

Логичко пројектовање

Предавање 4/10

Секвенцијалне мреже

Пројектовање секвенцијалних мрежа

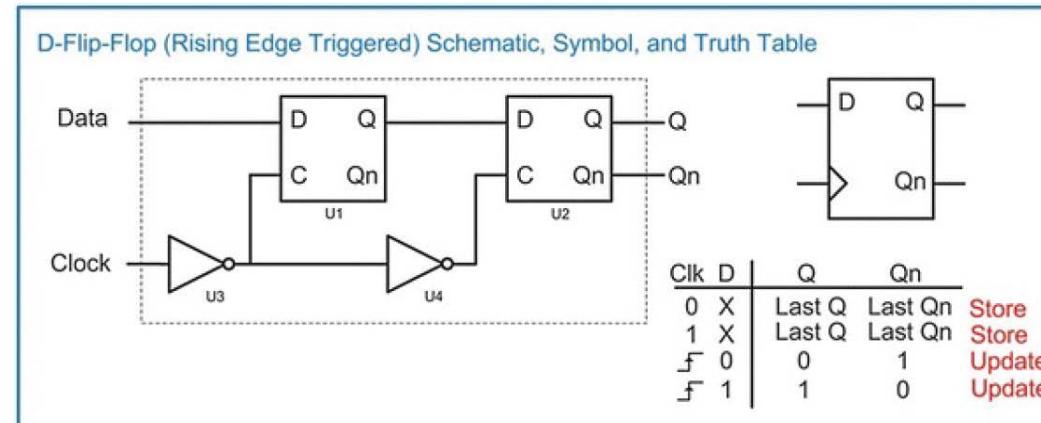
- Пројектовање секвенцијалних мрежа се разликује од пројектовања комбиноционих мрежа по томе што излази кола зависе не само од тренутних вредности улаза већ и од претходних вредности улаза.
- Ово се разликује од пројектовања комбиноционе логике где излаз кола зависи само од тренутних вредности улаза.
- Способност секвенцијалних мрежа за где излази зависе од тренутних и претходних улаза омогућава стварање софистициранијих и интелигентнијих система.
- Секвенцијални логички елемент за меморисање претходних вредности улаза
- Коначни аутомати (FSM – Finite state machine)
- Логички елементи за меморисање:
 - Унакрсно спрегнути инверторски пар (The Cross-Coupled Inverter Pair)
 - SR леч (SR Latch)
 - S'R' леч (S'R' Latch)
 - SR леч са Enable линијом (SR Latch with Enable)
 - D леч (D Latch)
 - D флип-флоп (D-Flip-Flop)

D флип-флоп (1/4)

- Најчешће коришћени меморијски елемент у секвенцијалној логици је D флип-флоп.
- D флип-флоп је сличан у понашању као D леч са изузетком да се режим за меморисање покреће прелазом или ивицом clock сигнала уместо нивоом clock сигнала.
- Ово омогућава D флип-флопу да имплементира системе виших фреквенција јер излази се ажурирају за краће време.
- Шема, симбол и таблица истинитости дата је на слици.
- Да би се означила осетљивост D флип-флопа на предњу ивицу, улаз за clock сигнал се означава са „>“.
- U3 инвертор на овој шеми креира понашање растуће ивице.
- Ако би U3 био изостављен, ово коло би било Д-флип-флоп изазван негативном ивицом.

D флип-флоп (2/4)

- D флип-флоп шема приказана на слици се назива master/slave конфигурација због начина на који се подаци преносе кроз ове леч елементе (U1 и U2).
- Због U4 инвертора, лечеви ће увек бити комплементарни. Када је U1 у режиму чекања, U2 ће бити у режиму праћења и обрнуто. Када clock сигнал пређе на HIGH, U1 ће сачувати последњу вредност података. Током времена када је clock сигнал HIGH, U2 ће ући у режим праћења и пренети вредност излаз Q. На овај начин, подаци се затварају у уређај за складиштење на растућу ивицу clock-а и присутни су на излазу Q.
- Ово је master операција флип-флопа јер U1, или први D-леч држи вредност, а други D-леч (slave) једноставно прослеђује ову вредност на излаз Q.



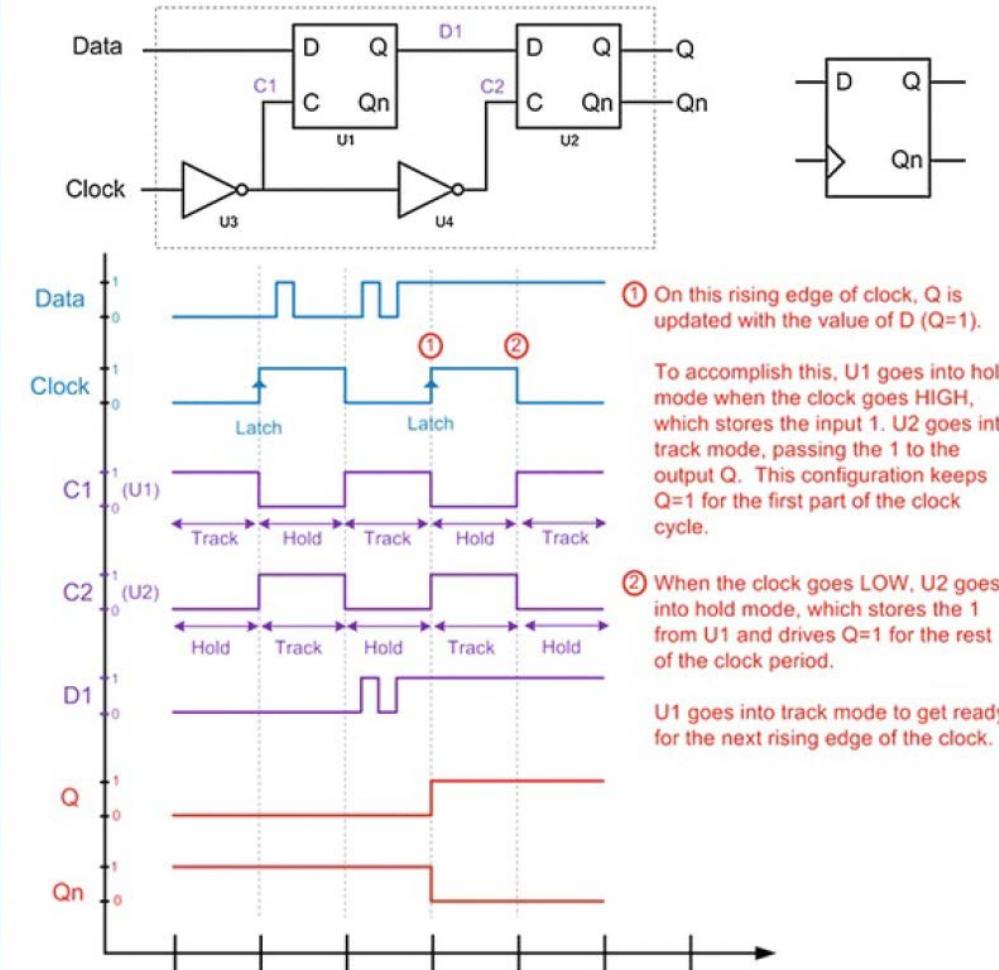
D флип-флоп (3/4)

- Када clock прелази у LOW, U2 ће сачувати излаз U1.
- Пошто постоји коначно кашњење кроз U1, U2 D-леч је у стању да сачува вредност пре него што U1 потпуно уђе у режим праћења.
- U2 ће држати вредност на Q за време док је clock LOW.
- Ово је slave операција флип-флопа јер U2, тј. други D-леч, држи вредност. За време када је clock LOW, U1 је у режиму праћења, што прослеђује улазне податке у средину D-флип-флопа припремајући се за следеће подизање ивице clock-а.
- Конфигурација master/slave ствара понашање где се Q излаз D флип-флопа ажурира са вредношћу улаза D, само кад је clock на растућој ивици. У свим осталим временским тренуцима, Q држи последњу вредност D.
- Пример времена дијаграм за рад Д-флип-флопа са растућом ивицом дат је на следећој слици.
-

D флип-флоп (4/4)

- Додатни сигнали:
- D-flip-flop with asynchronous reset and preset
- D-flip-flop with synchronous enable

D-Flip-Flop (Rising Edge Triggered) Timing Diagram



Коначни аутомати

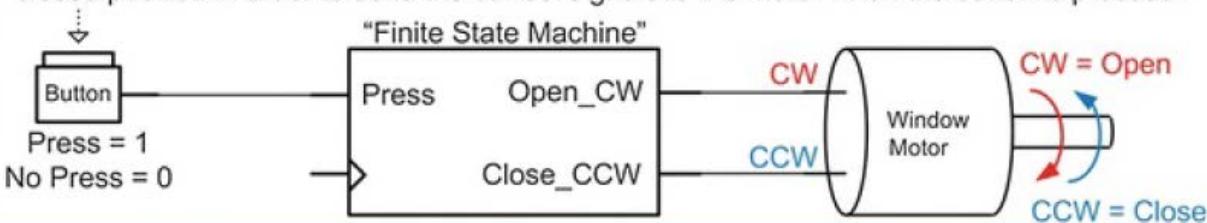
- Сада скрећемо пажњу на једно од најмоћнијих секвенцијалних логичких кола, коначни аутомат (FSM).
- FSM је коло које садржи унапред дефинисан број стања (тј. коначан број стања). Оно може бити у једном и само једном стању у исто време. Коло прелази између стања на основу покретања неког догађаја, најчешће ивица clock-а, или вредношћу неког улаза аутомата.
- Број стања и свих могућих прелаза су унапред дефинисани.
- Коришћењем стања и унапред дефинисаног низа прелаза, коло је у стању да доноси одлуке о следећем стању у које ће прећи на основу историје прошлих дешавањ.
- Ово омогућава колу да креира излазе који су више интелигентнији у поређењу са једноставним комбинационим логичким колом које има излазе само на основу тренутних вредности улаза.

Опис FSM-а

- Пројектовање коначног аутомата почиње апстрактним описом речима жељеног понашања кола. Користићемо пример пројектовања контролера за мотор са "push" дугметод.

Example: Push-Button Window Controller - Word Description

Design a system that will allow a user to open and close a window with the push of a button. The window is connected to a motor that has two inputs. The first input to the motor is asserted when the motor needs to spin in a clockwise (CW) direction to open the window, while the second input is asserted when the motor needs to spin in a counterclockwise (CCW) direction to close the window. The signals to the motor do not need to be held for the duration of the window opening/closing. Once the motor observes an assertion on one of its inputs, it will spin until the window is in the correct position and then stop. The inputs are not allowed to be asserted at the same time. The user will press a single button to either open or close the window so the system must keep track of whether the window is in the open or closed position in order to send the correct signals to the motor when the button is pressed.



Дијаграми стања

- Дијаграм стања је графички начин да се описе функционалност FSM-а. Дијаграм стања је облик усмереног графа, у коме свако стање унутар систем се означава као круг и даје му описано име. Имена су написано унутар кругова.
- Прелази између стања се означавају помоћу стрелица са написаним улазним условима који изазивају прелазе из једног стања у друго.
- Прелази се могу завршити у друго стање након одређеног уноса улаза или остати у истом стању.
- Ако се врши имплементација са секвенцијалним меморијским елементом, врши се евалуација када треба прећи у ново стање, тј сваки пут када меморијски елементи ажурирају своје излазе.
- На пример, ако се користи D-флип-флоп који се активира узлазном ивицом, онда се евалуација дешавал на сваку растућу ивицу clock-а.
- Постоје два различита типа излазних услова за аутомат.
- Први је када излаз зависи само од тренутног стања аутомата. Овај тип аутомата се назива Муров аутомат. У овом случају, излази система пишу се унутар кругова стања. Ово означава излазну вредност која ће бити генерисан за свако одређено стање.
- Други излазни услов је када излази зависе и од тренутног стања и од улаза. Ова врста аутомата се зову Милијеви аутомати. У овом случају, излази система су написани поред прелаза стања који одговарају одговарајућим улазним вредностима. Излази у дијаграму стања се обично пишу унутар заграда.

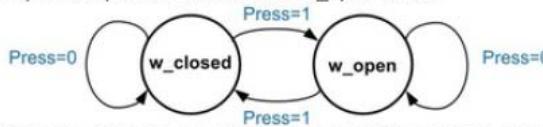
Дијаграми стања - пример

Example: Push-Button Window Controller - State Diagram

1) Defining the States - For this design, we will define two finite states. The first state is when the window is in the closed position. Let's call this state "w_closed". The second state is when the window is in the open position. Let's call this state "w_open". Each of these two states will be represented in the state diagram as circles. The names of the states are written inside of the circles.



2) Defining the Transitions - We now describe the transitions between states using arrows and labeling the arrows with the input conditions that trigger each transition. For this design, when the machine is in the "w_closed" state, a button press (Press=1) will cause a transition to the "w_open" state. When the button is not pressed, the machine will remain in the "w_closed" state (Press=0). When the machine is in the "w_open" state, a button press (Press=1) will cause a transition to the "w_closed" state, while the button not being pressed (Press=0) will keep the machine in the "w_open" state.



3) Defining the Outputs - We now describe the outputs of the system. For this design, the system will output the appropriate motor control signals upon a button press. This means that the outputs depend on both the current state and the current inputs. This is by definition a *Mealy Machine*. As such, the outputs are listed next to the state transitions. By listing the outputs in this location, both the current state and the input values producing the outputs are indicated. When this machine is in either the w_closed or w_open states and the button is NOT pressed, the outputs Open_CW and Close_CCW are both 0's. When the machine is in w_closed state and the button is pressed, the Open_CW output is asserted to rotate the motor clockwise and open the window. When the machine is in w_open state and the button is pressed, the Close_CCW output is asserted to rotate the motor counterclockwise and close the window. The final state diagram for this system is shown below.

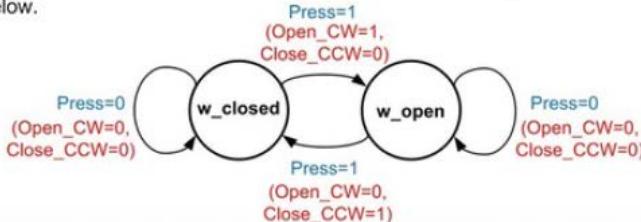


Таблица прелаза стања

- Дијаграм стања се сада може описати у облику таблице која је слична таблици истинитости.
- Ово ставља понашање коначног аутомата у облик који чини логичку синтезу једноставном. Табела садржи исте информације као у дијаграму стања.
- Стање у коме аутомат постоји назива се тренутно стање. За свако тренутно стање у које аутомат може да се налази, сви могући улази су написани поред стања заједно са одредишним стањима.
- Одредишно стање за прелаз назива се следеће стање. У табели су такође наведени излази који одговарају сваком тренутном стању и улазу, у случају Милијевог аутомата.
-

Таблица прелаза стања - пример

Example: Push-Button Window Controller - State Transition Table

A state transition table contains the same information as the state diagram but in a tabular format. This format is similar to a truth table and makes logic synthesis straight forward. Each state and input condition is listed in the table along with the corresponding next state and outputs.

(Input)

(Outputs)

Current State	Press	Next State	Open_CW	Close_CCW
w_closed	0	w_closed	0	0
w_closed	1	w_open	1	0
w_open	0	w_open	0	0
w_open	1	w_closed	0	1

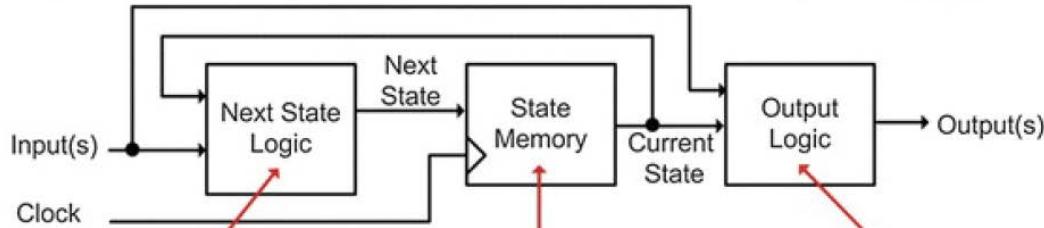
Логичка синтеза аутомата

- Једном када је описано понашање аутомата, аутомат може бити директно синтетизован. Постоје три главне компоненте аутомата, меморија аутомата, логика следећег стања и излазна логика.
- Слика на следећем слайду приказује блок дијаграм аутомата који истиче све три компоненте.
- Логички блок следећег стања је група комбинационе логике која производи сигнале следећег стања на основу тренутног стања и свих системских улаза.
- Меморија стања држи тренутно стање система. Тренутно стање се ажурира са следећим стањем на сваком растућа ивица clock-а, која је означена симболом „>“ унутар блокова.
- Ово понашање је креирано коришћењем D флип-флопова где се одржава тренутно стање на Q излазима D флип-флопова, док је следеће стање присутно на D улазима. На овај начин ће се свака растућа ивица clock-а покренути евалуацију у које следеће стање треба прећи. Ова одлука је заснована на основу тренутног стања и тренутних улаза.
- Излазни логички блок је група комбинационе логике која креира излазе система. Овај блок увек користи тренутно стање као и улаз у зависности од типа машине (Милијев или Муров).
- Код Муровог аутомата излазни блок користи следеће стање.

Логичка синтеза аутомата – главне компоненте

Main Components of a Finite State Machine

Mealy Machine – The output(s) depend on both the current state and system input(s).

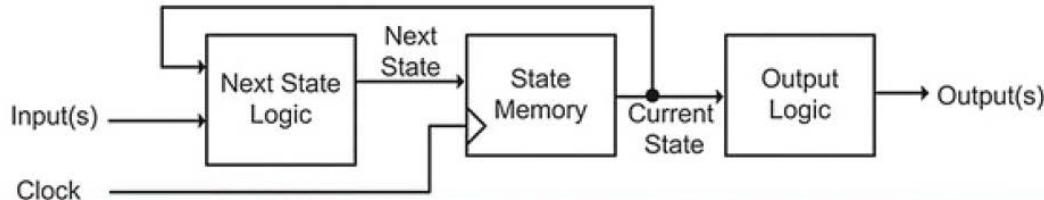


The next state logic creates the signal "next state" based on the current state and any system inputs. This block is implemented with combinational logic.

The state memory holds the current state. The current state is updated with "next state" on the rising edge of the clock. This block is implemented with D-Flip-Flops.

The output logic creates the system outputs. The output logic always depends on the current state of the machine and optionally (Mealy vs. Moore) the inputs of the system.

Moore Machine – The output(s) depends only on the current state.



Меморисање стања

- Меморија стања је коло које ће држати тренутно стање машине.
- Растућа ивица clock-а ажурираће тренутно стање са следећим стањем.
- У свим осталим временима, следећи унос стања се игнорише. Ово даје времена за следеће логичко коло стања за израчунавање резултата за следеће стање.
- Ово понашање је идентично понашању D флип-флопа, стога је меморија стања једноставно један или више D флип-флоп.
- Број потребних D флип-флопова зависи од тога како су стања кодирана.
- Кодирање стања је процес додељивања бинарне вредности описном имену стања из дијаграма стања и табеле прелаза стања.
- Једном када су дескриптивна имена претворена у репрезентативне кодове користећи 1 и 0, стања се могу имплементирати у стварна кола.
- Додела кодова стањима су произвољна и могу се бирати како би се минимизирала потребан број логичких кола у аутомату.
- Постоје три главна стила кодирања стања.

Меморисање стања - број флип-флопова

Solving For the Number of Bits Needed for Binary State Encoding

Problem: You are designing a state machine that has 41 unique states and are going to encode the states in binary. How many D-Flip-Flops do you need?

Solution: Each D-Flip-Flops will hold one bit of the state code. If the state memory has n-bits, it can encode 2^n states using binary encoding. We can use logarithms in order to solve for the n in the exponent.

$$2^n = (\# \text{ of states})$$

$$\log(2^n) = \log(\# \text{ of states})$$

$$n \cdot \log(2) = \log(\# \text{ of states})$$

$$n = \frac{\log(\# \text{ of states})}{\log(2)}$$

$$n = \frac{\log(41)}{\log(2)}$$

$$n = 5.36$$

Rounding up to the next whole number means that we need 6 bits, or 6-D-Flip-Flops to encode 41 states in binary.

Check: To check this, let's plug 6 back into the original expression. If we have 6 bits, we can encode 2^6 states, or 64 states. This is enough to encode our 41 states. If we had 1 less bit (e.g., 5), we could only encode up to $2^5=32$ states, so we require 6 bits for this state encoding. Note that not all of the possible binary values are used as state codes.

Меморисање стања

- Греј код шаблон

Creating an n-bit Gray Code Pattern

A gray code sequence begins with the known 2-bit pattern of 00, 01, 11, 10.

In order to increase the number of bits, the existing pattern is mirrored across an imaginary horizontal axis below the existing pattern. The bits above the axis are padded with leading 0's, and the bits below the axis are padded with leading 1's. This turns a 2-bit gray code pattern into a 3-bit pattern preserving the characteristic that each code only differs by its neighbor by one bit.

This process is repeated to create a 4-bit gray code pattern.

2-bit Gray Code Pattern

00
01
11
10

3-bit Gray Code Pattern

Pad the upper bits with leading 0's	000 001 011 010 110 111 101 100	Pad the lower bits with leading 1's	Mirror across this axis
-------------------------------------------	------------------------------------------------------	-------------------------------------------	----------------------------

Меморисање стања

- Кодирање стања

Comparison of Different State Encoding Approaches

A state machine has eight unique states named S0, S1, ... S7. The following is an example of how these states can be encoded using binary, gray code and one-hot.

<u>State Name</u>	<u>Binary</u>	<u>Gray Code</u>	<u>One-Hot</u>
S0	000	000	00000001
S1	001	001	00000010
S2	010	011	00000100
S3	011	010	00001000
S4	100	110	00010000
S5	101	111	00100000
S6	110	101	01000000
S7	111	100	10000000

Меморисање стања

- пример

Example: Push-Button Window Controller - State Encoding

This state machine contains two states, w_closed and w_open. The following are the three possible ways these states could be encoded.

<u>State Name</u>	<u>Binary</u>	<u>Gray Code</u>	<u>One-Hot</u>
w_closed	0	0	01
w_open	1	1	10

Since this machine is so small, there is no difference between the binary and gray code approaches. Both of these techniques will require one D-Flip-Flop to hold the state code. The one-hot approach will require two D-Flip-Flops. Let's choose binary state encoding for this example. Let's use the state variable names Q_cur and Q_nxt.

Once the state codes and state variables are chosen, the state transition table is updated with the new detailed information about the design.

Current State		Input	Next State		Outputs	
	Q_cur	Press		Q_nxt	Open_CW	Close_CCW
w_closed	0	0	w_closed	0	0	0
w_closed	0	1	w_open	1	1	0
w_open	1	0	w_open	1	0	0
w_open	1	1	w_closed	0	0	1

Логика следећег стања

Example: Push-Button Window Controller - Next State Logic

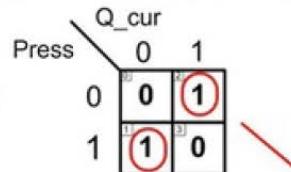
We need to synthesize the combinational logic circuit that will create the next state logic for Q_nxt. The behavior of this combinational logic circuit is described in the state transition table. In order to visualize where this information is within the table, let's pull it out and put it into a traditional truth table format.

Current State		Input	Next State		Outputs	
	Q_cur		Press	Q_nxt	Open_CW	Close_CCW
w_closed	0	Press	w_closed	0	0	0
w_closed	0		w_open	1	1	0
w_open	1	Press	w_open	1	0	0
w_open	1		w_closed	0	0	1

These columns are the inputs to the next state logic.

This column is the desired output for the next state logic variable Q_nxt.

Q_cur	Press	Q_nxt
0	0	0
0	1	1
1	0	1
1	1	0



$$\begin{aligned}
 Q_{\text{nxt}} &= (Q_{\text{cur}}' \cdot \text{Press}) + (Q_{\text{cur}} \cdot \text{Press}') \\
 \text{or} \\
 Q_{\text{nxt}} &= Q_{\text{cur}} \oplus \text{Press}
 \end{aligned}$$

Излазна логика

Example: Push-Button Window Controller - Output Logic

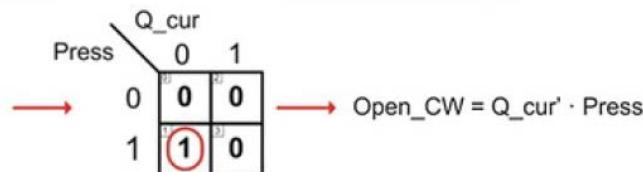
We need to synthesize the combinational logic circuits that will create the output logic for the signals "Open_CW" and "Close_CCW". The behavior of this combinational logic circuit is described in the state transition table. Again, in order to visualize where this information is within the table, let's pull it out and put it into traditional truth table formats.

Current State		Input	Next State		Outputs	
	Q _{cur}		Press	Q _{nxt}	Open_CW	Close_CCW
w_closed	0	0	w_closed	0	0	0
w_closed	0		w_open	1	1	0
w_open	1	1	w_open	1	0	0
w_open	1		w_closed	0	0	1

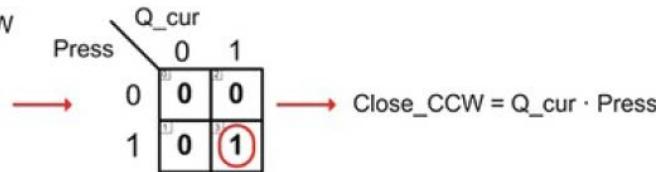
These columns are the inputs to the output logic.

These columns are the desired behavior of the outputs.

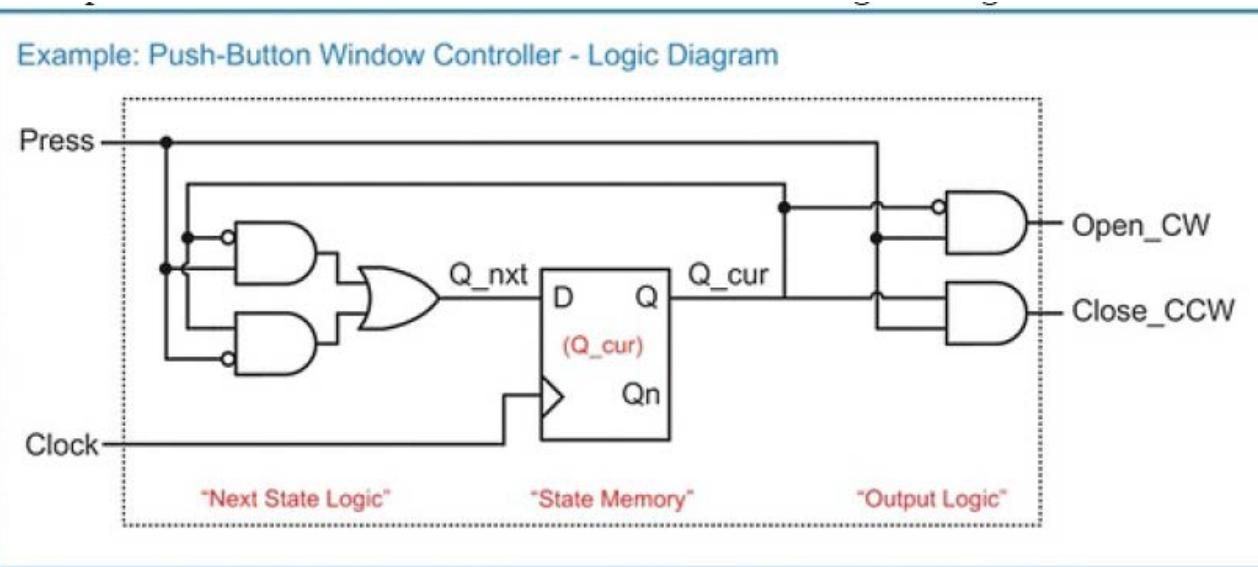
Q _{cur}	Press	Open_CW
0	0	0
0	1	1
1	0	0
1	1	0



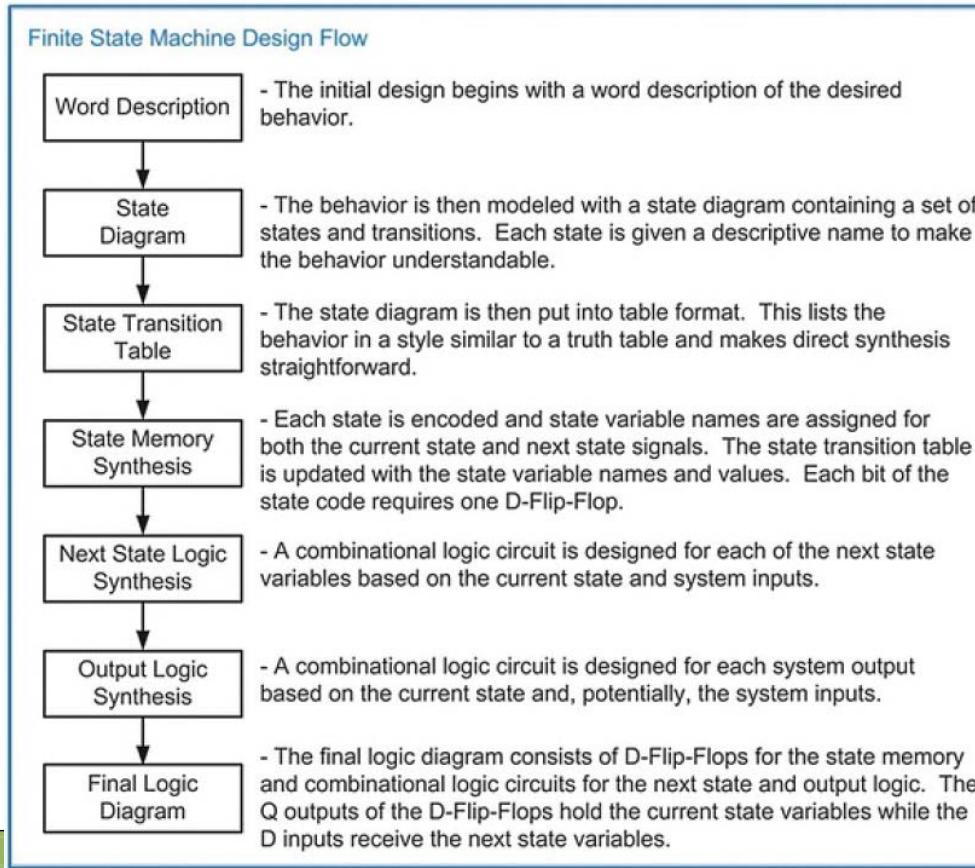
Q _{cur}	Press	Close_CCW
0	0	0
0	1	0
1	0	0
1	1	1



Логичка мрежа



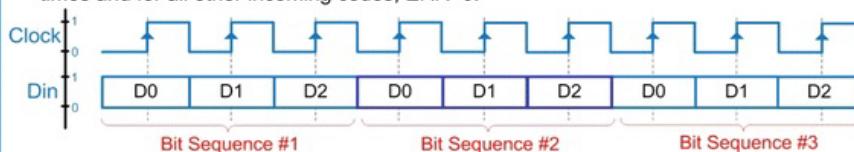
Пројектовање аутомата - преглед



Example: Serial Bit Sequence Detector (Part 1)

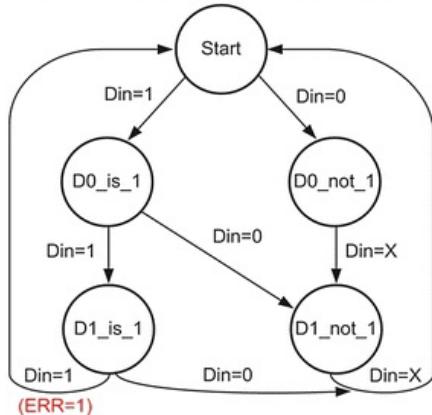
Word Description

We are going to design a circuit that will monitor an incoming serial bit stream. The information in the bit stream represents data in groups of three bits. The code "111" represents that an error has occurred in the transmitter. Our system needs to monitor the incoming bit stream and assert a signal called "ERR" if the sequence "111" is detected. At all other times and for all other incoming codes, $ERR=0$.



State Diagram & State Transition Table

To implement this design, we need a machine that can keep track of the number of incoming bits. In this way, the machine will know once the three bits within a sequence have been received. The machine must also track if the sequence of incoming bits are 1's. In order to do this, let's create a sequence of states that will be traversed when $Din=1$. We also need a parallel sequence of states that will be traversed if an incoming bit is ever a 0. Each of these parallel paths must contain enough states to track that three bits of the sequence have been received before starting over and monitoring the next incoming sequence. The only time the output ERR will be asserted is when three 1's are received within one three bit data sequence. To simplify the state diagram, the output of $ERR=1$ is only listed once next to the corresponding transition in the diagram. It is assumed that at all other times, $ERR=0$.



Current State	Din	(Input)		(Output)	
		Next State	ERR	Q2_cur	Q1_cur
Start	0	$D0_not_1$	0		
Start	1	$D0_is_1$	0		
$D0_is_1$	0	$D1_not_1$	0		
$D0_is_1$	1	$D1_is_1$	0		
$D1_is_1$	0	Start	0		
$D1_is_1$	1	Start	1		
$D0_not_1$	0	$D1_not_1$	0		
$D0_not_1$	1	$D1_not_1$	0		
$D1_not_1$	0	Start	0		
$D1_not_1$	1	Start	0		

Example: Serial Bit Sequence Detector (Part 2)

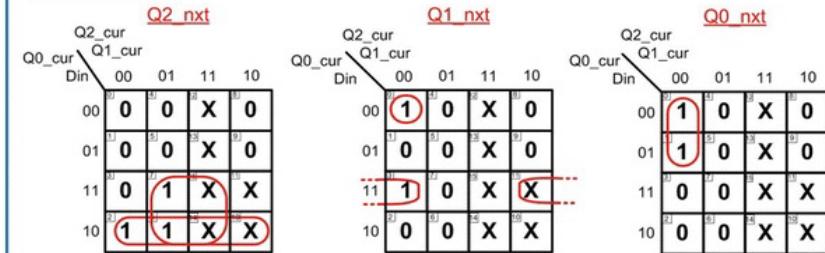
State Encoding

Let's encode the states in binary in order to minimize the number of D-Flip-Flops. Encoding in Gray Code will not benefit this design since the state transitions are not linear. Since there are 5 unique states, we'll need 3 bits to encode all of the states. At this point, we also need to assign the state variable names. Let's call the three variables for the current state $Q2_{cur}$, $Q1_{cur}$, and $Q0_{cur}$. Let's call the three variables for the next state $Q2_{nxt}$, $Q1_{nxt}$, and $Q0_{nxt}$. After the state codes are assigned, we can update the state transition table.

State	Code
Start	"000"
$D0_is_1$	"001"
$D1_is_1$	"010"
$D0_not_1$	"011"
$D1_not_1$	"100"

Current State	Input	Next State			Output
		$Q2_{nxt}$	$Q1_{nxt}$	$Q0_{nxt}$	
Start	0	0	1	1	0
Start	1	0	0	1	0
$D0_is_1$	0	0	1	0	0
$D0_is_1$	1	0	1	0	0
$D1_is_1$	0	1	0	0	0
$D1_is_1$	1	0	1	0	1
$D0_not_1$	0	1	1	0	0
$D0_not_1$	1	0	1	1	0
$D1_not_1$	1	0	0	0	0
$D1_not_1$	1	0	0	1	0

Next State Logic



$$\rightarrow Q0_{nxt} = (Q2_{cur} \cdot Q1_{cur} \cdot Q0_{cur})$$

$$\rightarrow Q1_{nxt} = (Q2_{cur} \cdot Q1_{cur} \cdot Q0_{cur} \cdot \text{Din}') + (Q1_{cur} \cdot Q0_{cur} \cdot \text{Din}')$$

$$\rightarrow Q2_{nxt} = (Q1_{cur} \cdot Q0_{cur}) + (Q0_{cur} \cdot \text{Din})$$

Секвенцијалне мреже

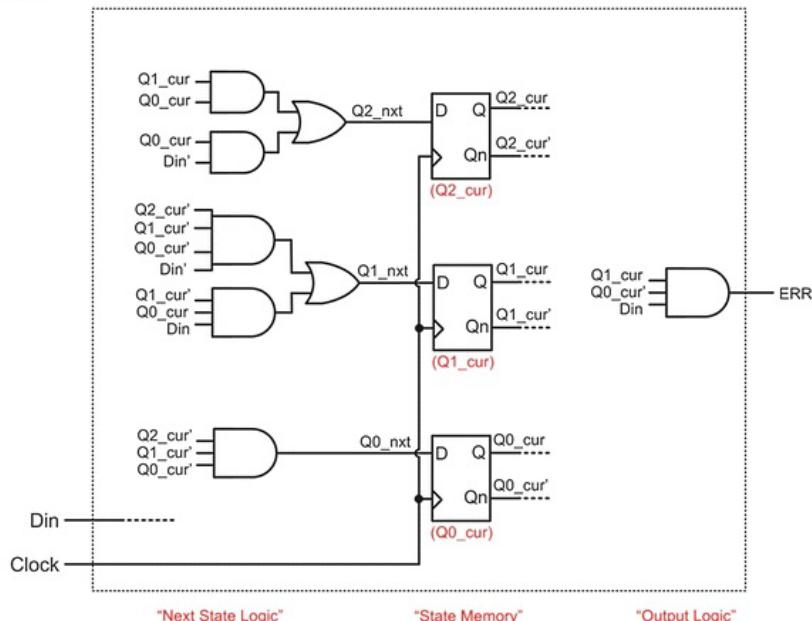
Example: Serial Bit Sequence Detector (Part 3)

Output Logic

		ERR	
		Q2_cur	Q1_cur
		Q0_cur	Din
00	00	0	0
01	01	1	X
11	11	0	X
10	10	0	X

→ $ERR = Q1_{cur} \cdot Q0_{cur'} \cdot Din$

Logic Diagram

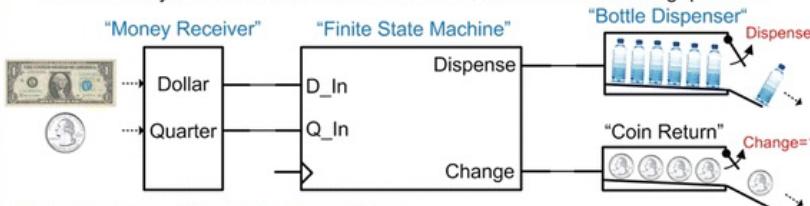


Note that many of the wires are not drawn in to make the diagram readable. This is a common practice. Nodes with the same name are assumed to be connected regardless of whether a wire is drawn.

Example: Vending Machine Controller (Part 1)

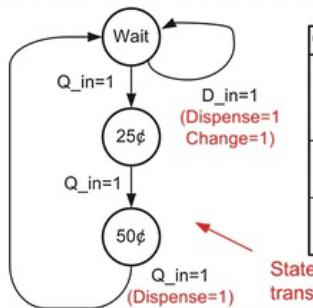
Word Description

We are going to design a simple vending machine controller. The vending machine will sell bottles of water for 75¢. Customers can enter either a dollar bill or quarters. Once a sufficient amount of money is entered, the vending machine will dispense a bottle of water. If the user entered a dollar it will return one quarter in change. A "Money Receiver" detects when money has been entered. The receiver sends two logic signals to our circuit indicating whether a dollar bill or quarter was received. A "Bottle Dispenser" system holds the water bottles and will release one bottle when its input signal is asserted. A "Coin Return" system holds quarters for change and will release one quarter when its input signal is asserted. The money receiver will reject money if a dollar and quarter are entered simultaneously or if a dollar is entered once the user has started entering quarters.



State Diagram and State Transition Table

To implement this state machine, we will need an initial state that the machine will wait until a customer enters money (Wait). If a dollar is entered, the machine will assert the "Dispense" signal to release a bottle of water and assert the "Change" signal to give one quarter in change. We do not need an additional state for the condition of when a dollar is entered because the machine will simply assert the output signals and return to the Wait state. When the customer pays with quarters, our machine needs to keep track of how many quarters have been received. We'll need two interim states that keep track of how many quarters have been entered (25¢ and 50¢). Once the third quarter has been entered, our machine will assert the "Dispense" signal and return to the Wait state.



Current State	Q _{in}	D _{in}	Next State	Dispense	Change
Wait	0	0	Wait	0	0
Wait	0	1	Wait	1	1
Wait	1	0	25¢	0	0
25¢	0	X	25¢	0	0
25¢	1	X	50¢	0	0
50¢	0	X	50¢	0	0
50¢	1	X	Wait	1	0

State diagrams can be simplified by only drawing transitions when a signal is asserted.

Example: Vending Machine Controller (Part 2)

State Encoding

Let's encode the states in binary and name the current state variables Q₁_cur and Q₀_cur and the next state variables Q₁_nxt and Q₀_nxt. In this table we list out all possible values the current state and the inputs to make the table more complete.

State	Code	Current State		Input		Next State		Outputs	
		Q ₁ _cur	Q ₀ _cur	Q _{in}	D _{in}	Q ₁ _nxt	Q ₀ _nxt	Dispense	Change
Wait	= "00"	0	0	0	0	Wait	0	0	0
Wait	= "01"	0	0	0	1	Wait	0	0	1
Wait	= "10"	0	0	1	0	25¢	0	1	0
25¢	= "00"	0	0	1	1	Wait	0	0	0
25¢	= "01"	0	1	0	0	25¢	0	1	0
25¢	= "10"	0	1	0	1	25¢	0	1	0
25¢	= "11"	0	1	1	0	50¢	1	0	0
25¢	= "100"	0	1	1	1	25¢	0	1	0
50¢	= "000"	1	0	0	0	50¢	1	0	0
50¢	= "001"	1	0	0	1	50¢	1	0	0
50¢	= "010"	1	0	1	0	Wait	0	0	1
50¢	= "011"	1	0	1	1	50¢	1	0	0

Next State Logic

The next state logic for this counter depends on both the current state variables and the system input Up. We can again take advantage of don't cares for the unused state code to minimize the logic.

		Q ₁ _cur	Q ₁ _nxt		
Q _{in}	D _{in}	00	01	11	10
0	0	0	0	X	1
0	1	0	0	X	1
1	0	0	0	X	1
1	X	0	1	X	0

		Q ₁ _cur	Q ₀ _nxt		
Q _{in}	D _{in}	00	01	11	10
0	0	0	1	X	0
0	1	0	1	X	0
1	0	0	1	X	0
1	X	1	0	X	0

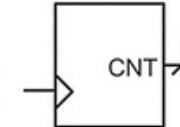
→ $Q_0_{nxt} = (Q_0_{cur} \cdot Q_{in}') + (Q_0_{cur} \cdot D_{in}) + (Q_1_{cur} \cdot Q_0_{cur}' \cdot Q_{in} \cdot D_{in}')$

→ $Q_1_{nxt} = (Q_1_{cur} \cdot Q_{in}') + (Q_1_{cur} \cdot D_{in}) + (Q_0_{cur} \cdot Q_{in} \cdot D_{in}')$

Example: 2-Bit Binary Up Counter (Part 1)

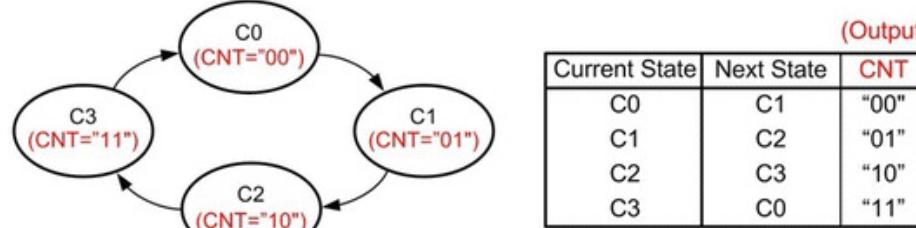
Word Description

We are going to design a 2-bit binary up counter. The counter will increment by 1 on every rising edge of the clock ("00", "01", "10", "11"). When the counter reaches "11", it will start over counting at "00". The output of the counter is called CNT.



State Diagram & State Transition Table

The state diagram for this counter is below. Notice that there are no inputs to the state machine. Also notice that the machine transitions in a linear pattern through the states and continually repeats the sequence of states. The outputs of this machine depend only on the current state so they are written inside of the state circles. This is a Moore machine.



State Encoding

When implementing this counter, we can use "state-encoded outputs". This means that we choose the state codes so that they match the desired output at each state. This allows the machine to simply use the current state variables for the system outputs. Let's name the current state variables Q_1_{cur} and Q_0_{cur} and the next state variables Q_1_{nxt} and Q_0_{nxt} . The state code assignments and updated state transition table are below.

State	Code	Current State		Next State		Outputs
		Q_1_{cur}	Q_0_{cur}	Q_1_{nxt}	Q_0_{nxt}	
C0	= "00"	0	0	C1	0	00
C1	= "01"	0	1	C2	1	01
C2	= "10"	1	0	C3	1	10
C3	= "11"	1	1	C0	0	11

Example: 2-Bit Binary Up Counter (Part 2)

Next State Logic

The next state logic for this counter only depends on the current state variables since there are no inputs to the system.

<u>Q_1_{nxt}</u>	
Q_1_{cur}	0 1
0	0 1
1	1 0

$$Q_1_{nxt} = (Q_1_{cur}' \cdot Q_0_{cur}) + (Q_1_{cur} \cdot Q_0_{cur}')$$

or

$$Q_1_{nxt} = Q_1_{cur} \oplus Q_0_{cur}$$

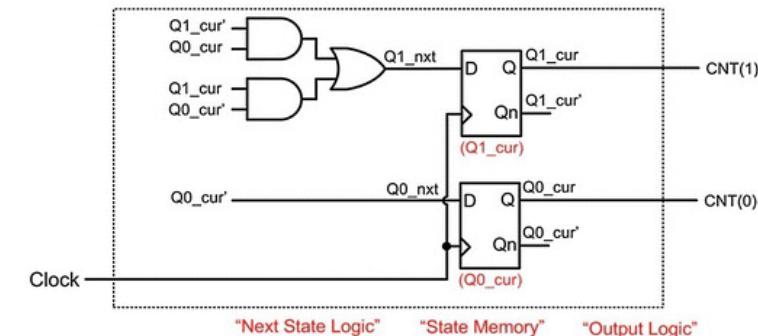
Output Logic

Since we are using state-encoded outputs, the outputs of the system will simply be the current state variables.

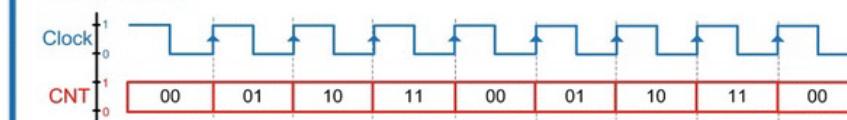
$$CNT(1) = Q_1_{cur}$$

$$CNT(0) = Q_0_{cur}$$

Logic Diagram



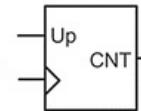
Timing Diagram



Example: 2-Bit Binary Up/Down Counter (Part 1)

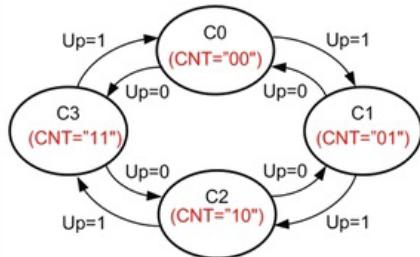
Word Description

We are going to design a 2-bit binary up/down counter. When the system input "Up" is asserted, the counter will increment by 1 on every rising edge of the clock. When Up=0, the counter will decrement by 1 on every rising edge of the clock. The output of the counter is called CNT.



State Diagram & State Transition Table

The state diagram for this counter is below. In this diagram, if the input Up=1, the machine will traverse the states in order to create an incrementing count. If the input Up=0, the machine will traverse the states in the opposite order. The outputs of this machine again only depend on the current state so they are written inside of the state circles. This is a Moore machine.



Current State	Up	(Input)		(Output)	
		Q1 _{cur}	Q0 _{cur}	Q1 _{nxt}	Q0 _{nxt}
C0	0	0	0	C3	"00"
	1	0	1	C1	
C1	0	0	1	C0	"01"
	1	0	0	C2	
C2	0	0	1	C1	"10"
	1	0	0	C3	
C3	0	0	1	C2	"11"
	1	0	0	C0	

State Encoding

Again, this counter will use "state-encoded outputs". Let's name the current state variables Q1_{cur} and Q0_{cur} and the next state variables Q1_{nxt} and Q0_{nxt}. The state code assignments and updated state transition table are below.

State	Code	Current State		Input	Next State		Outputs
		Q1 _{cur}	Q0 _{cur}		Q1 _{nxt}	Q0 _{nxt}	
C0	= "00"	0	0	0	C3	1	1 "00"
C0	= "00"	0	0	1	C1	0	1 "00"
C1	= "01"	0	1	0	C0	0	0 "01"
C1	= "01"	0	1	1	C2	1	0 "01"
C2	= "10"	1	0	0	C1	0	1 "10"
C2	= "10"	1	0	1	C3	1	1 "10"
C3	= "11"	1	1	0	C2	1	0 "11"
C3	= "11"	1	1	1	C0	0	0 "11"

Example: 2-Bit Binary Up/Down Counter (Part 2)

Next State Logic

The next state logic for this counter depends on both the current state variables and the input Up.

Q1 _{cur}		Q0 _{cur}		Up	Q1 _{nxt}
0	0	1	0	0	1
	1	0	1	1	0

$$Q1_{nxt} = Q1_{cur} \oplus Q0_{cur} \oplus Up$$

Q1 _{cur}		Q0 _{cur}		Up	Q0 _{nxt}
0	1	0	0	0	1
	1	1	0	1	0

$$Q0_{nxt} = Q0_{cur}'$$

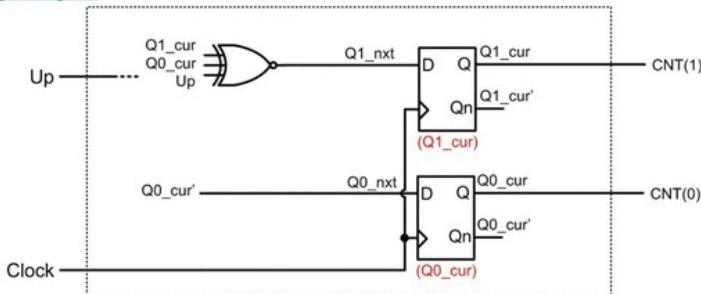
Output Logic

Since we are using state-encoded outputs, the outputs of the system will simply be the current state variables.

$$CNT(1) = Q1_{cur}$$

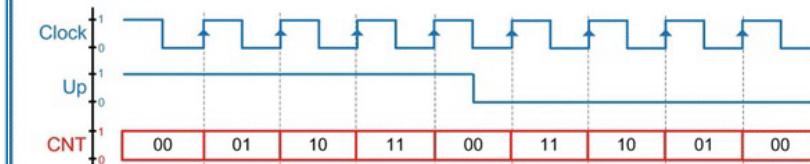
$$CNT(0) = Q0_{cur}$$

Logic Diagram



"Next State Logic" "State Memory" "Output Logic"

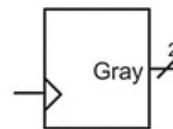
Timing Diagram



Example: 2-Bit Gray Code Up Counter (Part 1)

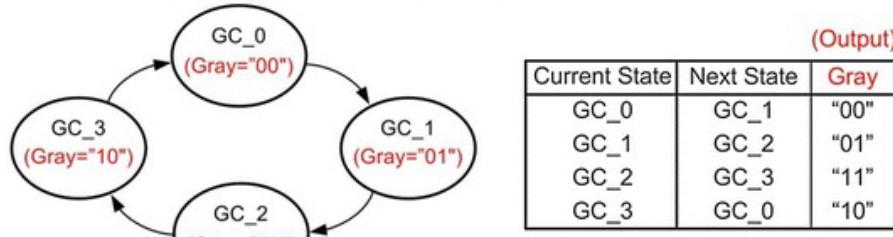
Word Description

We are going to design a 2-bit gray code up counter. The counter will output an incrementing gray code pattern on every rising edge of the clock ("00", "01", "11", "10"). When the counter reaches "11", it will start over counting at "00". The output of the counter is called Gray.



State Diagram & State Transition Table

The state diagram for this counter is below. Notice that there are no inputs to the state machine. Also notice that the machine transitions in a linear pattern through the states and continually repeats the sequence of states. The outputs of this machine depend only on the current state, so they are written inside of the state circles. This is a Moore machine.



State Encoding

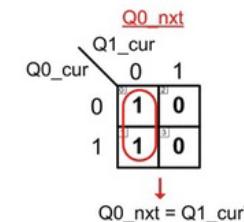
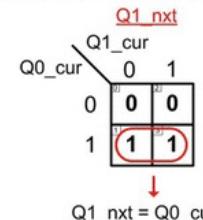
When implementing this counter, we can use "state-encoded outputs". This means that we choose the state codes so that they match the desired output at each state. This allows the machine to simply use the current state variables for the system outputs. Let's name the current state variables Q_1_{cur} and Q_0_{cur} and the next state variables Q_1_{nxt} and Q_0_{nxt} . The state code assignments and updated state transition table are below.

State	Code	Current State		Next State		Outputs	
		Q_1_{cur}	Q_0_{cur}	Q_1_{nxt}	Q_0_{nxt}		
GC_0	= "00"						
GC_1	= "01"	0	0	GC_1	0	1	"00"
GC_2	= "11"	0	1	GC_2	1	1	"01"
GC_3	= "10"	1	1	GC_3	1	0	"11"
		1	0	GC_0	0	0	"10"

Example: 2-Bit Gray Code Up Counter (Part 2)

Next State Logic

The next state logic for this counter only depends on the current state variables since there are no inputs to the system. Care must be taken when synthesizing the next state logic because the order of the current state variable values in the state transition table is not in a binary count order as in prior examples.

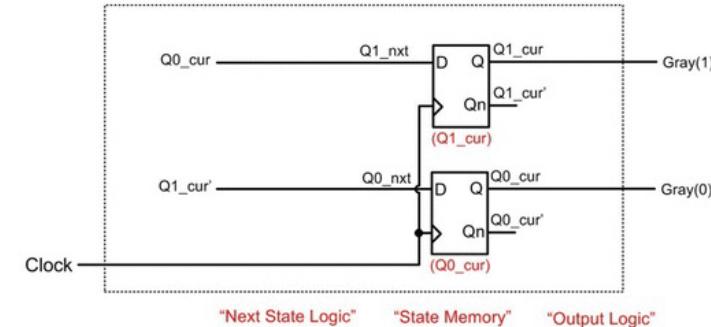


Output Logic

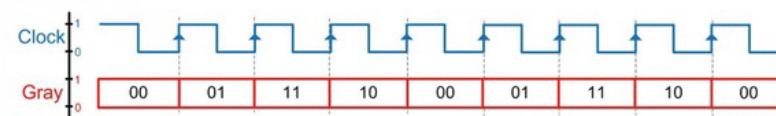
Since we are using state-encoded outputs, the outputs of the system will simply be the current state variables.

$$\begin{aligned} \text{Gray}(1) &= Q_1_{cur} \\ \text{Gray}(0) &= Q_0_{cur} \end{aligned}$$

Logic Diagram



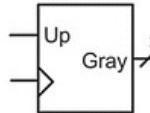
Timing Diagram



Example: 2-Bit Gray Code Up/Down Counter (Part 1)

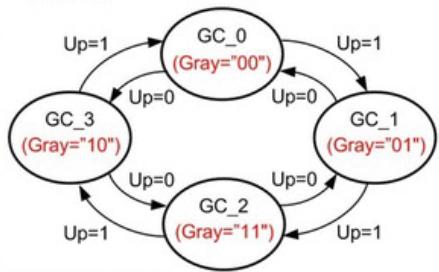
Word Description

We are going to design a 2-bit gray code up/down counter. When the system input "Up" is asserted, the counter will output an incrementing gray code pattern on every rising edge of the clock ("00", "01", "11", "10). When the input Up=0, the counter will output a decrementing gray code pattern. The output of the counter is called Gray.



State Diagram & State Transition Table

The state diagram for this counter is below. The outputs of this machine again only depend on the current state, so they are written inside of the state circles. This is a Moore machine.



	(Input)		(Output)	
Current State	Up	Next State	Gray	
GC_0	0	GC_3	"00"	
	1	GC_1		
GC_1	0	GC_0	"01"	
	1	GC_2		
GC_2	0	GC_1	"11"	
	1	GC_3		
GC_3	0	GC_2	"10"	
	1	GC_0		

State Encoding

Again, this counter will use "state-encoded outputs". Let's name the current state variables Q1_cur and Q0_cur and the next state variables Q1_nxt and Q0_nxt. The state code assignments and updated state transition table are below.

State	Code	Current State		Input	Next State		Outputs
		Q1_cur	Q0_cur		Up	Q1_nxt	
GC_0	"00"	0	0	0	GC_3	1	"00"
GC_0	"00"	0	0	1	GC_1	0	"00"
GC_1	"01"	0	1	0	GC_0	0	"01"
GC_1	"01"	0	1	1	GC_2	1	"01"
GC_2	"11"	1	1	0	GC_1	0	"11"
GC_2	"11"	1	1	1	GC_3	1	"11"
GC_3	"10"	1	0	0	GC_2	1	"10"
GC_3	"10"	1	0	1	GC_0	0	"10"

Example: 2-Bit Gray Code Up/Down Counter (Part 2)

Next State Logic

The next state logic for this counter depends on both the current state variables and the input Up. Again, care must be taken when synthesizing the next state logic due to the non-regular pattern of the current state codes in the state transition table.

Q1_nxt	
Q1_cur	Q0_cur
Up	00 01 11 10
0	1 0 0 1

$$Q1_{nxt} = (Q0_{cur}' \cdot Up') + (Q0_{cur} \cdot Up)$$

Q0_nxt	
Q1_cur	Q0_cur
Up	00 01 11 10
1	1 1 0 0

$$Q0_{nxt} = (Q1_{cur} \cdot Up) + (Q1_{cur}' \cdot Up')$$

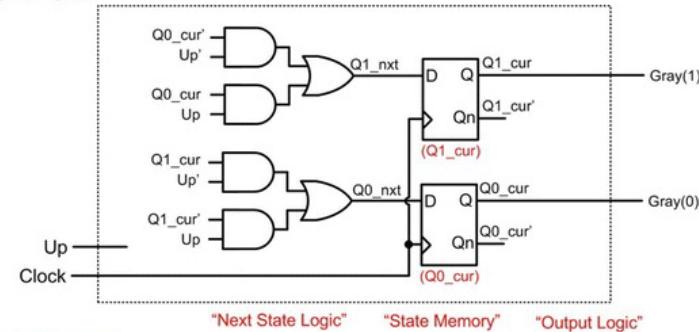
Output Logic

Since we are using state-encoded outputs, the outputs of the system will simply be the current state variables.

$$\text{Gray}(1) = Q1_{cur}$$

$$\text{Gray}(0) = Q0_{cur}$$

Logic Diagram



Timing Diagram

