ECE 2300 Digital Logic and Computer Organization

Topic 7: Finite State Machines

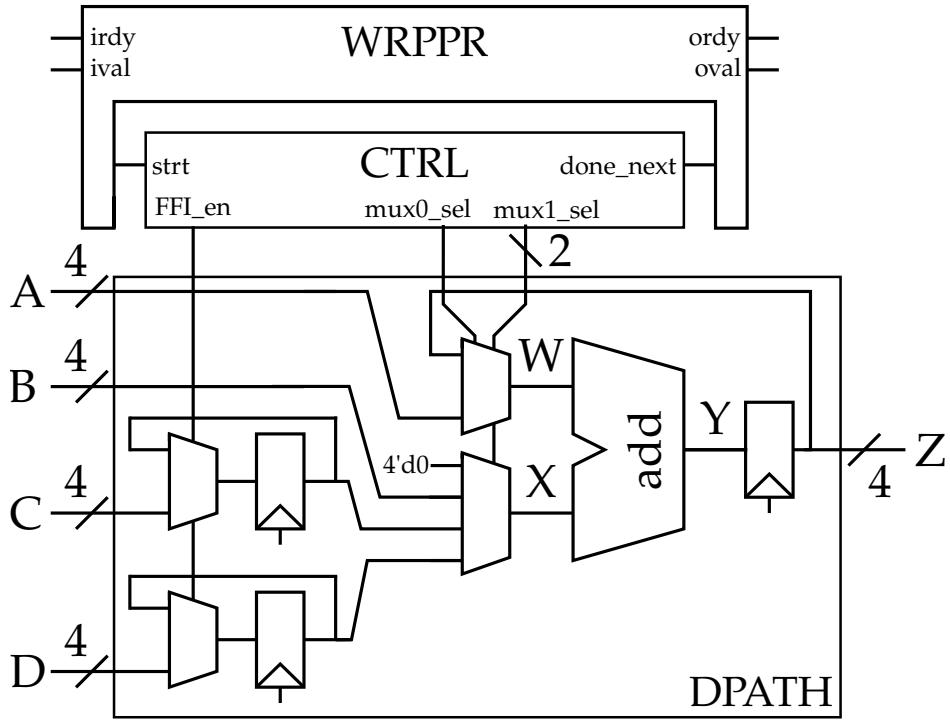
http://www.csl.cornell.edu/courses/ece2300
School of Electrical and Computer Engineering
Cornell University

revision: 2025-12-09-12-37

List of Problems

Problem 1. Control System for Multi-Cycle Quad Adder

In lecture (topic 8), we will discuss a multi-cycle quad adder (module dpath in figure below), which sums up four four-bit values in multiple cycles utilizing a single adder module. In this assignment, we will design a control system (module ctrl) for such a multi-cycle quad adder. Furthermore, we will design a wrapper FSM (module wrppr), which connects this multi-cycle quad adder via the *latency insensitive valid/ready interface* to a producer and consumer.



The data path of the multi-cycle quad adder has four four-bit input ports (**A**, **B**, **C**, **D**) and one four-bit output port (**Z**). Furthermore, the data path can be controlled by a control module via three input ports. **FFI_en** enables the registers of C and D. **mux0_sel** and **mux1_sel** control the two multiplexors in front of the adder.

Note: This multi-cycle quad adder is modified from lecture: For instance, input A and B are not stored in registers, the registers for C and D have additional enable logic, and the mux, which outputs X, has an additional input hardcoded to 0.

The control module itself has an input port **strt** to start the addition process and an output **done_next**, which indicates that in the next cycle the computation will be complete. The wrapper module connects to the control module via its strt and done_next ports. Furthermore, the wrapper module has has an irdy output and ival input for its input val/rdy interface, and an ordy input and oval output for its output val/rdy interface.

First, let us analyze the data-path module. **Use the following table to list every constraint which must be satisfied to ensure correct operation of the data-path. Start by labeling the two possible constraints at the top of the table.** Each path should be specified with just the start and end points of the path (feel free to name FFs yourself). Assume paths that start at an input port or end at an output port are unconstrained (for the moment, only cosider paths starting and ending at FFs). Each constraint should be expressed as an inequality. If the constraint is satisfied use $\geq$ or $\leq$. If the constraint is not satisified use $\ngeq$ or $\nleq$. **You must show each delay component in the inequality along with the the final sum. Circle any constraints which are not statisified and would result in a timing violation.**

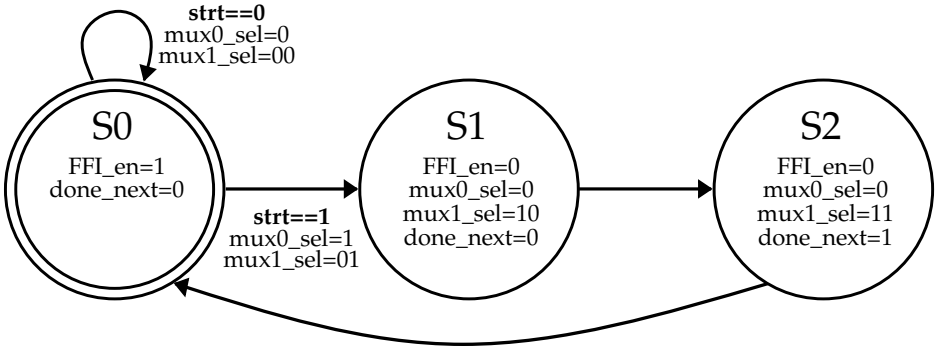
	t_{pd}	t_{cd}
Mux2	8τ	2τ
Mux4	15τ	3τ
4-Bit Adder	40τ	2τ
FF (t_{cq})	9τ	2τ

FF (t_{setup})	10τ
FF (t_{hold})	1τ
T_C	75τ

Path Start Point	**Path End Point**	_____ **Constraint**	_____ **Constraint**

Are there any unsatisfied constraints (i.e., timing violations) within the data-path? What is the minimum clock period (T_C) that would still ensure correct operation of the data-path?

We designed the following FSM for the ctrl unit.



Is this a Moore or Mealy FSM? Explain why.

Design the next state logic of the FSM with the following kmaps; write the resulting boolean expression under each kmap:

next_s[1]

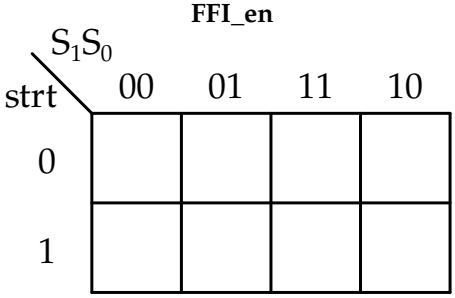
strt \ S_1S_0	00	01	11	10
0				
1				

next_s[0]

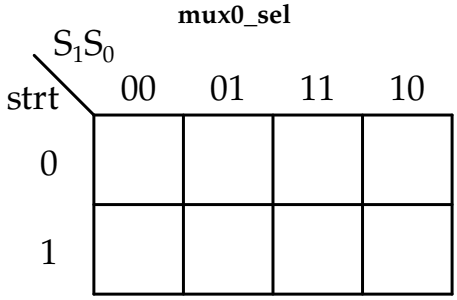
strt \ S_1S_0	00	01	11	10
0				
1				

Design the output logic of the FSM with the following kmaps; write the resulting boolean expression under each kmap:

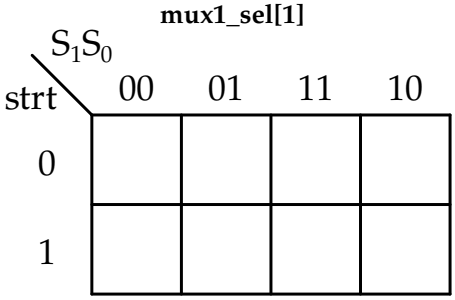
FFI_en



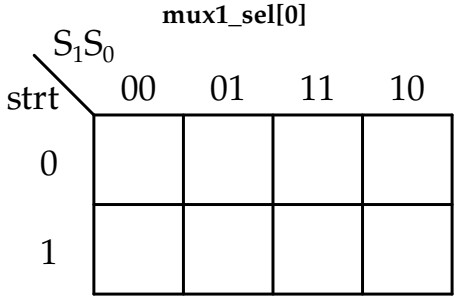
mux0_sel



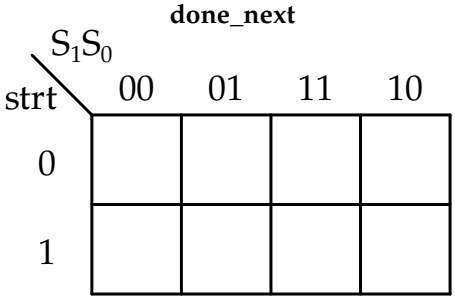
mux1_sel[1]



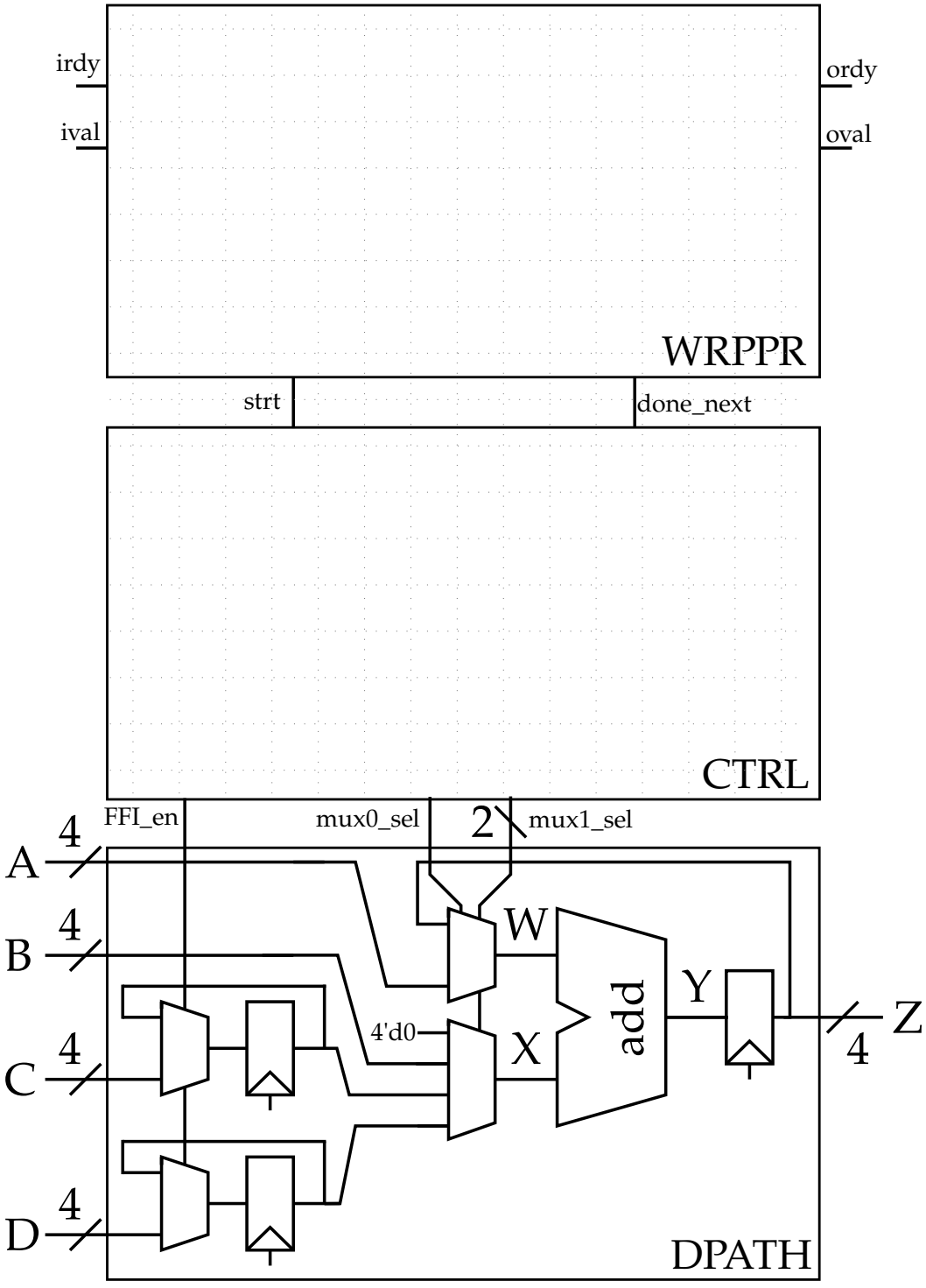
mux1_sel[0]



done_next



Draw the gate network and registers of the ctrl module in the its box below.



Next, let us analyze the ctrl module. **Use the following table to list every constraint which must be satisfied to ensure correct operation of the control unit. Start by labeling the two possible constraints at the top of the table.** Each path should be specified with just the start and end points of the path (feel free to name FFs yourself). Assume paths that start at an input port or end at an output port are unconstrained (for the moment, only cosider paths starting and ending at FFs within the control unit). Each constraint should be expressed as an inequality. If the constraint is satisfied use $\geq$ or $\leq$. If the constraint is not satisified use $\ngeq$ or $\nleq$. **You must show each delay component in the inequality along with the the final sum. Circle any constraints which are not statisified and would result in a timing violation.**

	t_{pd}	t_{cd}
NOT	1τ	1τ
AND2	3τ	1τ
OR2	4τ	1τ
FF (t_{cq})	9τ	2τ

FF (t_{setup})	10τ
FF (t_{hold})	1τ
T_C	75τ

Path Start Point	**Path End Point**	_____ **Constraint**	_____ **Constraint**

Are there any unsatisfied constraints (i.e., timing violations) within the control unit? What is the minimum clock period (T_C) that would still ensure correct operation of the control unit?

Next, we will design the wrapper FSM of the wrppr unit. It lets the multi-cycle quad adder accept a set of four four-bit inputs (A, B, C, D of the data-path) with its latency insensitive valid/ready input interface. After the computation is complete, the module transfers the sum via its output valid/ready interface to a subsequent consumer. The wrapper module communicates with the control unit of the data-path via the strt and done_next port.

The latency insensitive valid/ready interface works like this: The producer sets its valid signal when its data is ready for the consumer. The consumer sets its ready signal, when it is ready to accept incoming data. When at the end of a clock cycle both ready and valid signals are set, producer and consumer know that the data was transferred.

Draw the FSM diagram according to the following features:

1. The FSM should have the following three states: WS0, WS1, WS2.
2. It should reset to state WS0.
3. In state WS0, output irdy needs to be set and oval not set.
4. The FSM should move from state WS0 to WS1 when ival is set. In this case, the FSM should set strt. Otherwise the FSM would remain in WS0 and strt would be unset.
5. In state WS1, all its outputs (strt, irdy, oval) should not be set.
6. The system should move from WS1 to WS2 when done_next is set. Otherwise it should remain in WS1.
7. In state WS2, the FSM needs to set oval, while keeping strt and irdy unset.
8. The FSM needs to move from WS2 to WS0 when ordy is set. Otherwise it should remain in WS2.

Next, we will design the next state logic for the WRPPR unit. **Is this a moore or mealy FSM? Why can we not minimize it with kmaps?**

Instead, write the next state logic without major minimization in sum of products form from the FSM diagram.

However, we will minimize the output logic with kmaps. We won't consider done_next and ordy for minimization, since they don't have any impact on the outputs of the wrppr FSM. **Complete the kmaps for strt, irdy, and oval.**

strt

ival \ S_1S_0	00	01	11	10
0				
1				

irdy

ival \ S_1S_0	00	01	11	10
0				
1				

oval

ival \ S_1S_0	00	01	11	10
0				
1				

Draw the gate network and registers of the wrppr module in its box of the previous figure (the same figure, in which you drew the gate network of the control unit).

Let us analyze the timing constraints of the wrppr module. **Use the following table to list every constraint which must be satisfied to ensure correct operation of the control unit. Start by labeling the two possible constraints at the top of the table.** Each path should be specified with just the start and end points of the path (feel free to name FFs yourself). Assume paths that start at an input port or end at an output port are unconstrained (for the moment, only cosider paths starting and ending at FFs within the control unit). Each constraint should be expressed as an inequality. If the constraint is satisfied use $\geq$ or $\leq$. If the constraint is not satisified use $\ngeq$ or $\nleq$. **You must show each delay component in the inequality along with the the final sum. Circle any constraints which are not statisified and would result in a timing violation.** *Note: Depending on your implementation, you might not need all rows of the table.*

	t_{pd}	t_{cd}
NOT	1τ	1τ
AND2	3τ	1τ
OR2	4τ	1τ
FF (t_{cq})	9τ	2τ

FF (t_{setup})	10τ
FF (t_{hold})	1τ
T_C	75τ

Path Start Point	**Path End Point**	_____ **Constraint**	_____ **Constraint**

Are there any unsatisfied constraints (i.e., timing violations) within the control unit? What is the minimum clock period (T_C) that would still ensure correct operation of the control unit?

Until now, we only considered critical paths within each module starting at a FF and ending at a FF. For this part of the assignment, we will now consider paths starting at the input ports. **Can you find a path starting at the input port of the WRPPR module, crossing the ctrl unit and ending at the output register of the data-path? Mark this critical path in your gate diagram.**

Assume the input port has an input delay of 20τ. **Compute the critical path delay crossing all three modules. Would this critical path lead to timing violations?**

Explain the reason for this long critical path. Why does it go through all three modules?

How could this be prevented?

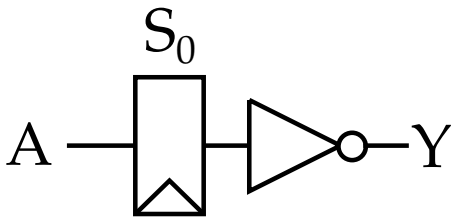
Anthony's Solution

`https://vod.video.cornell.edu/id/1_8oa6z69v`

Problem 2. Modulo Counters

Part 2.A A Simple FSM

Consider the following sequential gate-level network. **Fill in the the truth table (i.e. the next state and output table)**. Note that this *is not a simulation table.*



S	A	Y
0	0	
0	1	
1	0	
1	1	

What type of FSM is this? _____.

Draw the FSM diagram for this simple FSM.

Fill out the following simulation table for this FSM.

cycle	0	1	2	3	4	5	6	7	8	9	10
A	0	0	1	1	0	1	0	0	1	1	1
S	0										
S_{next}	0										
Y	1										

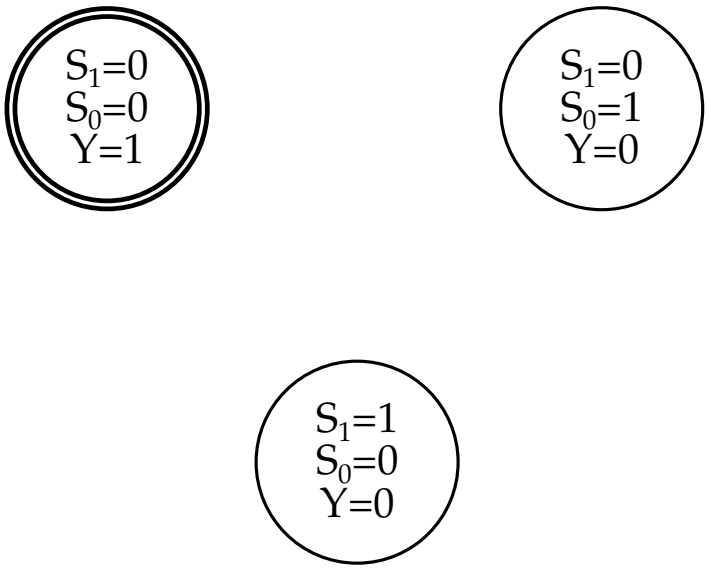
What does this FSM do?

Part 2.B A Generalization

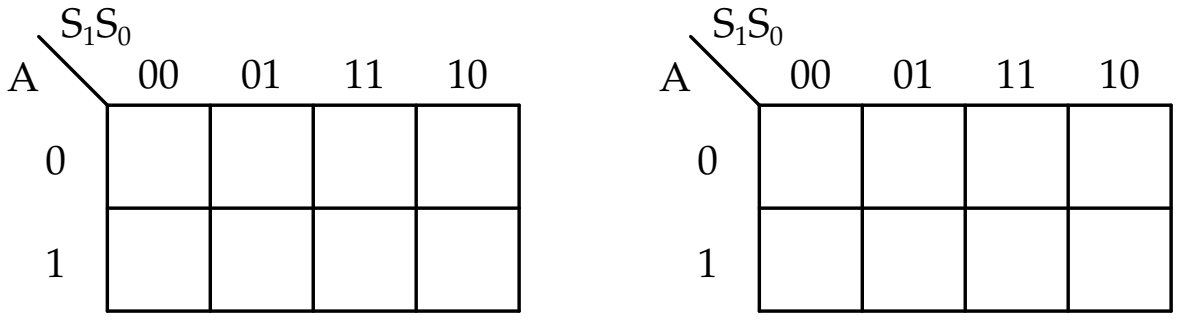
Generalizing this, this finite state machine can be thought of as representing a "divisibility" operator for $n = 2$. That is, it indicates whether a stream of bits representing a binary number is divisble by $n = 2$ or not.

In fact, we can create such finite state machines for any value of n. And, each of these finite state machines *only require n states*. Let's practice by creating another finite state machine for $n = 3$.

Draw a FSM diagram for a n = 3 "divisibility" operator. The reset state is indicated with a double circle. The values of S_0, S_1, Y have already been filled in for you.



Complete the Karnaugh Maps for S_0, S_1.



Part 2.C Gate-level implementation

Please implement the gate-level network for the n = 3 "divisibility" operator.

Gate	t_{pd}	t_{cd}
NOT	1τ	1τ
NAND2	2τ	1τ
NOR2	3τ	1τ
AND2	3τ	1τ
OR2	4τ	1τ
XOR2	7τ	6τ
FF (t_{cq})	9τ	2τ

FF	t_{setup}	10τ
FF	t_{hold}	1τ
T_C		26τ

Use the following table to list every constraint which must be satisfied to ensure correct operation of this gate-level network. You should specify each gate on the path. Assume output ports are unconstrained. **Just write the *final sum* and constraint. Circle any constraints that are violated.**

Path	**1st Constraint**	**2nd Constraint**

Similar to problem 4F in topic 2, we can optimize our gate-level network to meet timing (if it currently isn't)by switching **most** *ANDs and ORs to NAND gates. Think of why this is equivalent (T03).*

Part 2.D A Further Generalization

Why does your $n = 3$ finite state machine work? This is explained by modulo arithmetic.

Each state represents a *modulo equivalence class*. Let v denote the value given by the stream of A. Then, each state *represents* v mod n (i.e. the remainder of v when divided by n). The initial state represents *when the remainder is zero*. By using *modulo* arithmetic, we can derive all of the next state logic.

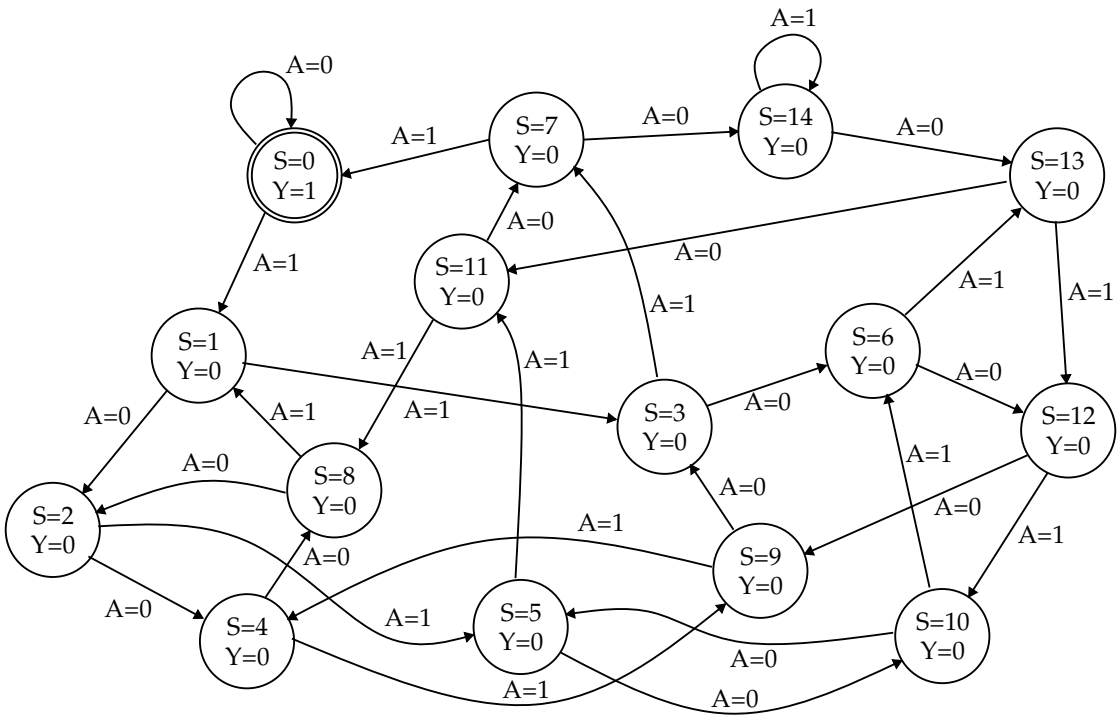
Suppose you are in module state r. Then, if we see a zero, *the remainder doubles*. Thus, the next state we go to is $2r$ mod n. If we see a one, *the remainder doubles* **AND** *we add one*. Thus, the next state we go to is $2r + 1$ mod n. This may be initially confusing, but we will practice with $n = 5$.

Draw the FSM diagram for a n $= 5$ modulo counter ("divisibility" operator).

Part 2.E Representing Larger Numbers

We've explored very small n values thus far. But, what if we wanted to make a finite state machine for $n = 15$? Following our earlier reasoning, we would need 15 states!

This seems like (and is) a lot of work! In fact, the FSM diagram is incredibly complicated:



We should (and will) find a more intelligent approach to building such FSMs. To do so, we will need to exploit the course principles of *modularity, hierarchy, and regularity*. **Fill in the blanks below.**

If a number is divisble by 15, it must also be divisible by _____ and _____.

Let's try to construct a 15-counter with a 3-counter and a 5-counter. In class, you compose FSMs by having one impact the state of another. Now, we will explore a different method. **Create a combinational gate-level network that utilizes the output from the 3-counter, Y_3, and the output from the 5-counter, Y_5 to indicate when an input is divisible by 15 (i.e. Y_{15}).**

Part 2.F A quick comparison

Which of the two constructions of the 15-counter do you prefer? Is one definitely better, and if so, why? Is there any tradeoff between the two? Does this hold for all modular modulo counters?

Part 2.G One Final Optimization

Instead, consider building a modulo 6 counter.

How many states do we need if we use a single FSM? _____ How about modular FSMs? _____

In fact, we can actually build this counter with *just four states*. And, *in a single FSM*. To do this, we exploit the factor of two in six. **Draw the FSM diagram for a 4-state modulo 6 counter.**