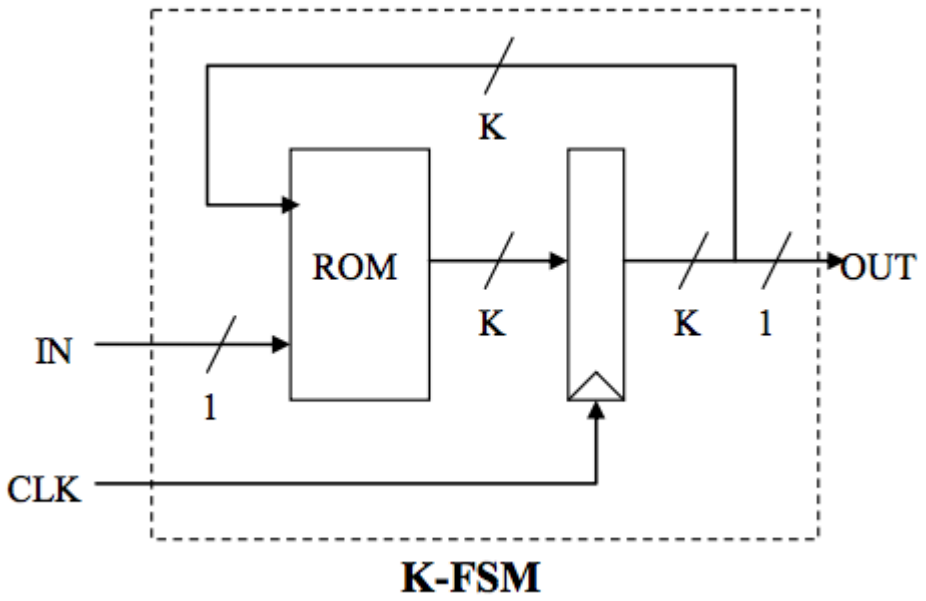
LE6.2

Bookmark this page

LE6.2.1 Number of states

6 points possible (ungraded)

The following is a circuit for a K-FSM, a finite state machine with a single input, a single output and K state bits (one of which is also the output):



(A) What is the maximum number of states a K-FSM may have? Please enter a formula using the variable K. The exponentiation operator is "^".

Formula in terms of K:

2^K

Answer: 2^k

Sorry, couldn't parse formula

Explanation

There are 2^K unique combinations of the K state bits.

(B) Give the number of locations in the ROM and the number of bits in each location. Please enter formulas using the variable K.

Number of locations in the ROM:

2^(k+1)

Answer: 2^(k+1)

Sorry, couldn't parse formula

Number of bits per location:

k

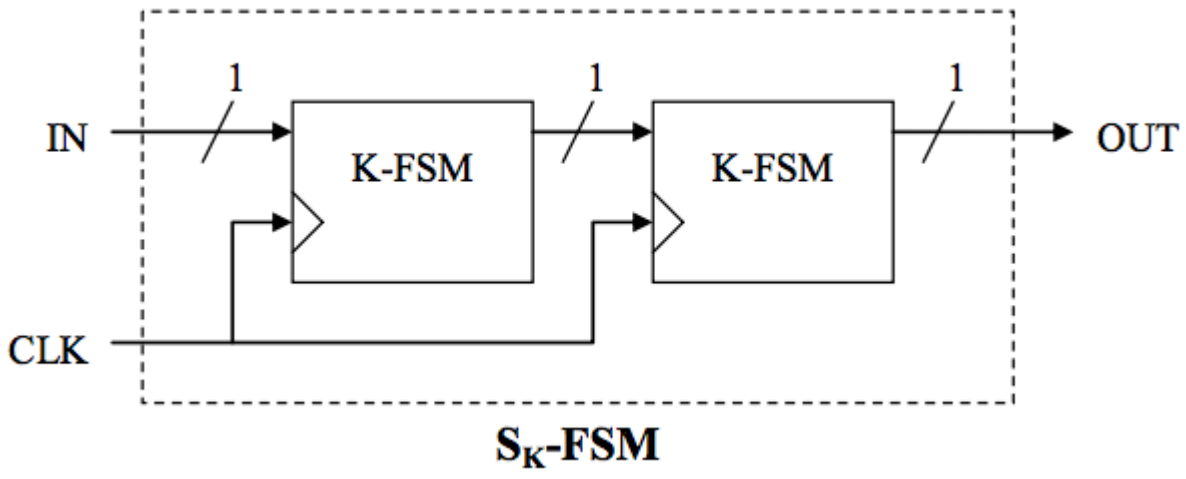
Answer: k

k

Explanation

The ROM has $K+1$ inputs, so it has 2^{K+1} locations. The ROM output is a K-bit signal, so each location must supply K bits.

Consider the connection of two K-FSMs in series to create an S_K-FSM:



Each K-FSM in the diagram has the same size register (K bits) but their ROMs may be programmed differently.

(C) What is the maximum number of states an S_K-FSM may have? Please enter a formula using the variable K.

Formula in terms of K:

Calculator

2^(2*k)

Answer: 2^(2*k)

$2^{2 \cdot k}$

Explanation
The internal K-FSMs are independent, so at any point in time each may be in one of its 2^K states. So the maximum number of states is $2^K \cdot 2^K = 2^{2K}$.

Compare the S_K-FSM above with a (2K)-FSM, a single K-FSM with double the number of state bits.

(D) Can every FSM that can be implemented using an S_K-FSM device also be implemented using a (2K)-FSM device?

- [x] Yes
- [ ] No

Explanation
You can use the lower K bits of the (2K)-FSM state register and the lower K output bits to emulate the left K-FSM inside the SK-FSM; and the high K bits for the right K-FSM.

(E) Can every FSM that can be implemented using a (2K)-FSM device also be implemented using an S_K-FSM device?

- [x] No
- [ ] Yes

Explanation
A (2K)-FSM has a ROM with 2^{2K+1} locations while an S_K-FSM has $2 \cdot 2^{K+1} = 2^{K+2}$ ROM locations, so it's possible to create a transition table in the (2K)-FSM that can't be emulated by a S_K-FSM.

Submit

Discussion

Topic: 6. Finite State Machines / LE6.2

Hide Discussion

Add a Post

All Posts

Unclear what the "state of SK-FSM" is defined to be

discussion posted 8 years ago by **zputin**

The expected answer implies that the combined state of all the K-FSMs is the state of the Sk-FSM.

Its possible to view all the states of all the K-FSMs *preceding the last one* as resolving to just **one bit**. So it can be argued that the output is determined by only 2^(K+1) states i.e., just those of the last K-FSM in the chain.

This post is visible to everyone.

Add a Response

2 respons

Calculator

silvinahw (Staff)
8 years ago

The Sk-FSM consists of 2 K-FSMs in series. There is only 1 output for each K-FSM, but there are still 2^K states associated with each of the K-FSMs. If the first machine can be in states S1-SX. For any one of those states, the second machine can be in one of 2^k states S1-SX. So the total number of states of the larger Sk-FSM is 2^k * 2^k.

That I figured out after I saw the "correct" answer. My point was it wasn't clear from the question what was defined as the "state" of the Sk-FSM. The exhaustive state of every flip flop/memory element in a sequential circuit is generally not very useful (too much data). The subset of 2^(k + 1) states of the last k-FSM in the series is all you really need to know how it will behave.

Just my opinion.

posted 8 years ago by **zputin**

You can't eliminate part of the chain just because it generates 1 bit. As one way to view it, suppose we need to encode the input for the FSM2 using output from the FSM1.

posted 8 years ago by **AlekseiBaryzhikov**

I was not eliminating anything. I was simply pointing out that a large part of the intermediate state can (will?) be ignored if all you want to do is characterize the response of the circuit. Its analogous to ignoring the intermediate states of a ripple carry adder until its final result is available.

posted 8 years ago by **zputin**

If I understand you correctly, you are saying that to know the output of the whole thing one only needs to know the states and transitions of the last FSM in the chain. That is just not true. You can't predict the output of the circuit in general case if you ignore the intermediate state(s), except some special cases. The process is the following. Suppose S_k-FSM consists of a M1→M2 chain. M1 takes input string, makes some transformation, and produces a new string of the same length. M2 does the same in turn, but with it's own pattern. So output string depends on both machines. Nothing can be ignored.

The difference between S_k-FSM and (2K)-FSM is not in the number of states, but in transition functions. In case of (2K)-FSM each transition takes into account the whole 2K-bit state, while the machines of S_k-FSM do not have the information about each other's state and have to rely only on their own K-bit states. It takes away a lot of power and makes a set of possible S_k-FSMs a proper subset of (2K)-FSMs.

posted 8 years ago by **AlekseiBaryzhikov**

You can ignore the intermediate FSMs because their net effect is to output 1 bit. I'm not saying the intermediate functionality can be ignored. I'm not saying you can simulate the Sk-FSM with one K-FSM and an extra bit. I'm simply saying the output of the Sk-FSM is uniquely determined by (k + 1) bits of state.

posted 8 years ago by **zputin**

Well, then I just don't understand what you're saying.

posted 8 years ago by **AlekseiBaryzhikov**

In simple terms, I'm saying there's a lot of state that is redundant if you look at the whole thing like a black box.

posted 7 years ago by **zputin**

Even if an FSM in a chain is just outputting one bit, does not mean its state is redundant at all. See again the binary lock example from the video lecture: the state an FSM is in, could also determine what the output will be in the next cycle, or in ten cycles from now. The situation can be very different with some FSM in a chain being in one state or another. So the state of the wholemachine, is indeed the state of all individual flipflops. Change of of the flipflops in that chain, even in an FSM very far away from the final output point, and the output could after some number of cycles be different.

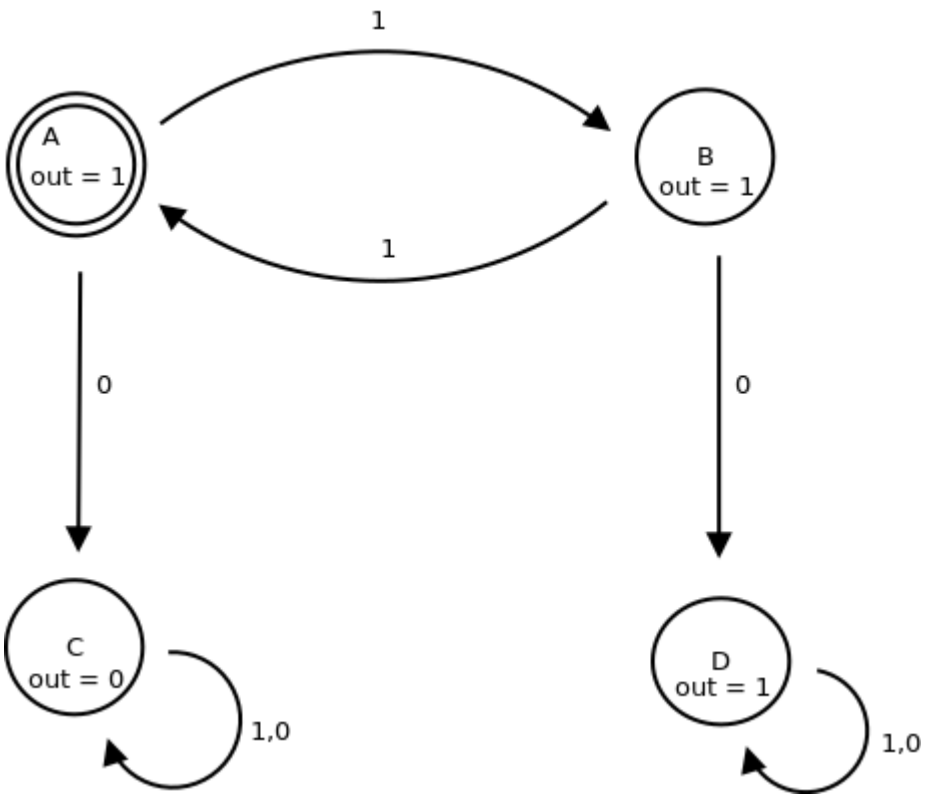
posted 7 years ago by **jandevries**

All those things you mention reduce to just one bit. Note again, we're not trying to reproduce the detailed internal behavior of the system from its state.

posted 7 years ago by **zputin**

No, but we are trying to predict the next state and next output from the current state and the current input, and so forth for more inputs. We cannot do that if the current state is not complete. The current state is not complete if we do not know the states of all the FSMs.

I hope this helps: I made a drawing of an FSM that is in series with another FSM. The other FSM is the last one, the one drawn here is the first. We follow your proposal, and we encoded the current state of the system as a whole as the state of the last FSM, plus one bit which is the last output of this FSM.



The last output of this FSM was 1. Now the next input to the system is 0. What will be the output of the system, and the what will be the next state? If this FSM was in state A, it will then be in state C and output a 0. If it was in state B or C, it will then be in state C and output a 1.

The output thus generated is the input for the next FSM. So this next FSM receives a 1 or a 0, we cannot know which one. Thus we are not sure what its next state, nor its output, will be. That means we do not know the next state or output of the system.

I hope I understood you correctly! Let me know what you think.

posted 7 years ago by **jandevries**

We are not trying to predict anything. That would mean trying to reproduce behavior from state. We are simply saying what *is* not what *was* or *will be* Even from the 2 * (k + 1) bits of state from both FSMs you have no way to tell what happens next.

The (k + 1) bits of state from the last FSM tell you all the possible things that could happen next. This is why I call the other (k + 1) bits from the first FSM "redundant", they tell you nothing more about what could happen next.

posted 7 years ago by **zputin**

Your message has a for me confusing element: in the first paragraph you are arguing that state is not at all helpful in predicting FSM behaviour, and in the second you are arguing that with just (k + 1) bits you *can* tell what happens next. Probably I misunderstood.

Anyway, I disagree with the notion that when you know which state the FSM or set of FSMs is in, you cannot predict what happens next. Actually you can. If an FSM is in a certain state, and you give the next input symbol, you know exactly to what state the FSM will change and what output it will give.

But maybe you meant something differently? Do you mean to say that, if we are only interested in what happens *just one step* from now, knowing the state of the last FSM, plus the input it is about to receive from its preceding FSM is enough? If that is what you mean, yes I agree.

But it does not fully characterize where we are. Maybe you have a different vision of what 'state' means.

As I understand it, a system in situation S1 or the same system in situation S2 is in a different state, if there is an input sequence that would yield a different output if we started in S1 then if we started in S2.

Calculator

With two K-FSMs in series, with just (K + 1) bits of information to encode state, we do not have enough room to encode the state of the first K-FSM. Thus we cannot say how our system will react to the input of, say, 110010100101. If we know the exact state of every K-FSM that is in series, we can say in what state the system is after that input and also what the output that results in.

Put in different words: to construct a truth table of the Sk-FSM, you need $2^{K+1} * 2^{K+1}$ rows. Not just 2^{K+1}.

posted 7 years ago by **jandevries**

ZPutin, you're incorrect on this.

Just because two different states of an FSM yield the same output does *not* mean you can ignore the states of the FSM and treat the output as the state.

The two distinct states may indeed yield the same output, but they may *not* yield the same output on the next clock cycle, or two clock cycles hence, etc. If you ignored their different states and just used the output as a proxy for their states, you would incorrectly deduce that their future streams of outputs would also match, which clearly doesn't have to be the case, as jandevries showed.

In fact, unless the number of outputs is $2^{(\#\text{ of states})}$, you're always going to be forced to have at least two states yielding the same exact output.

posted 7 years ago by **Grimeson**

You need to separate the notion of the system's "state" from its operational behavior. I have said many times, nobody should be trying to recreate the behavior of the system from the state. In fact its is just an artifact of this particular system (it is so simple/simplistic) that it allows you the illusion of being able to recreate the system's behavior from the state.

The reason I say the first (k + 1) bits of state ate "redundant" is that they all resolve to **one bit** which is the +1 in (K + 1) bits driving the last FSM.

The confusion here is how you think about the system's "state" as opposed to how the system is supposed to work. It makes much more sense to think of the system's state as what's left **after** after the system has done its work.

To take your arguments to a logical conclusion, you would have to argue that the **state** of any computation (an algorithm for example) includes **all** of the intermediate results from all of the tiny data manipulations that may have been made as part of the larger algorithm. In practice, all of that intermediate state would be thrown away.

posted 7 years ago by **zputin**

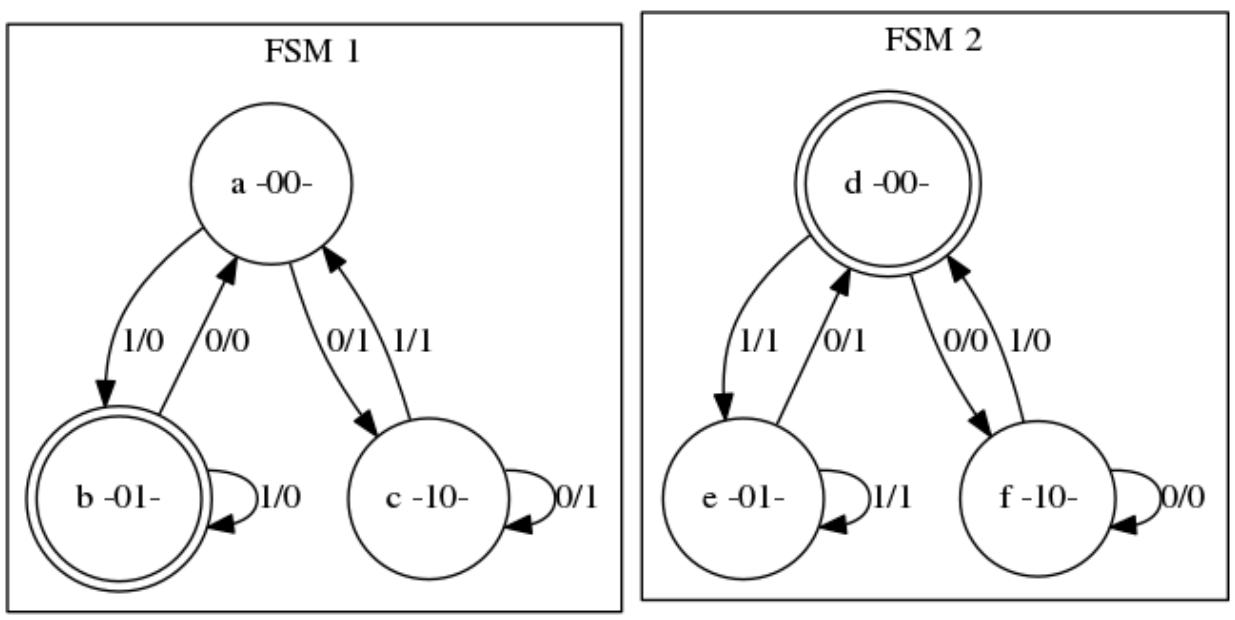
I agree with you that state is more analogous to what's left, than to how a system works. But it is even more analogous to 'where is the system now'.

The state does indeed also not describe how a system works. To keep with the comparison with an algorithm, (a more general entity than a finite state machine): the state of the algorithm, or any computation, is "at which step are we now".

I would like to argue that we do not really know at which step we are in the case of two linked finite state machines, if we only know the state of the downstream FSM and a single bit (I guess encoding the last seen output) concerning the upstream FSM.

I really hope you can find the patience to study the following example in detail. I tried to make something that would illustrate the difference between 'small state', as I call your proposal, and 'large state', as I call the state that is encoding the current node of every FSM.

Consider these two linked K-FSMs where K is 3.



Calculator

input goes into the left FSM, its output is the input for the right FSM, the output of the right FSM is the output of the system. The nodes of the two FSMs have names a - f, and the binary encoding of every node is next to its name. Along the edges is the input/output (Mealy machine).

Say up till now the input 1 0 0 has been processed. FSM-1 has started in node b and moved to nodes b a c, generating output 0 0 1. FSM-2 has started in state d, and its input (the output of FSM-1, thus 0 0 1) has brought it to states f f d.

The 'large state' encoding of this system needs two bits for FSM-1 (10) and two bits for FSM-2 (00). In the 'small state' encoding we only need to know the node of FSM-2, thus 00, and one bit for FSM-1. I am not quite sure what you want with that one bit but I presume it will be the last seen output, thus 1.

The 'small state' encoding of 1 - 00 does not unambiguously tell us at which position the system is. If the next input is 1, we have no idea what the system will do. Since the last output of FSM-1 is 1, it could be in states a or c. The next input could thus generate an output of 0 (a → b) or 1 (c → a). We know from the encoding of 00 that FSM-2 is at node d, an input to FSM-2 of 1 would give a system output of 1 (d → e), an input to FSM-2 of 0 would give a system output of 0 (d →f).

Thus the 'small state' encoding cannot unambiguously tell us at which step the system is. The practical consequence of this, is that we cannot build a truth table of the system with just the 'small state' bits as inputs.

To return to the analogy with algorithms or computations in general, with the 'small state' encoding there is no certain way to tell till which step the computation or algorithm has progressed. Part of it is known, but still several options are possible.

It is as if encoding the state of a **single** FSM that has 16 states with just 3 bits. To encode 16 states 4 bits are needed. With the three bits there is a lot of information, we can rule out several nodes, but we cannot be certain at which node the FSM is now.

It is difficult to convey feelings through forum messages, but I have the feeling I am annoying you. If that is so, feel free to say that, I can if you like stop posting in this thread. Second thing what I would like to say, is that I hope you are right about the 'small state', it would actually save a lot of ROM locations for instance.

posted 7 years ago by **jandevries**

Add a comment

Grimeson
7 years ago

> You need to separate the notion of the system's "state" from its operational behavior.

No, the question was about the system's state.

> I have said many times, nobody should be trying to recreate the behavior of the system from the state.

The problem is asking specifically about the number of states, not about the best practical way to recreate system behavior.

> The confusion here is how you think about the system's "state" as opposed to how the system is supposed to work.

There's only one way to think about a system's state, it's a clearly defined term.

> To take your arguments to a logical conclusion, you would have to argue that the state of any computation (an algorithm for example) includes all of the intermediate results from all of the tiny data manipulations that may have been made as part of the larger algorithm.

That may or may not be true, depending on the FSM. The concept of state answers precisely the question of what is or is not "thrown away" when performing a task.

Once again, your initial claim, that the output of the previous K-FSM is all you need to determine its state, is patently wrong. That output bit is **not** an input to the system; the input is on the left-side of the entire Sk-FSM chain. Therefore, the state of the left K-FSM iindeed can effect behavior of the system in a way that that one output bit doesn't capture on its own.

The ROMs in the FSM implement the **system logic**. This is the first time I've encountered the notion that **logic** is

Calculator

state. I believe the confusion is as follows: This exercise makes the point that in order to maintain state, you need **memory**. This is absolutely true.

The giant leap from there that **therefore all memory is state** is in my opinion, a completely useless notion i.e., it buys you absolutely nothing. Are the terabytes of system memory on you laptop the system's state? What about the CDs, DVDs, Flash drives, etc ? So "State" becomes just another word for "Memory"?

I also absolutely reject the notion that the operating logic of a system is its state.

All your other arguments indicate that you have an extremely rigid notion of a system\'s state. There is no hard definition of system state. It is what you choose. The choice is always determined by practical usefulness e.g., when you use that state to drive subsequent behavior of the FSM. So a **previous** state is used like an input to derive a **next** state and so on.

posted 7 years ago by **zputin**

One could indeed argue that what state is precisely, is not rigidly defined. I agree wholeheartedly that a practical notion of state would be that the previous state is like an input to derive the next state.

But from this follows that in the current example, we need to know from all the FSMs at which node they currently are, to be able to compute the next state. This is not the same as if we need to take the setup of the FSM as state. Just, from every FSM, in what node are they currently.

If that is not known, we cannot derive the bit that the upstream FSM will show in the next state to the downstream FSM.

posted 7 years ago by **jandevries**

Add a comment

‹ Previous Next ›

Calculator