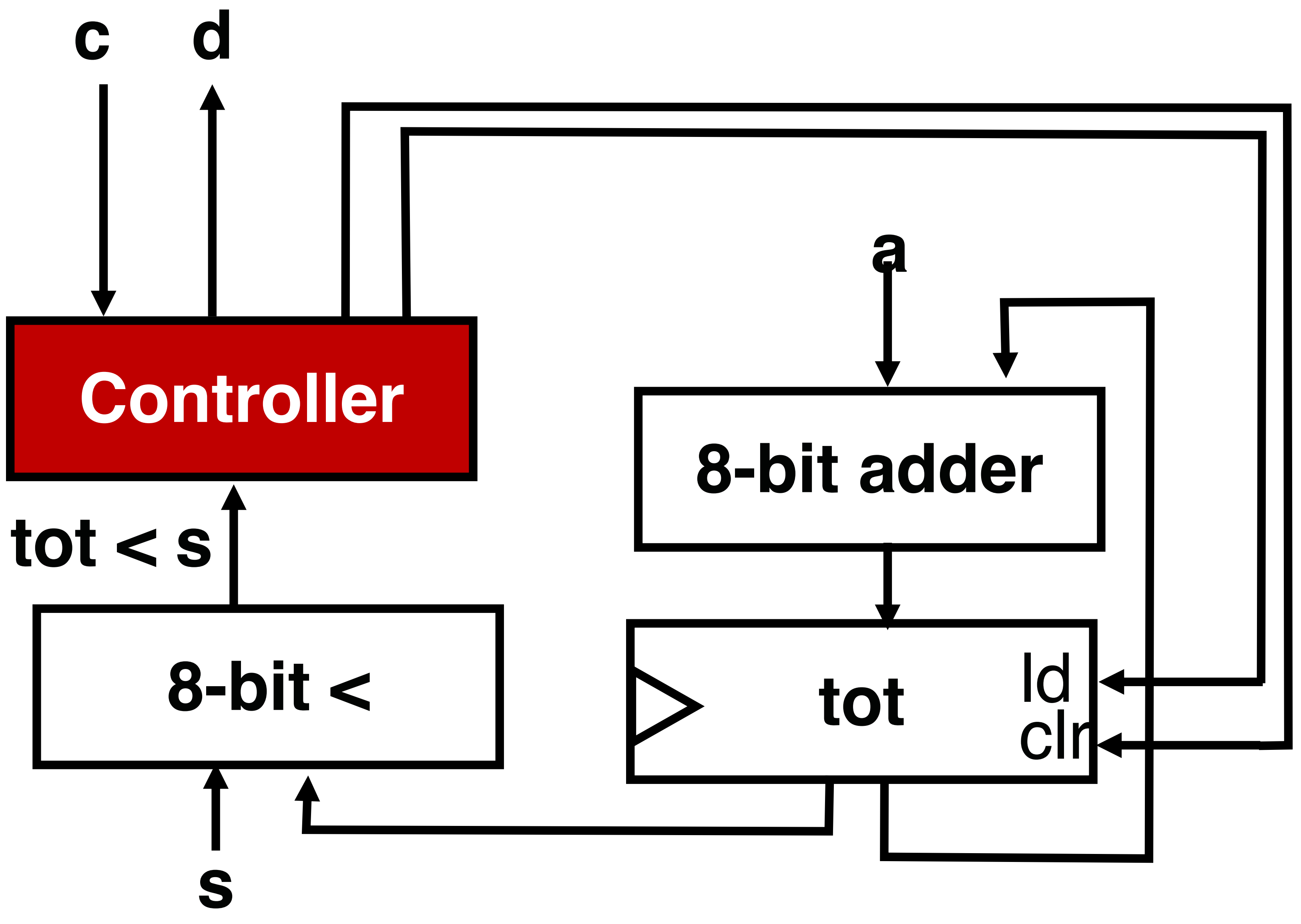
Datapath (4 inputs and 1 output)

Connect Datapath to a Controller



Connect Datapath to a Controller

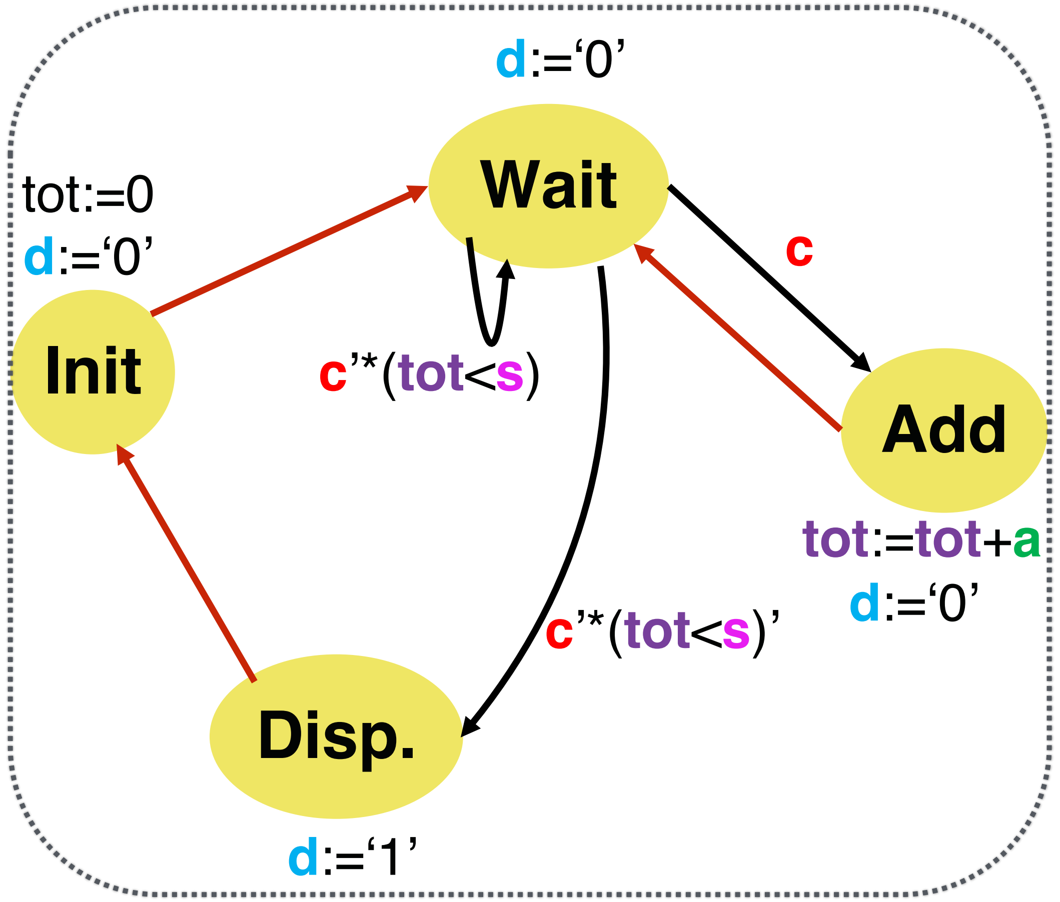
- Controller's inputs
  - External input c (coin detected)
  - Input from datapath comparator's output, which we named tot<s
- Controller's outputs
  - External output d (dispense soda)
  - Outputs to datapath to load and clear the tot register

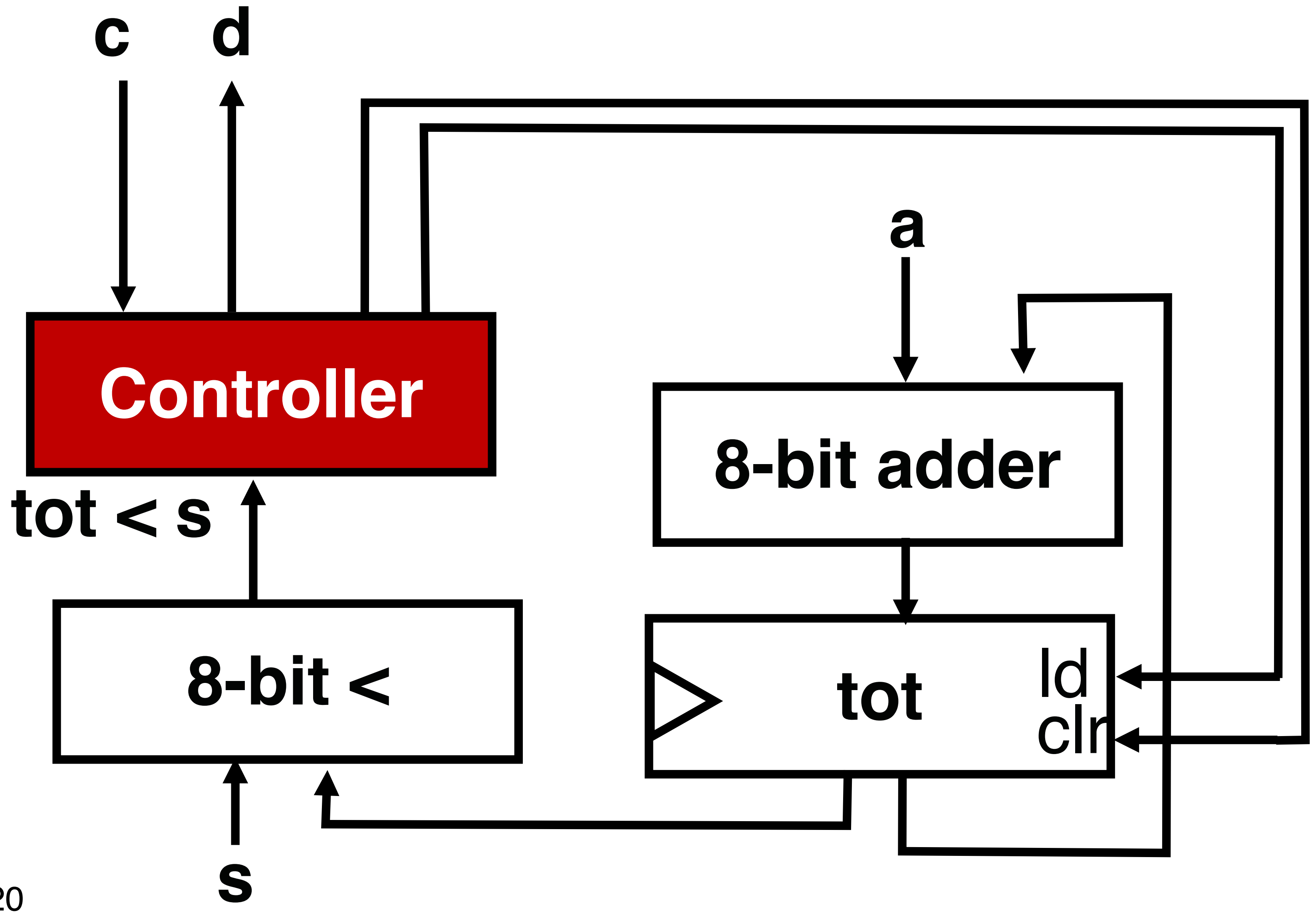


So far, only the design of the Controller is missing!

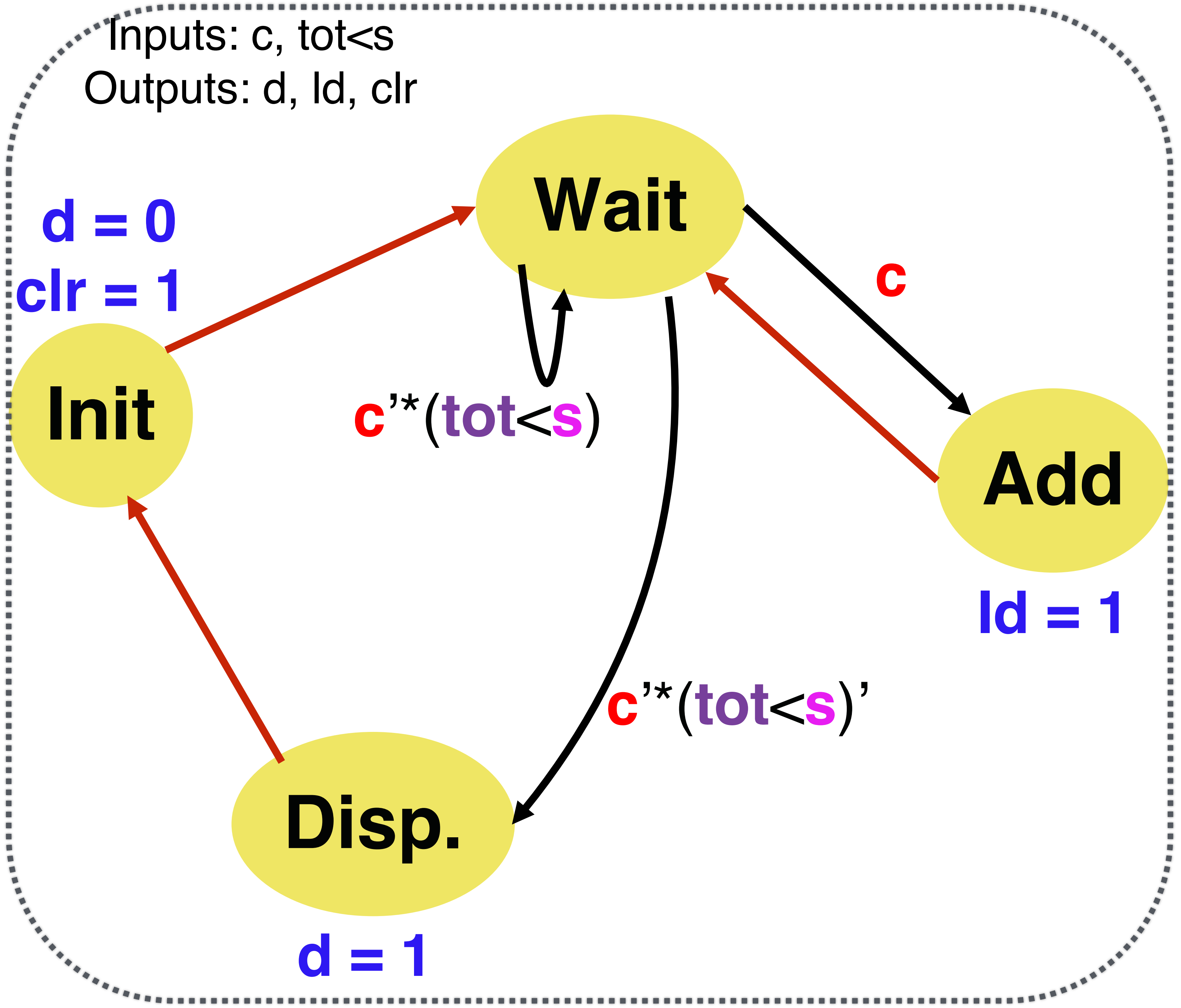
Derive the Controller’s FSM

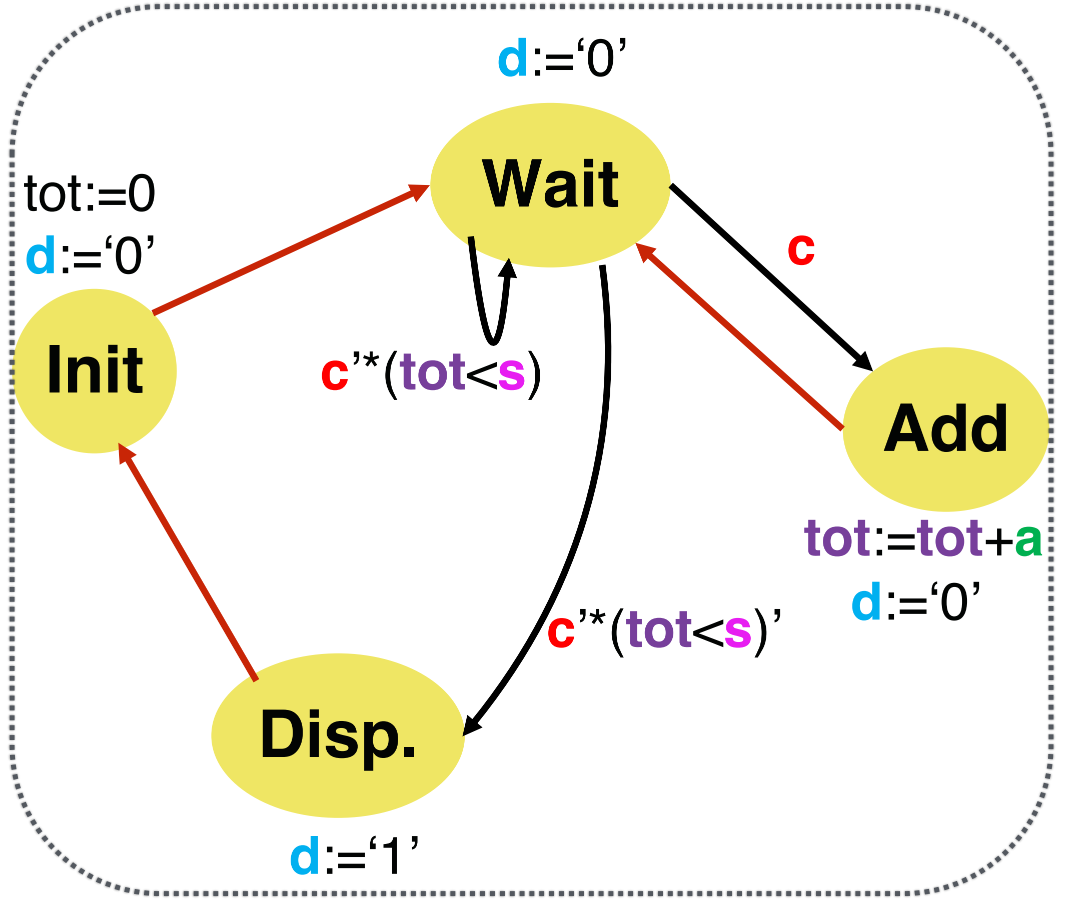
- FSM has the same states and arcs as HLSM
- But set/read datapath control signals (signals connected to the controller) for all datapath operations and conditions

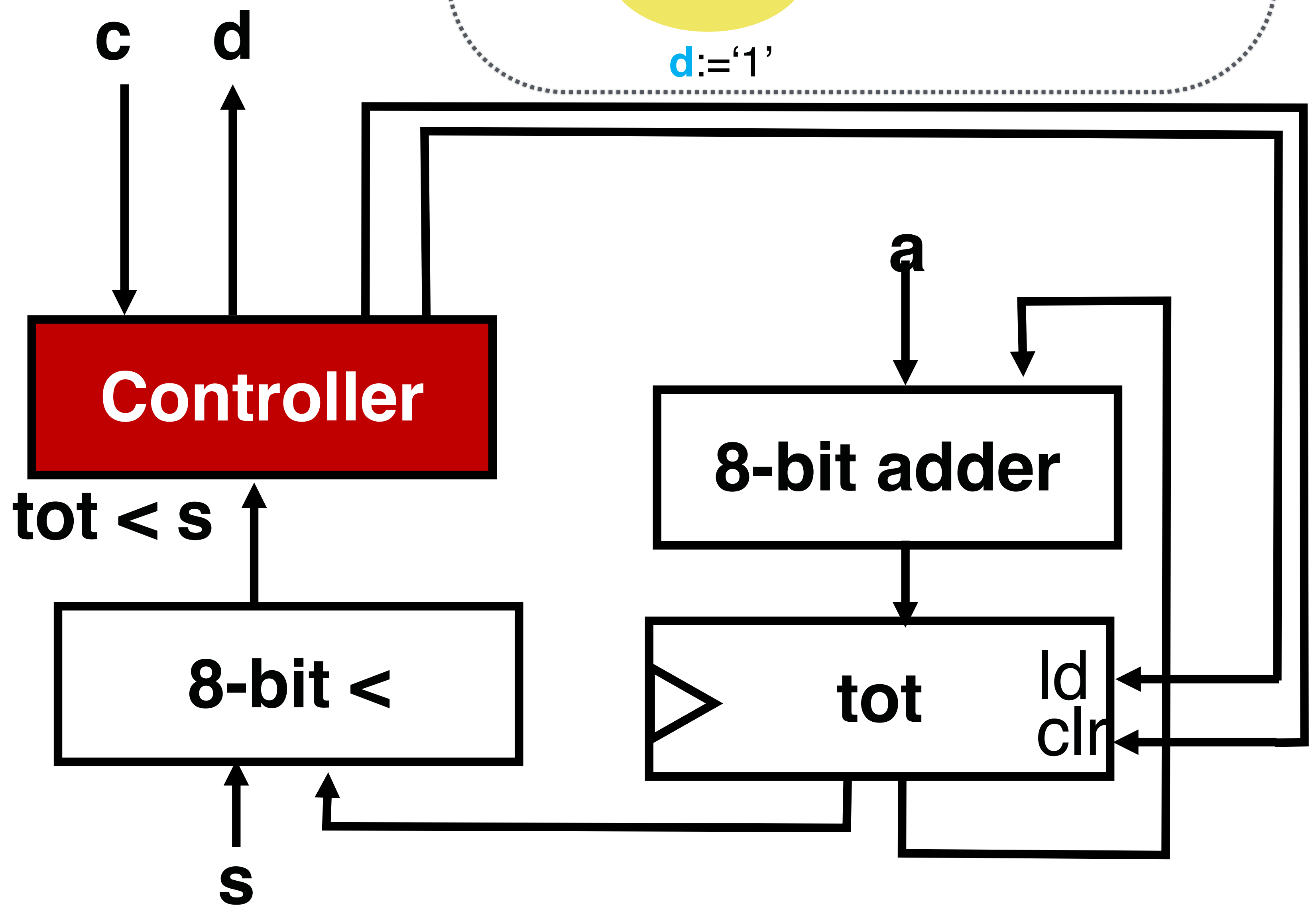




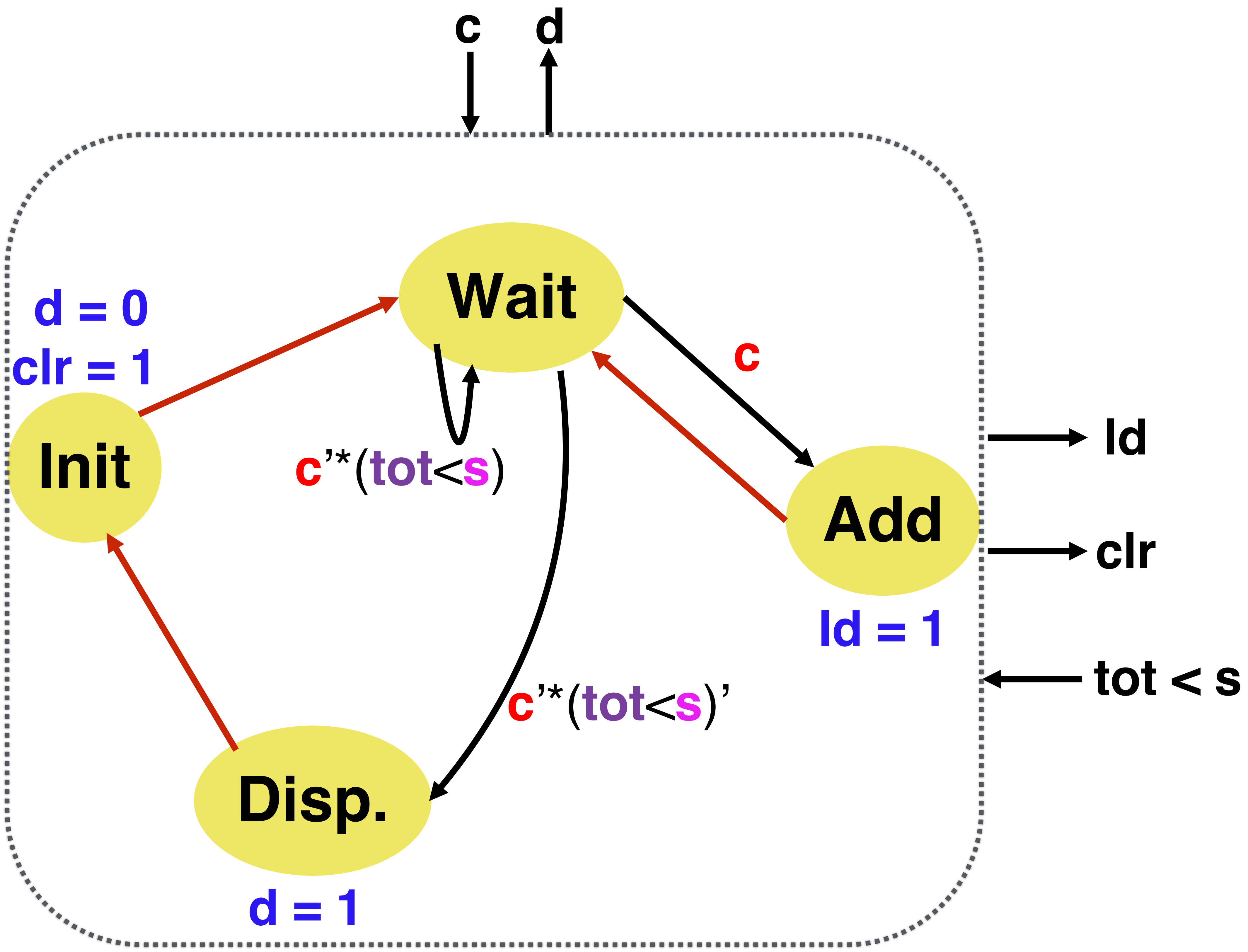
Derive the Controller’s FSM



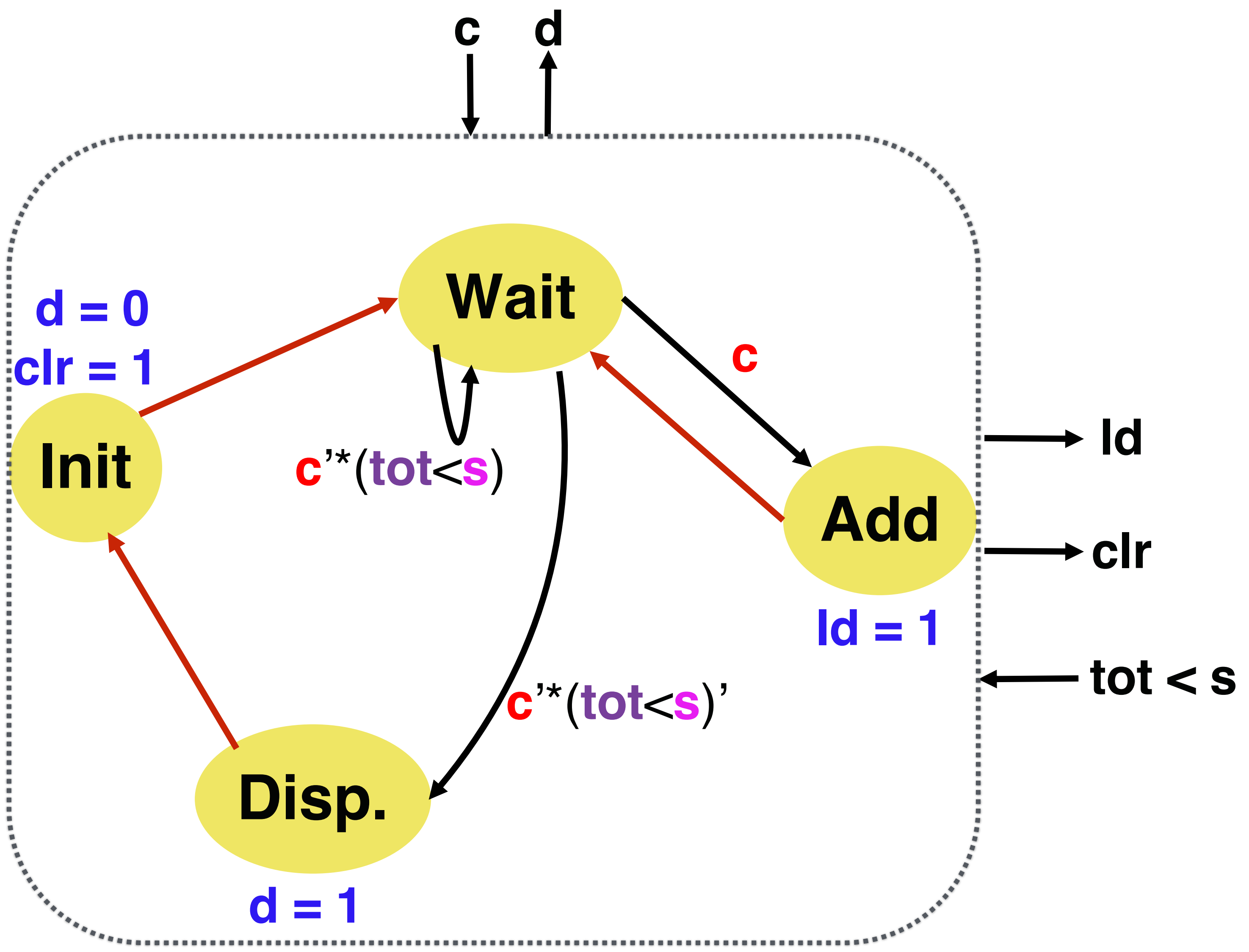




Final Step: Implement the controller FSM



Final Step: Implement the controller FSM



Excitation table

	Current state		Inputs		Next state		Outputs		
	Q1	Q0	c	tot<s	D1	D0	d	ld	clr
Init	0	0	0	0	0	1	0	0	1
	0	0	0	1	0	1	0	0	1
	0	0	1	0	0	1	0	0	1
	0	0	1	1	0	1	0	0	1
Wait	0	1	0	0	1	1	0	0	0
	0	1	0	1	0	1	0	0	0
	0	1	1	0	1	0	0	0	0
	0	1	1	1	1	0	0	0	0
Add	1	0	0	0	0	1	0	1	0
		• • •					• • •		
Disp	1	1	0	0	0	0	1	0	0
		• • •					• • •		

… means the other 3 rows have the same outputs

Soda dispenser example

- ❑ Not an FSM because:
  - ✓ Multi-bit (data) inputs *a* and *s*
  - ✓ Local register *tot*
  - ✓ Data operations *tot=0, tot<s, tot=tot+a*.
- ❑ Useful high-level state machine:
  - ✓ Data types beyond just bits
  - ✓ Local registers
  - ✓ Arithmetic equations/expressions