**Sequential Multiplier**

**7D**

The given Verilog module is a seven-portion show (SSD) decoder with a multiplexing highlight. It takes two information signs, switch and in, and produces two result signs, fragments and anode\_active. A breakdown of the module's capabilities is as follows:

Inputs:

• toggle, with two bits: The seven-segment display's active anode is determined by this signal.

• four bits: The input value that will be displayed on the SSD is represented by this signal.

Outputs:

• segments (7 bits): This signal controls the segments of the SSD, indicating which segments should be turned on to display the desired input value.

• anode\_active (4 bits): This signal activates the appropriate anode of the SSD to display the output.

Behavior: The module has two always blocks that specify the behavior of the outputs based on the inputs.

1. The first always @ (in) block:

• This block triggers whenever the in signal changes.

• It uses a case statement to map the input value (in) to the corresponding segments that need to be turned on to display the desired value.

• Each case corresponds to a decimal digit (0 to 9) or a specific value (10).

• For example, when in is 0, the segments output is set to 7'b0000001, which activates segments a and b of the seven-segment display.

• The default case (when none of the specified values match) sets segments to 7'b11111111, turning off all the segments of the display.

2. The second always @ \* block:

• This block triggers whenever any of the input signals change (toggle in this case).

• It uses a case statement to determine which anode of the seven-segment display should be active based on the toggle value.

• Each case corresponds to a specific toggle value (0 to 3).

• For example, when toggle is 0, the anode\_active output is set to 4'b1110, activating the first anode of the SSD and deactivating the rest.

• There is no default case specified, so if the toggle value is outside the defined range, the anode\_active output will retain its previous value.

**Binary to BCD**

The given Verilog module addresses a parallel to-BCD (Twofold Coded Decimal) converter. It generates five BCD output signals, D0, D1, D2, D3, and D4, from a binary input of 16 bits (Binary). An explanation of the module's capabilities can be found here:

Inputs:

• 16-bit binary: This sign addresses the parallel worth to be changed over completely to BCD.

Outputs:

• D0, D1, D2, D3, and D4 (each five bits): The BCD equivalent of the input binary value is represented by these signals.

Internal indicators:

• Decimal: Twenty bits Throughout the conversion, this signal stores the intermediate BCD value.

Behavior: The module has one generally @(Binary) block, which triggers at whatever point the Paired information changes. Using a technique known as the double-dabble algorithm, it converts binary data into BCD.

Within the block that is always:

• At the beginning of each trigger, the Decimal signal is reset to zero (Decimal = 0).

• Each bit of the binary input is iterated over in reverse order using a for loop (i ranges from 0 to 15).

• The BCD conversion is carried out inside the loop by means of a series of conditional statements.

• Three bits are added to the least significant four bits of Decimal (Decimal[3:0]) when those four bits are greater than or equal to 5.

• The same procedure is followed for the remaining four groups of four bits (Decimal[11:8], Decimal[15:12], and Decimal[19:16]), and so on.

• This step guarantees that BCD digits more noteworthy than or equivalent to 5 are accurately changed by adding 3.

• In the end, the value in Decimal is changed by moving it one bit to the left and making the least important bit the current bit in Binary.

• As the loop progresses, the converted BCD digits in Decimal are effectively accumulated by this. After the consistently block, the BCD digits are alloted to their particular results utilizing appoint articulations:

• The four decimal bits D0 are the least important (Decimal[3:0]).

• D1 is alloted the following four pieces (Decimal[7:4]).

• The following four bits are assigned to D2 (Decimal[11:8]).

• D3 is allocated the following four pieces (Decimal[15:12]).

• D4 receives the four bits with the greatest significance (Decimal[19:16]).

In summary, this Verilog module uses the double-dabble algorithm to implement a binary-to-BCD converter. It switches a 16-bit double contribution over completely to five separate BCD yields (D0 to D4), permitting the portrayal of decimal digits in a BCD design.

**Counter**

A counter with configurable parameters is represented by the Verilog module provided. It counts from 0 to a predetermined value and returns to 0 upon assertion of a reset signal. An explanation of the module's capabilities can be found here:

Parameters:

• x (the default is: 4): The number of bits used to represent the counter value is determined by this parameter.

• n (default esteem: 10): Before resetting, this parameter determines the maximum count value.

Inputs:

• clk: This is the clock signal that sets off the counter on the positive edge.

• reset: This sign is utilized to reset the counter to 0 when declared.

• en: When the counter is active, it can increment thanks to this signal.

Outputs:

• count: With a width of x bits, this signal indicates the counter's current value.

Internal indicators:

• count: The counter's current value is stored in this register.

Behavior: The module has one generally @(posedge clk, posedge reset) block, which triggers on the positive edge of the clock sign or when the reset signal is declared.

Within the block that is always:

• In the event that the reset signal is 1, showing a reset demand, the count register is set to 0 (count <= 0).

• The counter can go up if the en signal is active (non-zero).

• Assuming that the ongoing worth of count is equivalent to n-1, demonstrating the most extreme count esteem has been reached, the count register is hindered to 0 to reset the counter (count <= 0).

• In any case, in the event that the greatest count esteem has not been reached, the count register is augmented by 1 (count <= count + 1).

Based on the clock and input signals, the module continuously updates the count register, allowing the counter to increment or reset in accordance with the defined conditions.

In rundown, this Verilog module executes a counter with configurable boundaries. It counts from 0 to a predefined esteem (n) and resets back to 0 when a reset signal is stated. The counter additions on the positive edge of the clock signal when empowered (en signal). The ongoing include esteem is put away in the count register, which is x pieces wide.

**Debouncer**

A debouncer circuit is represented by the Verilog module provided. It produces a debounced output signal (out) from an input signal (in). The debouncer is intended to eliminate clamor or errors from the info signal and give a steady result. An explanation of the module's capabilities can be found here:

Inputs:

• clk: The clock signal that starts the debouncer on the positive edge is this one.

• rst: This sign is utilized to reset the debouncer to its underlying state when affirmed.

• in: This is the information sign to be debounced.

Outputs:

• out: This sign addresses the debounced yield.

Inside signals:

• q1, q2, q3: During the debouncing process, the intermediate values are stored in these registers.

Behavior: The module has one always @(posedge clk, posedge rst) block that fires when the reset signal is asserted or the positive edge of the clock signal.

Within the block that is always:

• In the event that the rst signal is stated (high), demonstrating a reset demand, every one of the three registers (q1, q2, q3) are set to 0, successfully resetting the debouncer.

• The debouncing process begins in the event that the initial signal is not asserted (low):

• q1 is given the value of the input signal (in).

• The input signal is effectively delayed by one clock cycle when the value of q1 is assigned to q2.

• The worth of q2 is appointed to q3, further deferring the info signal by two clock cycles.

An assign statement is used to assign the out signal based on the debounced values following the always block:

• The out signal is set to 0 if the first signal is asserted.

• On the off chance that the rst signal isn't attested, the out signal is alloted the legitimate AND (and) activity of q1, q2, and q3.

• This indicates that a stable and debounced input signal indicates that the out signal will be high only if all three registers have high values simultaneously.

In a nutshell, this Verilog module uses a debouncer circuit to get rid of glitches or noise in an input signal. It compares the values of three registers to provide a stable output signal after delaying the input signal. The debouncing process takes place on the clock signal's positive edge, and the debouncer can be reset to its initial state with a reset signal.

**SM**

An unsigned multiplier is represented by the Verilog module provided. It takes a 8-digit multiplier (Multiplier) and multiplicand (Multiplicand) as sources of info and produces a 16-bit result (resultant) of the increase activity. A done signal is also provided by the module to let you know when the multiplication operation is finished. An explanation of the module's capabilities can be found here:

Inputs:

• 8-bit multiplier: This sign addresses the multiplier for the increase activity.

• Multiplicand (8 pieces): The multiplicand for the multiplication operation is represented by this signal.

• load: When new values should be loaded into the multiplier, multiplicand, and accumulator, this signal tells you when to do so.

Outputs:

• done: When the multiplication operation is finished, this signal indicates it.

• the final (16 bits): The result of the multiplication operation is represented by this signal.

Inside signals:

• M1 (16 bits): This register holds the drawn out variant of the multiplier esteem.

• M2 (8 bits): The value of the multiplicand is in this register.

• acc (sixteen bits): The cumulative result of the multiplication operation is stored in this register.

Behavior: The module has one always @(posedge clk or posedge reset) block that activates when the reset signal is asserted or the positive edge of the clock signal.

Within the block that is always:

• All registers (M1, M2, acc) are reset to zero, effectively resetting the module, when the reset signal is asserted (high), which indicates a request for a reset.

• The multiplication operation proceeds even though the reset signal is not asserted (low):

• The M1 register receives the extended Multiplier value (with eight leading zeros), the M2 register receives the Multiplicand value, and the acc register is reset to zero if the load signal is asserted, indicating that new values should be loaded.

• The multiplication operation continues even if the M2 register is not zero and the load signal is not asserted:

• If the most un-critical piece of M2 (M2[0]) is 1, the acc register is refreshed by adding the worth of M1 to it.

• The M1 register is shifted to the left by one, and the M2 register is shifted to the right by one.

• The acc register remains unchanged and the value of acc is assigned to the resultant register if the load signal is not asserted and the M2 register is zero, indicating that the multiplication operation has been completed.

Moreover, there is another consistently @ (\*) block, which triggers at whatever point any of its bits of feedbacks change.

Within the block with always @ (\*):

• On the off chance that the heap or reset signals are stated, showing a heap or reset activity, the done flag is set to 0.

• The done signal is set to 1 if the M2 register is zero, indicating that the multiplication operation is finished.

In conclusion, an unsigned multiplier is implemented by this Verilog module. It performs the multiplication operation by accumulating the result in the accumulator while iteratively shifting the multiplier and multiplicand. When the multiplication is finished, the done signal indicates it, and the resultant signal stores the result.

**SS driver**

The given Verilog module addresses a SSD (Seven-Section Show) driver with extra usefulness for showing different BCD (Parallel Coded Decimal) values in view of a switch empower signal and a sign. An explanation of the module's capabilities can be found here:

Inputs:

• sign: This sign addresses the sign worth, which is utilized to decide the showed esteem on the SSD.

• clk: This is the internal operations clock signal.

• rst: The module is reset to its initial state with this signal.

BCD1, BCD2, and BCD0: The SSD's BCD values are represented by these signals.

Outputs:

• portions (7 pieces): This sign addresses the fragments that should be gone on to show the BCD esteem on the SSD.

• en: four bits: The SSD's active digit is determined by this signal.

Inside signals:

• flashimgClock: This is a wire signal utilized as the clock input for a clock divider.

• toggleEnable (2 pieces): Based on the toggling functionality, this signal decides which BCD value should be displayed on the SSD.

• in1: four bits The value that will be displayed on the SSD based on the sign value and the selected BCD value are stored in this register.

Modules that were created:

• divider\_clock: This is an occasion of a clock divider module (clock\_divider) that separates the info clock recurrence to create a more slow clock signal (flashingClock).

• Counter: The toggling feature is made possible by this instance of a counter module (Counter) that counts from 0 to 3.

• SD: This is an occasion of a seven-section decoder module (SD) that translates the BCD values into portions to show on the SSD.

Behavior: One always @(\*) block in the module fires whenever one of its inputs changes.

Inside the consistently @(\*) block:

• In light of the worth of toggleEnable, the in1 register is alloted different BCD values:

• BCD0 is assigned to in1 if toggleEnable is set to 0.

• BCD1 is assigned to in1 if toggleEnable is set to 1.

• BCD2 is assigned to in1 if toggleEnable is set to 2.

• Assuming toggleEnable is 3, in1 is allocated either 4'b1010 (10 in BCD) or 4'b1011 (11 in BCD) in light of the worth of the sign.

• Using the SD module, the decoded segments that correspond to the value of in1 are assigned to the segments that are output.

• The value toggleEnable is assigned to the en output, indicating which SSD digit should be active.

The module likewise incorporates the launch of a clock divider module (clock\_divider) and a counter module (Counter) to give the flipping usefulness to showing different BCD values on the SSD.

In summary, this Verilog module is a driver for a Seven-Segment Display (SSD) with additional functionality to display different BCD values based on a toggle enable signal and a sign signal. It uses a clock divider and a counter to control the toggling functionality, and a Seven-Segment Decoder (SD) module to decode the BCD values into segments for display on the SSD.

**Shift register**

The given Verilog module addresses a shift register with a selectable information. The three outputs of the shift register—S0, S1, and S2—store the values of the input signals (D0, D1, D2, D3, and D4) in accordance with the input that has been selected. The shift register can be controlled utilizing clock (clk) and reset (rst) signals, as well as compose control signals (w1 and w2). An explanation of the module's capabilities can be found here:

Inputs:

• D0, D1, D2, D3, and D4 The data that needs to be entered into the shift register is represented by these signals.

• clk: The shift register's clock signal is represented by this.

• rst: The shift register is reset to its initial state with this signal.

• w1, w2: The shift register's input is selected by means of these write control signals.

Outputs:

• S0, S1, S2 (4 pieces): The shift register's output is represented by these signals. The clock signal is used to update the shift register's values, which are determined by the selected input.

Internal indicators:

• sel (2 pieces): This register holds the ongoing determination an incentive for the contribution of the shift register.

Behavior: The module has two generally impedes.

Within the first block with always @ (\*):

• Any time one of the block's inputs changes, the block is triggered.

• The sel register's value and the input signals D0, D1, D2, D3, and D4 are used to assign the output signals S0, S1, and S2.

• On the off chance that sel is 2'b00 or rst is stated (high), showing a reset condition, the result signals are relegated the upsides of D0, D1, and D2.

• In the event that sel is 2'b01, the result signals are alloted the upsides of D1, D2, and D3.

• The values of D2, D3, and D4 are assigned to the output signals if sel is 2'b10 or 2'b11.

Within the second block with always @ (posedge clk, posedge rst):

• The block is set off on the positive edge of the clock sign or when the reset signal is affirmed.

• The sel register is set to 0 if the reset signal (rst) is asserted (high), which indicates a request for a reset.

• The shift register is updated based on the values of the write control signals (w1 and w2) and the current value of sel if the reset signal is not asserted.

• The sel register does not change if both w1 and w2 are asserted (high), which indicates that the selection has not changed.

• The sel register does not change if w1 and w2 are not asserted (low), which again indicates that the selection has not changed.

• The sel register is increased by one if w1 is not asserted and w2 is asserted and the current value of sel is less than 2.

• The sel register is decremented by 1 if w1 is asserted and w2 is not, and the current value of sel is greater than 0.

**Sign calculator**

A sign calculator for two input numbers is represented by the Verilog module that is provided. It determines the sign of the result and calculates the absolute values of the input numbers (num1 and num2). An explanation of the module's capabilities can be found here:

Inputs:

• Numbers 1 and 2 (eight bits): The numbers that will be used to calculate the sign and absolute values are represented by these signals.

Outputs:

• one-bit sign: The sign of the result is represented by this signal. If the numbers have different signs (one is positive, and the other is negative), it will be 1 and 0 respectively. If the numbers have the same sign, it will be 0.

• num1\_abs and num2\_abs (8 pieces): The absolute values of num1 and num2 are respectively represented by these signals.

Behavior: The module utilizes the allocate explanation to relegate values to its result signals.

• The result of the XOR operation between the input numbers' most significant bits (num1[7] and num2[7]) is assigned to the sign output. The results of this XOR operation are used to see if the numbers have different signs. The sign output will be 1 if they do; If not, it will be 0.

• A conditional expression is used to assign the absolute value of num1 to the num1\_abs output. In the event that num1[7] (the sign piece) is 0 (showing a positive number), num1\_abs will be relegated the worth of num1 itself. Otherwise, the bitwise complement operator is used to calculate the two's complement of num1 and add 1, and the result is assigned to num1\_abs if num1[7] is 1 (indicating a negative number). The negative number is effectively transformed into the positive number.

• A conditional expression is used to assign the absolute value of num2 to the num2\_abs output in a similar manner. num2\_abs will be assigned num2 if num2[7] is 0; Otherwise, the two's complement of num2 is calculated and assigned to num2\_abs if num2[7] is 1.

In a nutshell, this Verilog module determines the absolute values and the sign of two input numbers. The positive equivalents of the input numbers are provided by num1\_abs and num2\_abs, while the sign output indicates whether the input numbers have distinct signs.

**Synchronizer**

The given Verilog module addresses a synchronizer circuit, which is utilized to synchronize an offbeat sign (sig) to a clock space (clk). An explanation of the module's capabilities can be found here:

Inputs:

• Clock signal, or clk: The signal sig needs to be synchronized with this, which is the clock signal of the clock domain.

• sig (offbeat sign): This is the information signal that should be synchronized to the clock area.

• First (signal to reset): The synchronizer circuit is reset with this synchronous reset signal.

Outputs:

• The synchronized signal sig1: In the clock domain of clk, this is the output signal that represents the synchronized version of the input signal, sig.

Variables inside:

• Register with META: The input signal sig is captured in sync with the clock by this internal register.

Behavior: The module makes use of an always block that is sensitive to the positive edges of clk and rst. Inside the consistently block, the way of behaving is characterized in light of the clock and reset conditions.

• The synchronizer circuit is reset when the reset signal rst is asserted (a positive edge). Assuring a known initial state, the META register and the sig1 output are both set to 0 (1'b0).

• The input signal sig is synchronized with the clock domain when the reset signal is not asserted, indicating that the circuit is not in a reset state. The worth of sig is caught in the META register on the positive edge of clk, guaranteeing appropriate synchronization. The output signal sig1, which represents the synchronized version of sig, is then given the synchronized value that is stored in META.

In conclusion, this Verilog module implements a fundamental synchronizer circuit that synchronizes an asynchronous input signal, sig, with a clock domain that is specified by clk. The synchronized result signal sig1 addresses the synchronized rendition of sig in the clock space of clk. To ensure a known initial state for the synchronizer circuit, the module also includes synchronous reset functionality.

**Clock divider**

A clock divider circuit that divides the frequency of an input clock signal (clk) to generate an output clock signal (clk\_out) is represented by the given Verilog module. The parameter n determines the frequency division factor. The module's functionality is described as follows:

Inputs:

• Clock signal, or clk: The input clock signal that requires division is this one.

• First (signal to reset): This is a nonconcurrent reset signal used to reset the clock divider circuit.

Outputs:

• The divided clock signal clk\_out: This is the result clock signal that addresses the separated adaptation of the information clock clk.

Parameters:

• n: This boundary determines the recurrence division factor. It figures out how many clock cycles are necessary to produce one cycle of the divided clock signal.

Variables inside:

• count (register with 32 bits): This register monitors the ongoing count an incentive for recurrence division.

Behavior: The module incorporates two generally impedes, each set off on the positive edge of clk and the positive edge of rst.

• The behavior is as follows in the first always block:

• The count register is set to 32'b0 to guarantee a known initial state for frequency division if the reset signal rst is asserted, indicating an asynchronous reset.

• Assuming the count esteem arrives at n-1 (showing the ideal recurrence division factor), the count register is reset to 32'b0 to begin the following pattern of recurrence division.

• If not, the count register is incremented by one in all other cases (count = count + 1), indicating the progression of clock cycles.

• The behavior is as follows in the second always block:

• The output clock clk\_out is set to 0 (clk\_out = 0) when the reset signal rst is asserted, ensuring a known initial state for the output clock signal.

• The output clock clk\_out is toggled by complementing its current value (clk\_out = clk\_out) when the count value reaches n-1, which indicates the desired frequency division factor. This really isolates the recurrence of the information clock clk by the variable n and produces the partitioned clock signal clk\_out.

To summarize, this Verilog module uses a counter (count) to keep track of the clock cycles and toggles the output clock signal (clk\_out) at the desired frequency division points to implement a clock divider circuit that divides the frequency of an input clock signal by the factor specified by the parameter n. To ensure a known initial state for the counter and output clock signal, the module also includes asynchronous reset functionality.

**Push button detector**

The given Verilog module addresses a pushbutton identifier circuit. Using a variety of sub-modules, it finds the rising edge of a pushbutton signal (w). An explanation of the module's capabilities can be found here:

Inputs:

• Clock signal, or clk: The pushbutton detection circuit uses this as a reference for the input clock signal.

• First (signal to reset): The circuit is reset with this asynchronous reset signal.

• w, or the push-button signal: This is the pushbutton's input signal that needs to be found.

Outputs:

• z, or the output signal: This is the result signal showing the identification of a rising edge on the pushbutton signal.

Inner Wires:

• clk\_out (wire): The clock\_divider module's output clock signal is represented by this wire.

• The wires t1, t2: The pushbutton signal is synchronized and debounced by means of these wires, which are intermediate signals.

Sub-Modules:

• divider\_clock: The clock\_divider module, which produces a divided version of the input clock signal, is demonstrated here.

• Declarator: The Debouncer module, which synchronizes and debounces the pushbutton signal, is demonstrated here.

• Synculator: The Synchronizer module further synchronizes the debounced pushbutton signal in this instance.

• The rising-edge-detector: The rising edge of the synchronized pushbutton signal is detected by this instance of the rising\_edge\_detector module.

Behavior:

1. The clock\_divider module is launched with the info clk and rst, and it creates the partitioned clock signal clk\_out.

2. The Debouncer module is launched with clk\_out, rst, w, and t1. Using the divided clock signal as a reference, it synchronizes and debounces the pushbutton signal w and outputs the synchronized and debounced signal on t1.

3. The Synchronizer module is started up with clk\_out, rst, t1, and t2. It further synchronizes the debounced signal t1 utilizing the partitioned clock signal as a kind of perspective, and results the synchronized sign on t2.

4. The rising\_edge\_detector module is instantiated with clk\_out, rst, t2, and z. It uses the divided clock signal as a reference to find the rising edge of the synchronized signal t2 and outputs the result on z. The Verilog module is used to create a pushbutton detection circuit. It reliably detects and responds to rising edges on the pushbutton signal by utilizing techniques such as clock division, synchronization, debouncing, and rising edge detection. The detection of a rising edge in the pushbutton signal is indicated by the output signal z.

**Rising edge detector**

A rising edge detector circuit is represented by the given Verilog module. Using a finite state machine, it finds the rising edge of an input signal (w). An explanation of the module's capabilities can be found here:

Inputs:

• clock (signal for the clock): This is the information clock signal utilized as a source of perspective for the rising edge discovery circuit.

• reset (reset signal): The circuit is reset with this asynchronous reset signal.

• w (signal for input): The rising edge needs to be identified for this input signal.

Outputs:

• z, or the output signal: The output signal that indicates the presence of a rising edge in the input signal is this.

Signals from within:

• state (register of two bits): This register addresses the present status of the limited state machine.

Constants:

• A, B, and C (two bits): These boundaries address the potential conditions of the limited state machine.

Behavior:

1. The always block is triggered when either the clock signal's positive edge or the reset signal's positive edge are reached.

2. The circuit is asynchronously reset to the initial state A if the reset signal is active (high). The circuit operates in accordance with the current state if the reset signal is inactive.

4. The case proclamation actually takes a look at the present status of the circuit (state) and decides the following state in light of the information signal (w).

5. In express A, in the event that w is high (showing the info signal is 1), the circuit changes to state B.

6. The circuit returns to state A in state B if w becomes low, indicating a falling edge; however, if w remains high, indicating that the input signal is still 1, the circuit moves to state C. The circuit returns to state A if w becomes low in state C. If the current state is B, the output signal z receives the value 1, indicating a rising edge detection.

In conclusion, a rising edge detector is implemented by means of a finite state machine in the Verilog module. It provides the detection result on the output signal z after it has identified the rising edge of the input signal w. The rising edge detection takes place when the input signal moves from low to high.