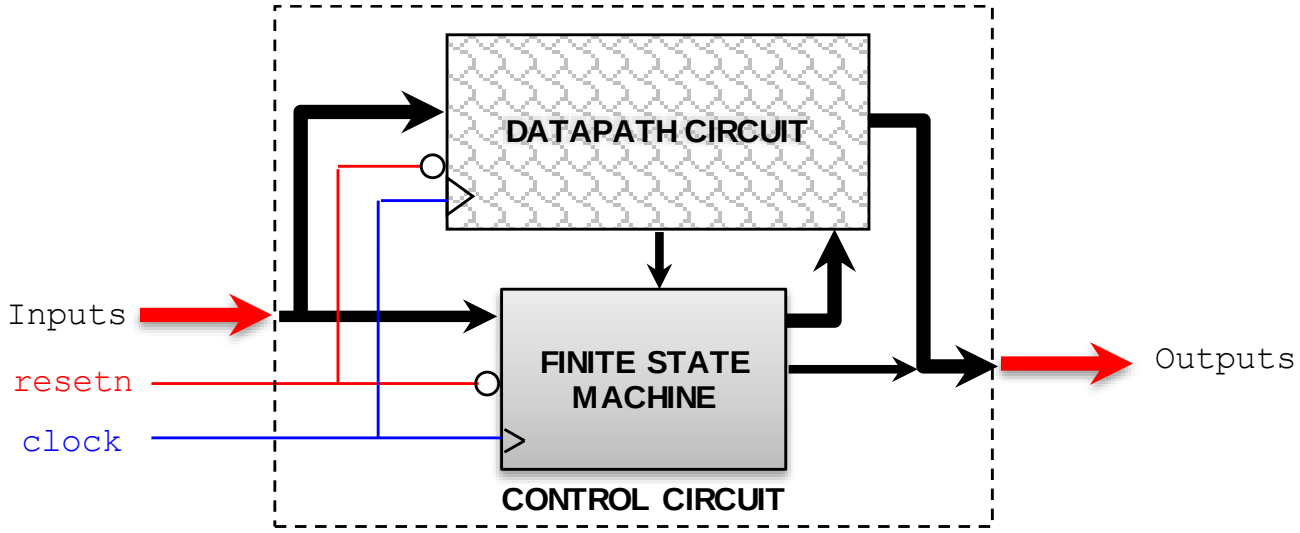
Unit 7 – Introduction to Digital System Design

Digital System Model

- FSM + Datapath Circuit:

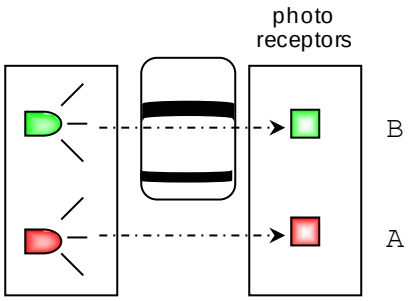


Use of Generic Components

- Among the most common components used in the development of digital systems we have registers, shift registers, and counters. To optimize design time, it is recommended to use parameterized components (VHDL for FPGA Tutorial – Unit 5):
  - ✓ n-bit register with enable and synchronous clear: my_rege
  - ✓ Counter modulo-N with enable and synchronous clear: my_genpulse_sclr
  - ✓ n-bit parallel access (right/left) register with enable and synchronous clear: my_pashiftreg_sclr

Examples

Example: Car Lot Counter



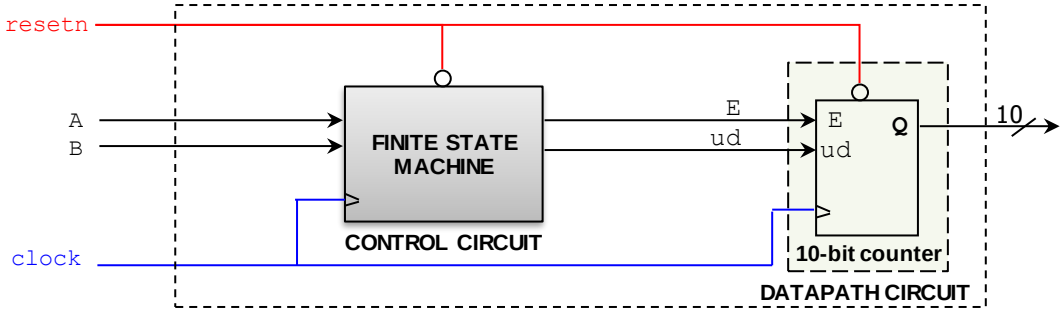
If A = 1 → No light received (car obstructing LED A)
If B = 1 → No light received (car obstructing LED B)

If car enters the lot, the following sequence (A|B) must be followed:
00 → 10 → 11 → 01 → 00
If car leaves the lot, the following sequence (A|B) must be followed:
00 → 01 → 11 → 10 → 00

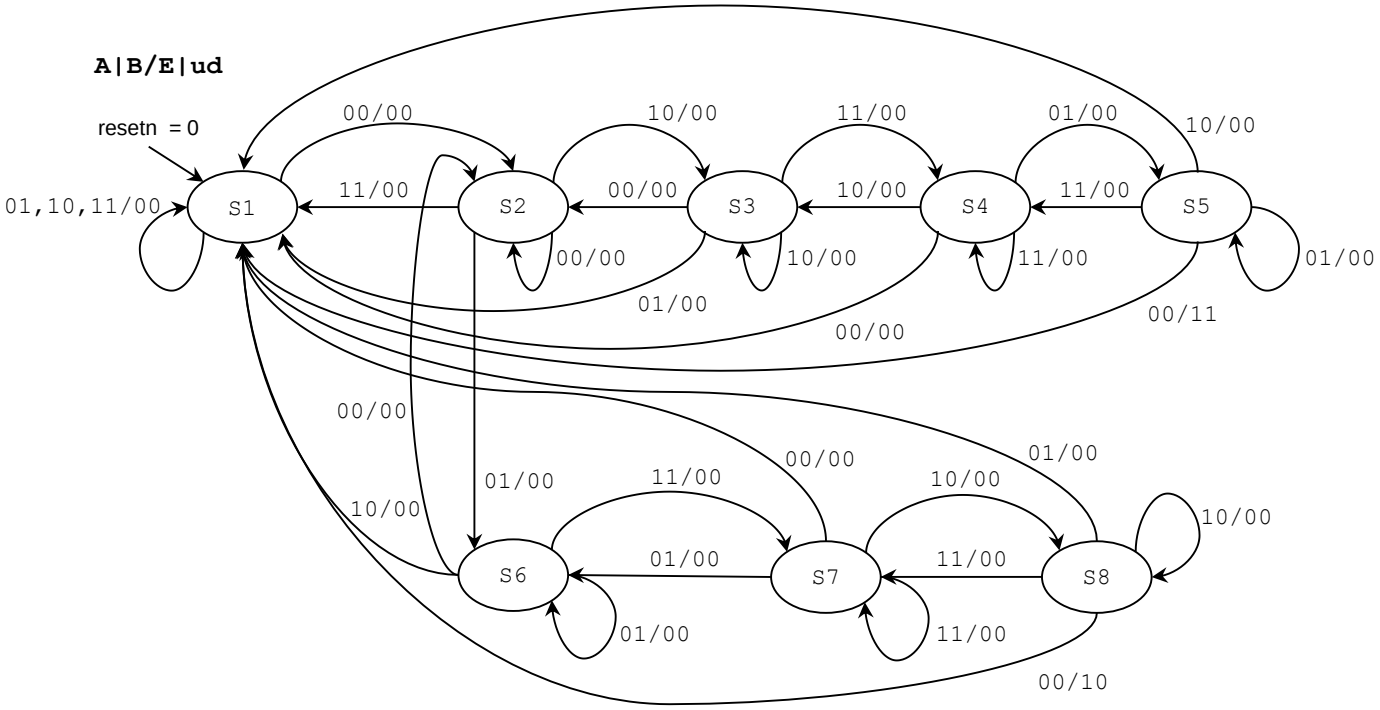
A car might stay in a state for many cycles since the car speed is very large compared to that of the clock frequency.

DIGITAL SYSTEM (FSM + Datapath circuit)

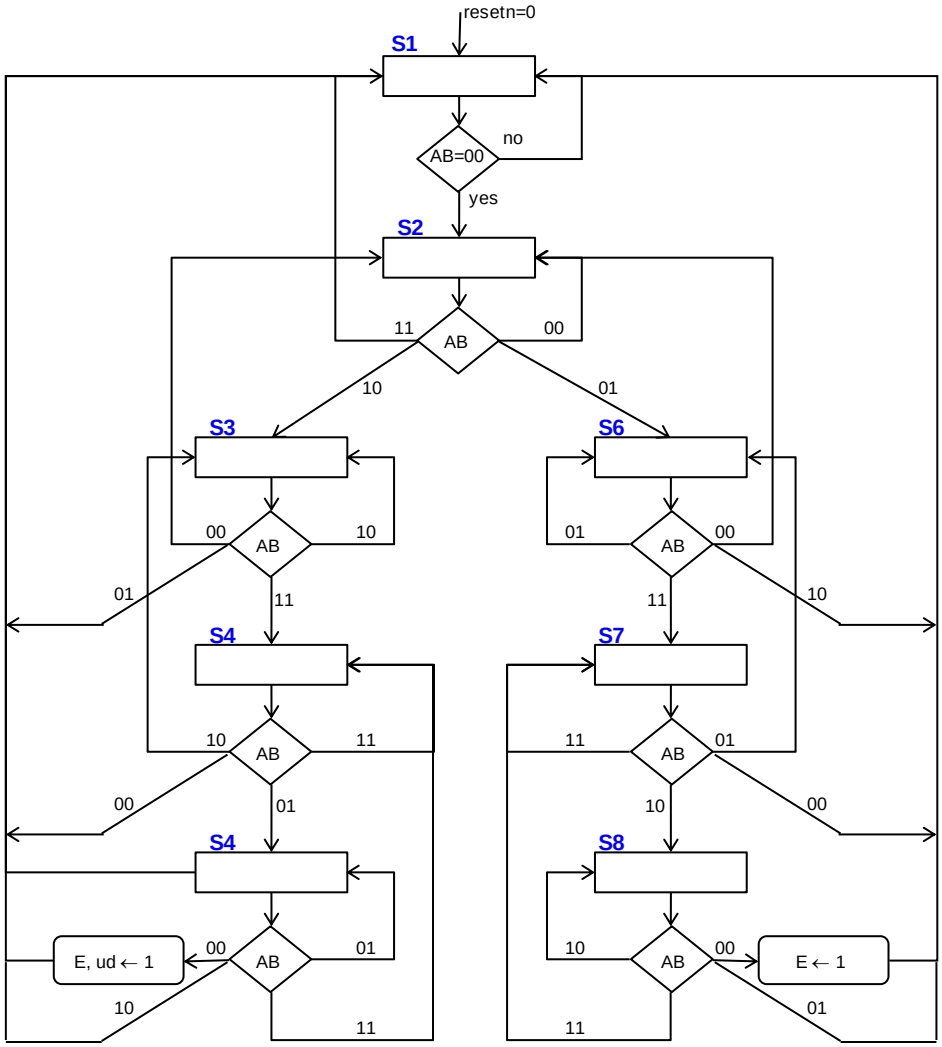
- Usually, when 'resetn' (asynchronous clear) and 'clock' are not drawn, they are implied.



- **Finite State Machine (FSM)**:



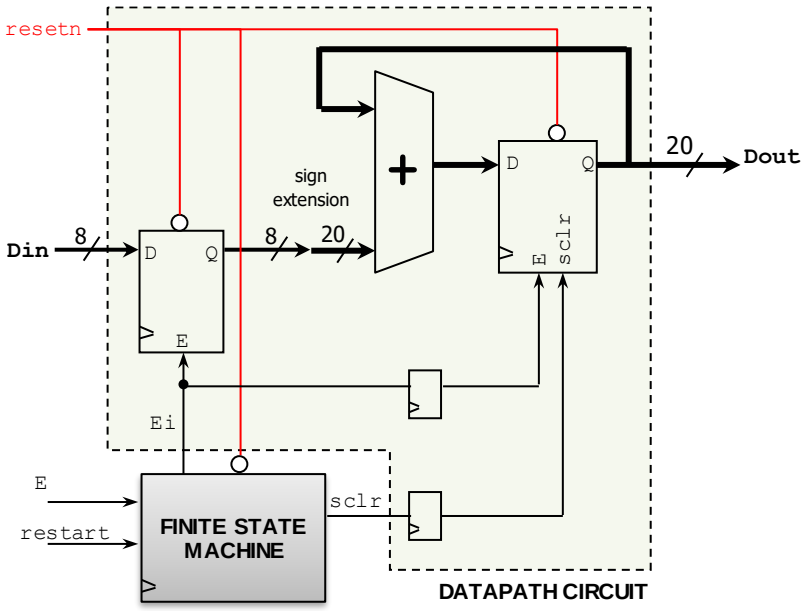
- **Algorithmic State Machine (ASM) chart:**



EXAMPLE: ACCUMULATOR

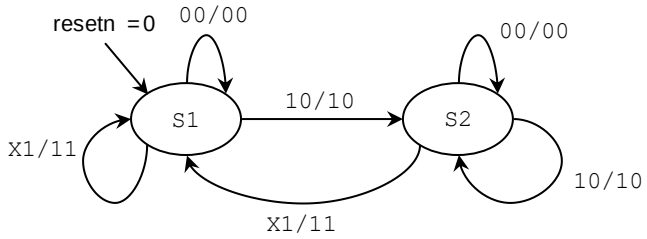
DIGITAL SYSTEM (FSM + Datapath circuit)

- Register: `sclr`: Synchronous clear. If E = '1' and sclr = '1', then the output bits of the registers are set to zero.

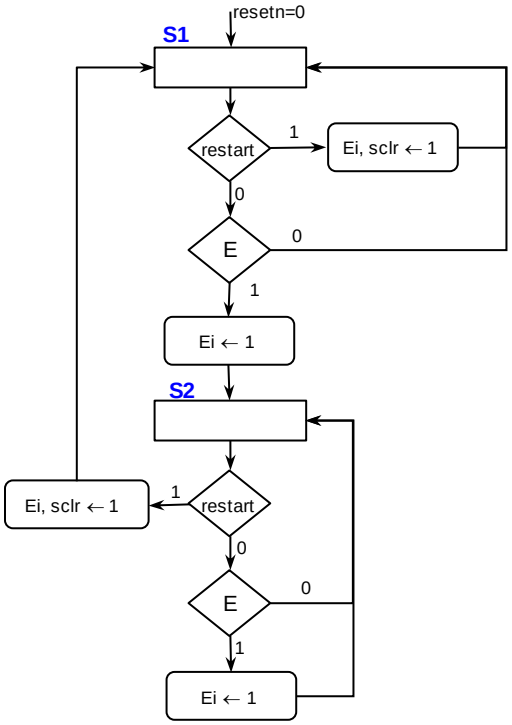


- **Finite State Machine (FSM)**:

E|restart/Ei|sclr



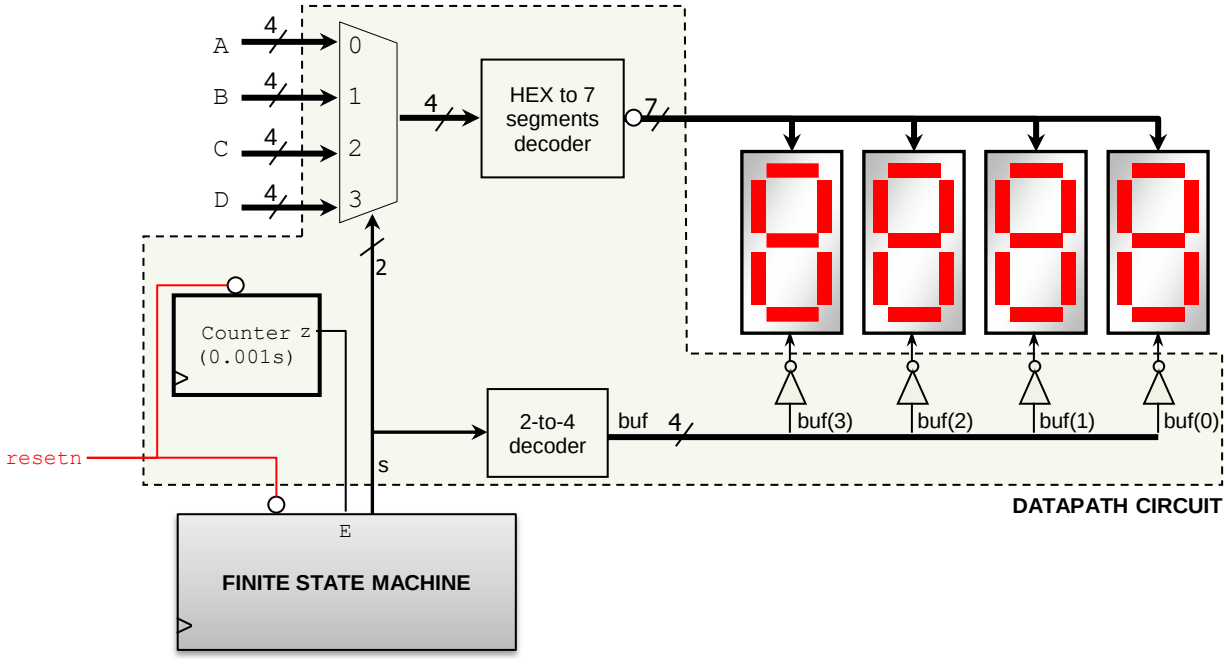
- **Algorithmic State Machine (ASM) Chart**:



EXAMPLE: 7-SEGMENT SERIALIZER (VHDL CODE)

DIGITAL SYSTEM (FSM + Datapath circuit)

- Most FPGA Development board have a number of 7-segment displays (e.g., 4, 8). However, only one can be used at a time.
- If we want to display four digits (inputs A, B, C, D), we can design a serializer that will only show one digit at a time on the 7-segment displays.
- Since only one 7-segment display can be used at a time, we need to serialize the four HEX (or BCD) outputs. In order for each digit to appear bright and continuously illuminated, each digit is illuminated for 1 ms every 4 ms (i.e. a digit is un-illuminated for 3 ms and illuminated for 1 ms). This is taken care of by feeding the output z of the 'counter to 0.001s' to the enable input of the FSM. This way, state transitions only occur each 0.001 s.
- Note: the input signals as well as the enable signals to the four 7-segment displays are active low (this is the proper configuration for the Nexys A7-50T/A7-100T, Nexys 4-DDR).



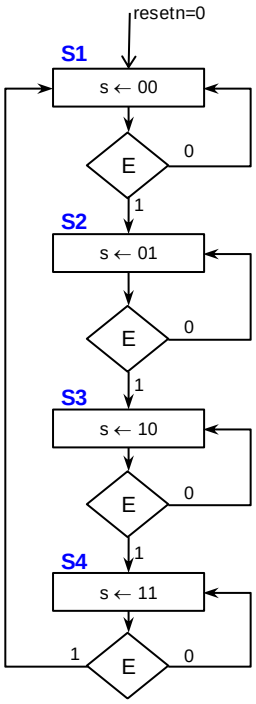
- Generic Component: Behavior on the clock tick.

0.001 s counter (modulo-10^5): Free running counter

```
if Q = 10^5 - 1 then
      Q ← 0
   else
      Q ← Q+1
  end if;
end if;

* z = 1 if Q = 10^5-1
```

- **Algorithmic State Machine (ASM) chart:** This is a Moore-type FSM.



EXAMPLE: BIT-COUNTING CIRCUIT (Parametric VHDL Code) (Video: VHDL coding in Vivado, n=8)

- It counts the number of bits in a register A that have the value '1'.
- Example: A=10110011 → count = 0101.

SEQUENTIAL ALGORITHM

```
C ← 0
while A ≠ 0
   if a0 = 1 then
      C ← C + 1
   end if
   right shift A
end while
```

DIGITAL SYSTEM (FSM + Datapath circuit)

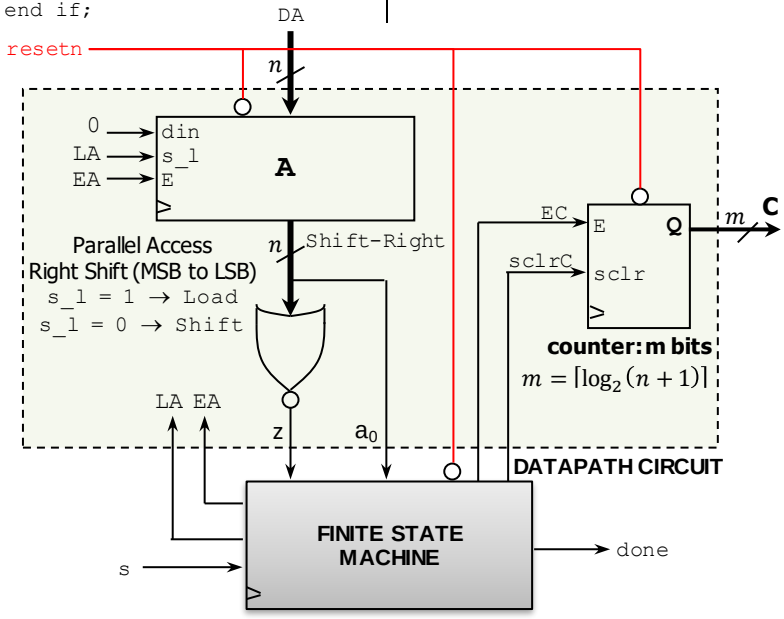
- FSM: as a ASM Chart. Modulo-n+1 bit counter (m bits): it has synchronous clear ($sclr$).
- Generic components: Behavior on the clock tick:

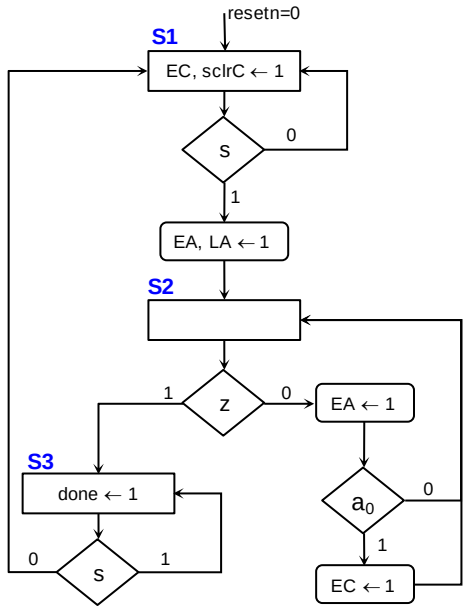
Counter modulo-n+1 (0 to n):
If E=0, the count stays.

```
if E = 1 then
   if sclr = 1 then
      Q ← 0
   elseif Q = n then
      Q ← 0
   else
      Q ← Q+1
  end if;
end if;
```

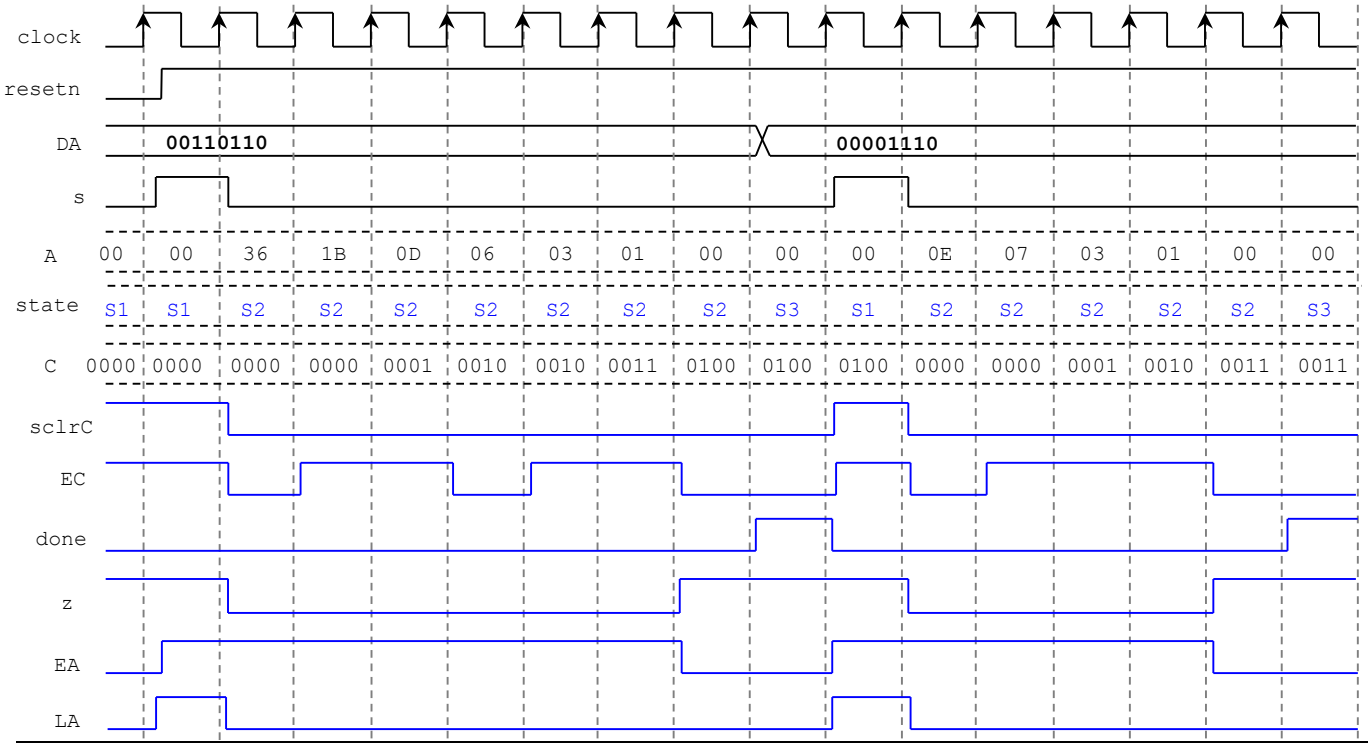
n-bit Parallel access shift register:
If E=0, the output is kept

```
if E = 1 then
   if s_l = '1' then
      Q ← D
   else
      Q ← shift in 'din' (to the right)
  end if;
end if;
```





- **Example** (timing diagram): $n = 8, m = 4$. Video: Timing diagram completion (different DA values)



EXAMPLE: SEQUENTIAL MULTIPLIER (Parametric VHDL Code) (Video: VHDL coding in Vivado, n=4)

UNSIGNED MULTIPLICATION: SEQUENTIAL ALGORITHM

```
P ← 0, Load A,B
while B ≠ 0
   if b0 = 1 then
      P ← P + A
   end if
   left shift A
   right shift B
end while
```

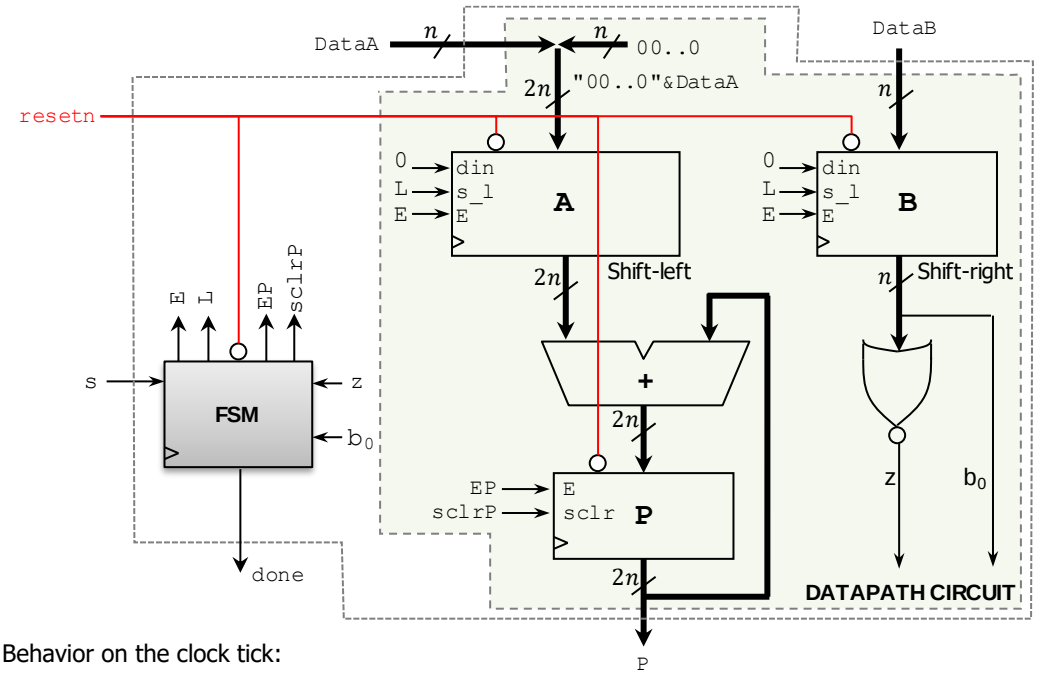
Example:

```
          1 1 1 1 x
          1 1 0 1
          ---------
          1 1 1 1   ⟶ P ← 0 + 1111
        0 0 0 0     ⟶ P ← 1111
      1 1 1 1       ⟶ P ← 1111 + 111100 = 1001011
    1 1 1 1         ⟶ P ← 1001011 + 1111000 = 11000011
  -----------------
  1 1 0 0 0 0 1 1
```

```
P ← 0, A ← 1111, B ← 1101
b0=1 ⇒ P ← P + A = 1111.          A ← 11110, B ← 110
b0=0 ⇒ P ← P = 1111.              A ← 111100, B ← 11
b0=1 ⇒ P ← P + A = 1111 + 111100 = 1001011.       A ← 1111000, B ← 1
b0=1 ⇒ P ← P + A = 1001011 + 1111000 = 11000011.  A ← 11110000, B ← 0
```

DIGITAL SYSTEM (FSM + Datapath circuit)

- Iterative Multiplier Architecture. Register P: $sclr$: synchronous clear. The result is computed in at most $n + 1$ cycles.



- Generic Components: Behavior on the clock tick:

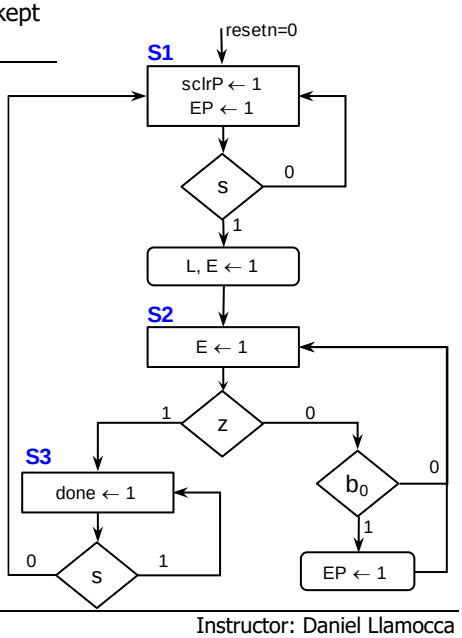
$2n$-bit register: If E=0, the output is kept

```
if E = 1 then
   if sclr = 1 then
      Q ← 0
   else
      Q ← D
  end if;
end if;
```

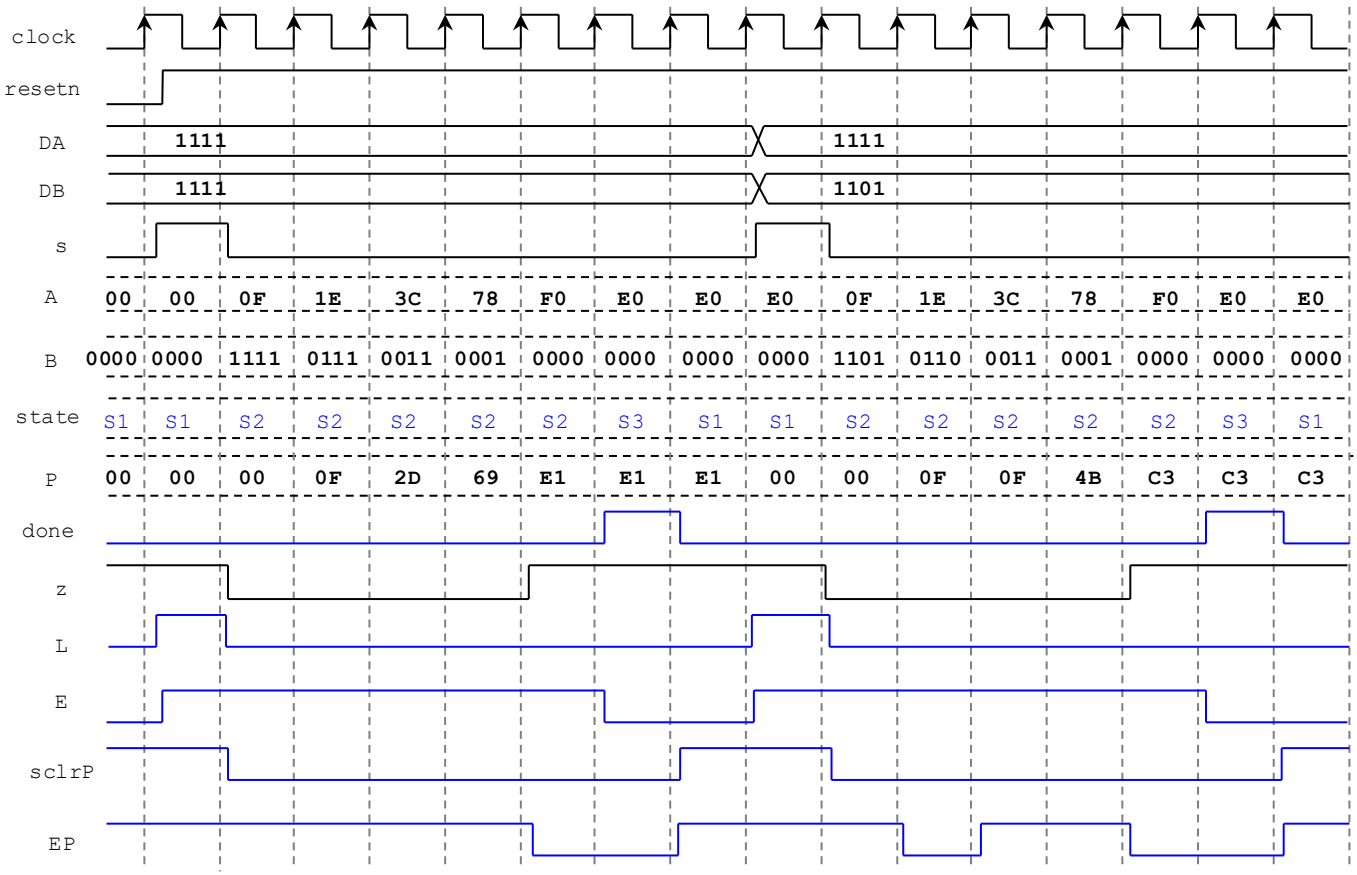
Parallel access shift register: If E=0, the output is kept A ($2n$ bits, left shift), B (n bits, right shift)

```
if E = 1 then
   if s_l = '1' then
      Q ← D
   else
      Q ← shift in 'din'
          (to the left (A) or right (B))
  end if;
end if;
```

- **Algorithmic State Machine (ASM) chart ⇒**

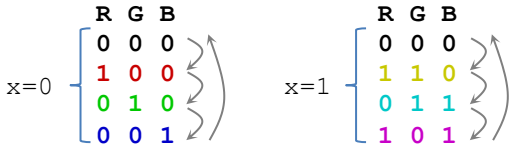


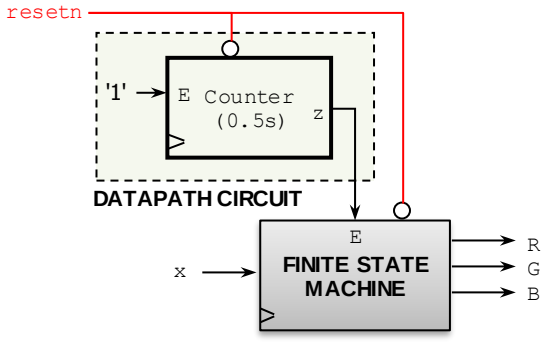
- **Timing Diagram example:** $n = 4$

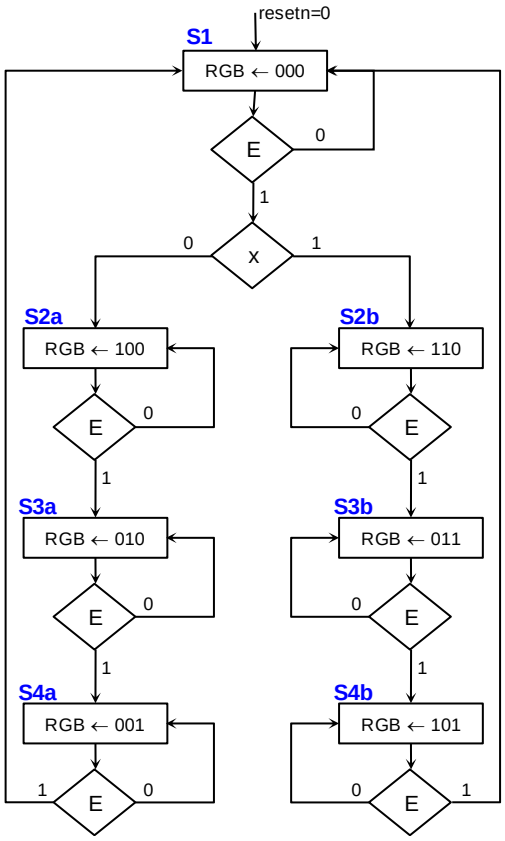


EXAMPLE: RGB LED CONTROL

- We change the color of the RGB LED every 0.5 seconds. Two modes are allowed (based on an input 'x').
  - ✓ First Mode: BLACK → RED → GREEN → BLUE
  - ✓ Second Mode: BLACK → YELLOW → CYAN → VIOLET

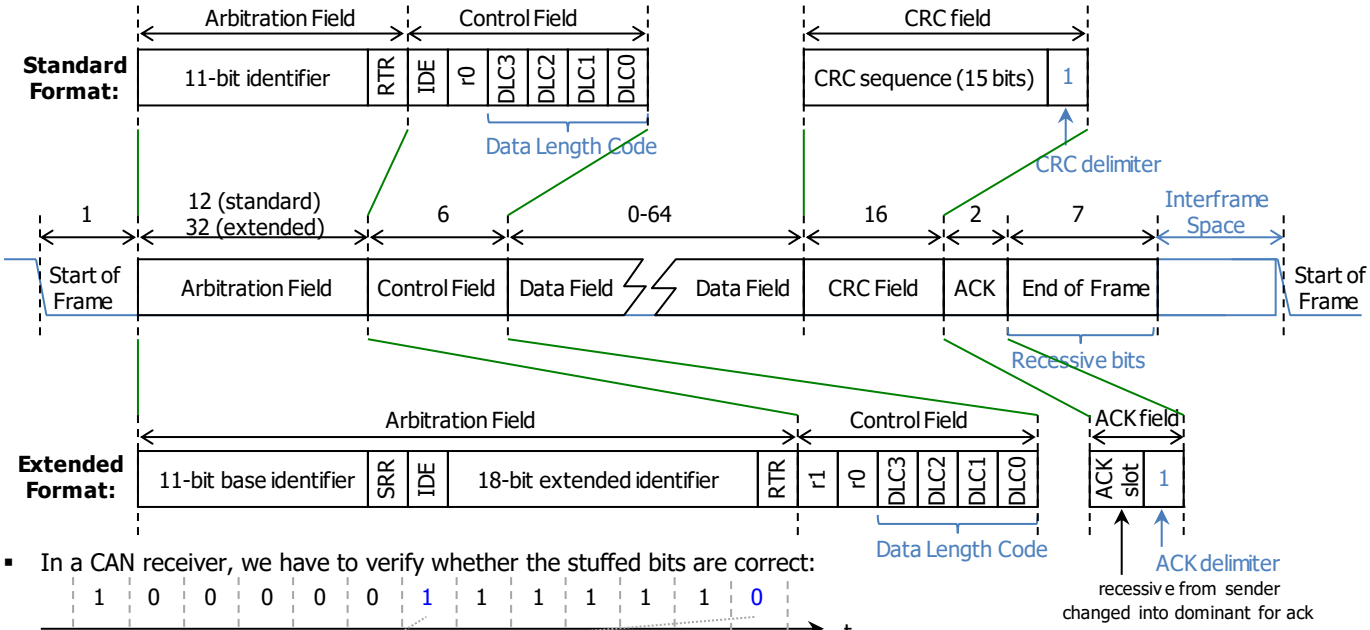




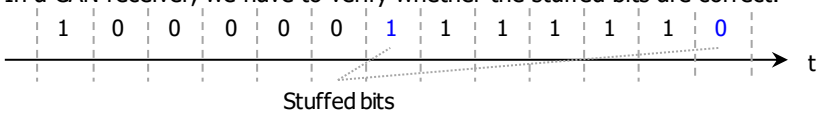


EXAMPLE: CAN BIT STUFFING (RECEIVER)

- Whenever a CAN transmitter detects 5 consecutive identical bits, it inserts a complemented bit. This is applied only to the start-of-frame field, arbitration, control, data, and CRC field. The figure shows a frame in the CAN version 2.0b.

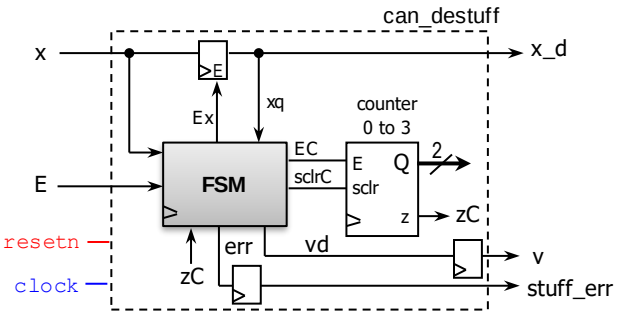


- In a CAN receiver, we have to verify whether the stuffed bits are correct:

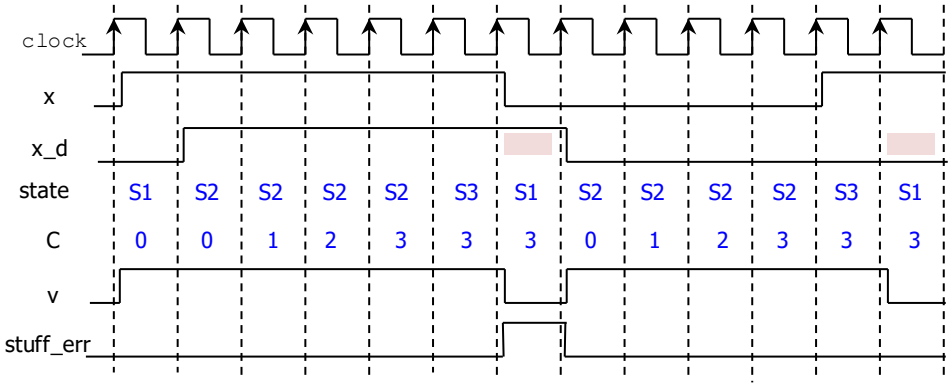


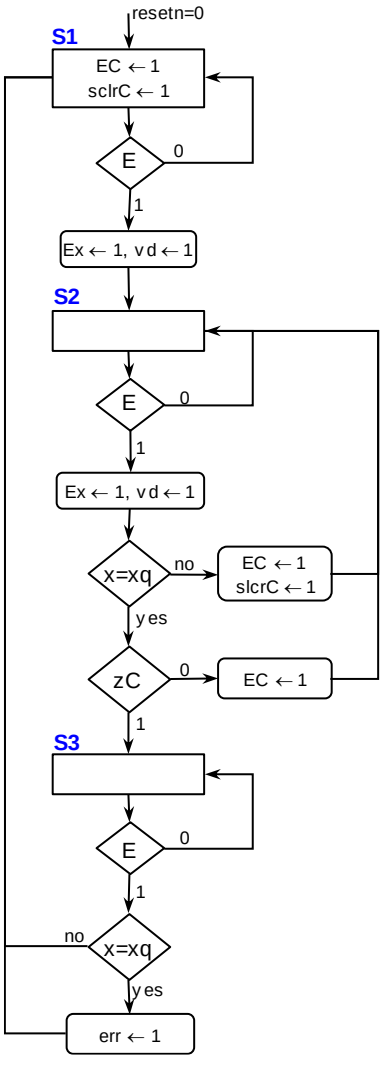
DIGITAL SYSTEM (FSM + Datapath Circuit)

- Input signal x: qualified by E. The system generates x_d (de-stuffed x) qualified by v. If 6 consecutive bits are identical, we assert stuff_err (stuffing error).
- We compare two consecutive samples ($x[n]$ and $x[n-1]$). Here, $x = x[n]$ and $x\_d = x[n-1]$. To determine whether 5 consecutive bits are identical, we compare $x[n]$ and $x[n-1]$ four times.



- Finite State Machine:
  - ✓ S1: captures the first bit, though there is no other bit to compare it to.
  - ✓ S2: captures the other incoming bits, and it compares two consecutive bits at a time. If they match, then C←C+1, otherwise C←0. This can happen indefinitely until C=3 and these two bits match (i.e., after 4 comparisons, we determined that 5 consecutive bits are identical). Then, we move to S3.
  - ✓ S3: It determines whether the stuffed bit is correct. If the 5th and 6th bits match, there is an error. On the next cycle, x_d is invalid and we do not assert the valid flag v, as we are de-stuffing this extra bit. We then return to S1 and start over.
- In the figure, we start right after resetn was asserted. Cases: 111111, 000001. E=1.

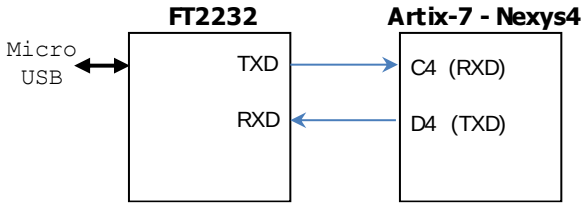




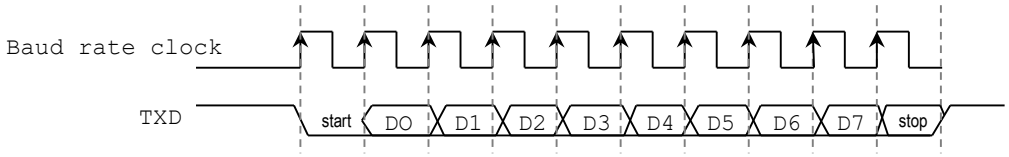
EXAMPLE: SERIAL DATA TRANSMISSION WITH UART (VHDL CODE)

UART Interface

- This interface transfers data asynchronously (clock is not transmitted; transmitter and receiver use their own clocks).
- Data communication: RXD (receive pin), TXD (transmit pin). The FT2232 chip inside the Nexys-4 board is in charge of handling the USB communication with a computer.
- Format of a Frame: Start bit ('0'), 8 to 9 data bits (LSB transmitted first), optional parity bit, and a stop bit ('1').



- Transmitter: Simple design that transmit the data frame at the Baud rate (or bit rate in bps).
- Receiver: It uses a clock signal whose frequency is a multiple (usually 16) of the incoming data rate.



DIGITAL SYSTEM (FSM + Datapath circuit)

- This circuit sends data from the Artix-7 FPGA (that is read via switches) to the FT2232 chip.
- For a baud rate of 9600 bps, the Baud rate clock is 9600 Hz. The bit time is 104.2 us.

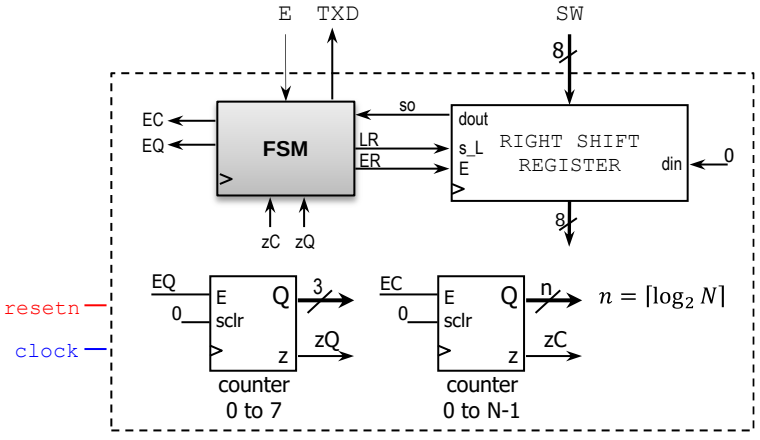
  Then: $N = \frac{1/9600}{10\ ns} = 10416$. We need a counter modulo-N in order to generate the proper time interval (bit time of 104.2 us).

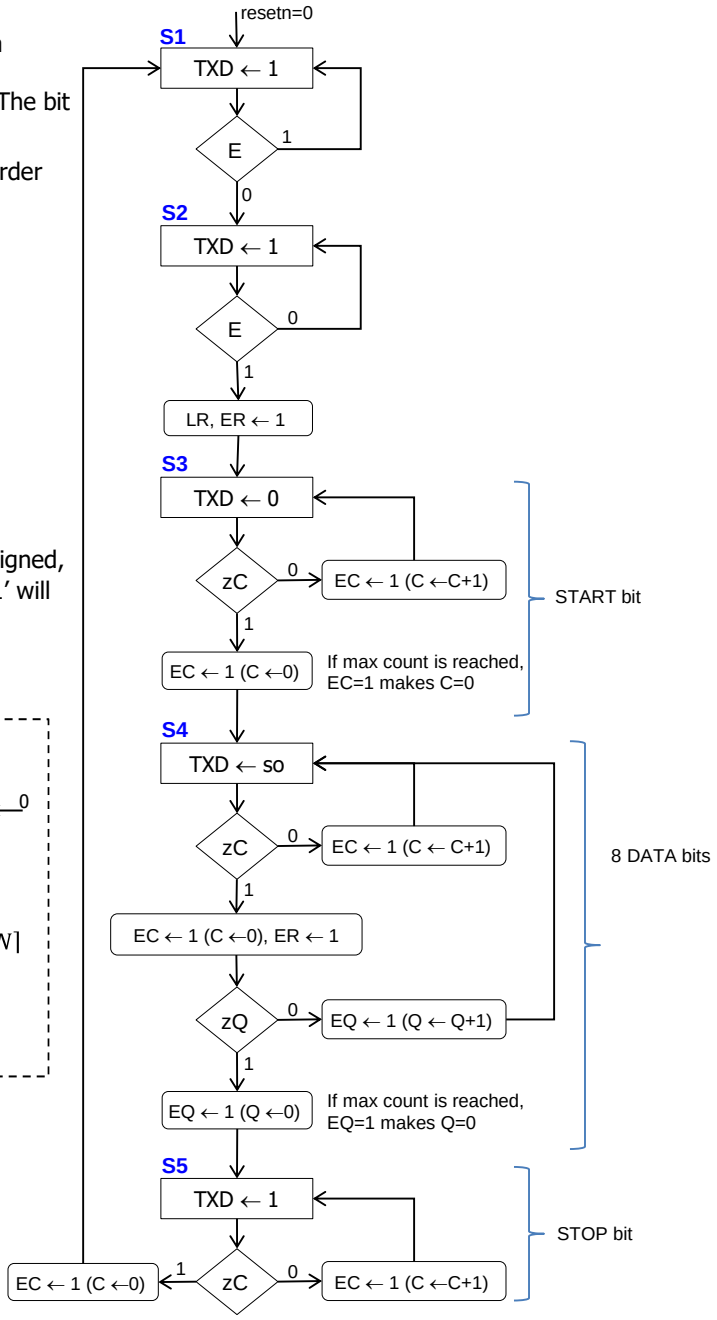
- Generic component (counter): Behavior on the clock tick:

  If E=0, the count stays.

```
if E = 1 then
   if sclr = 1 then
      Q ← 0
   else
      Q ← Q+1
  end if;
end if;
* z=1 if Q = N-1 (max. count)
```

✓ Note that the way this counter (my_genpulse_sclr) is designed, once the maximum count is reachd, asserting enable to '1' will set the count to 0.

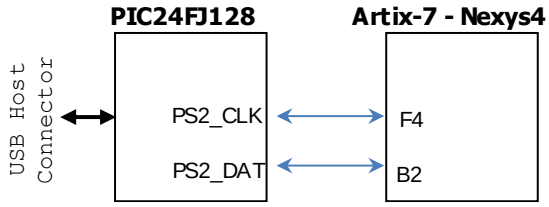




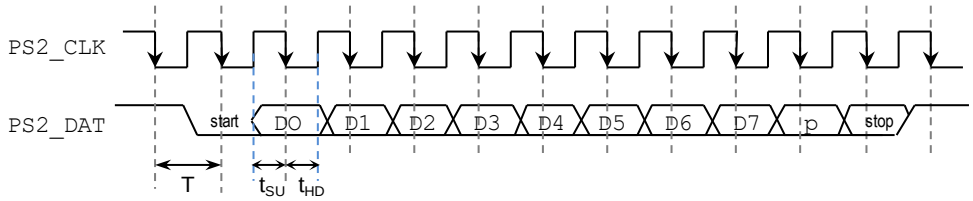
EXAMPLE: PS/2 INTERFACE FOR KEYBOARD (VHDL CODE)

PS/2 Interface

- This interface transfers data synchronously (clock is transmitted alongside data).
- The PIC24FJ128 chip (auxiliary function microcontroller) inside the Nexys-4 DDR board emulates an old-style PS/2 bus and presents a PS/2 protocol to the FPGA. The PS/2 bus signals are converted to the USB protocol. Thus, we can interface with a USB keyboard or mouse as if they were using the PS/2 protocol.
- PS/2 bus uses a bidirectional two-wire serial bus (`PS2_CLK` and `PS2_DATA`) to communicate with a host. The FPGA plays the role of the host.

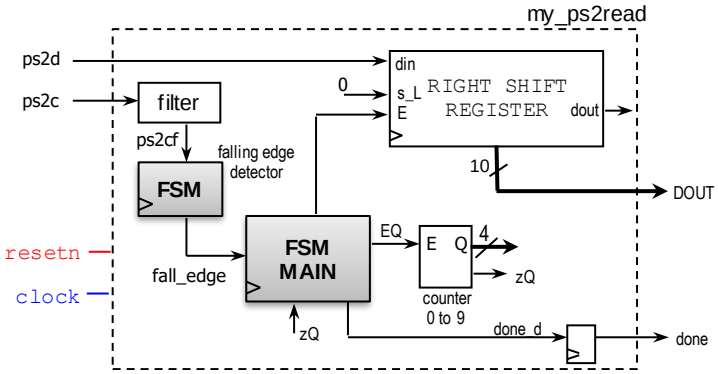


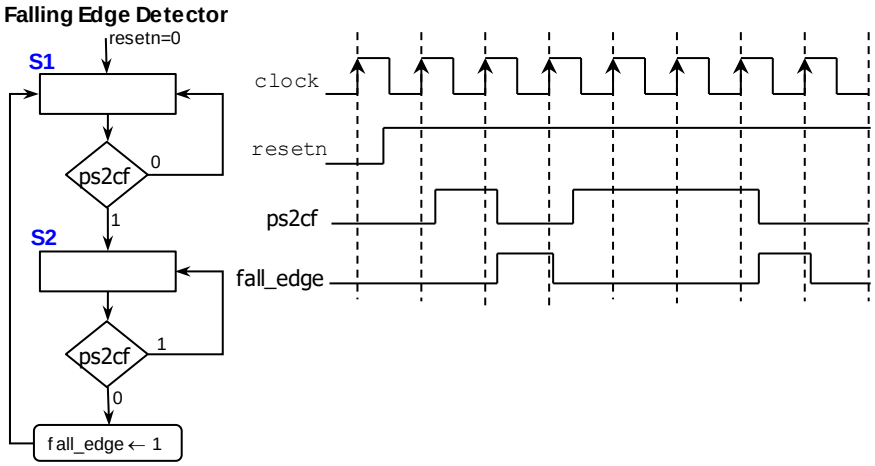
- Format of a Frame: Start bit ('0'), 8 data bits (LSB transmitted first), parity bit (odd), and a stop bit ('1').
- Timing Diagram: Data (1 byte) is captured on the falling edge. The following are times found in the Nexys-4 DDR datasheet: $T \in [60\text{ us}, 100\text{ us}]$, $t_{SU}, t_{HD} \in [5\text{ us}, 25\text{ us}]$

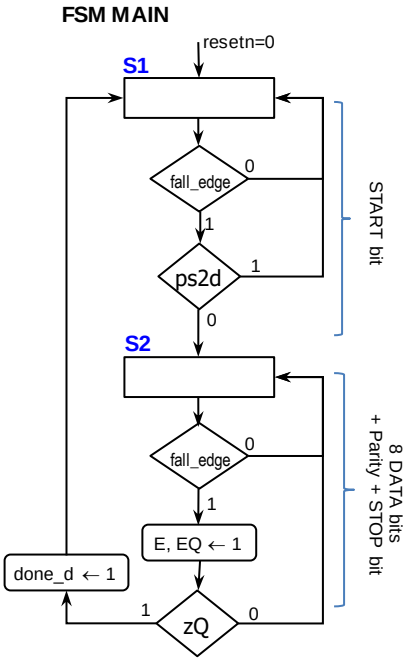


BYTE READ - DIGITAL SYSTEM (FSM + Datapath)

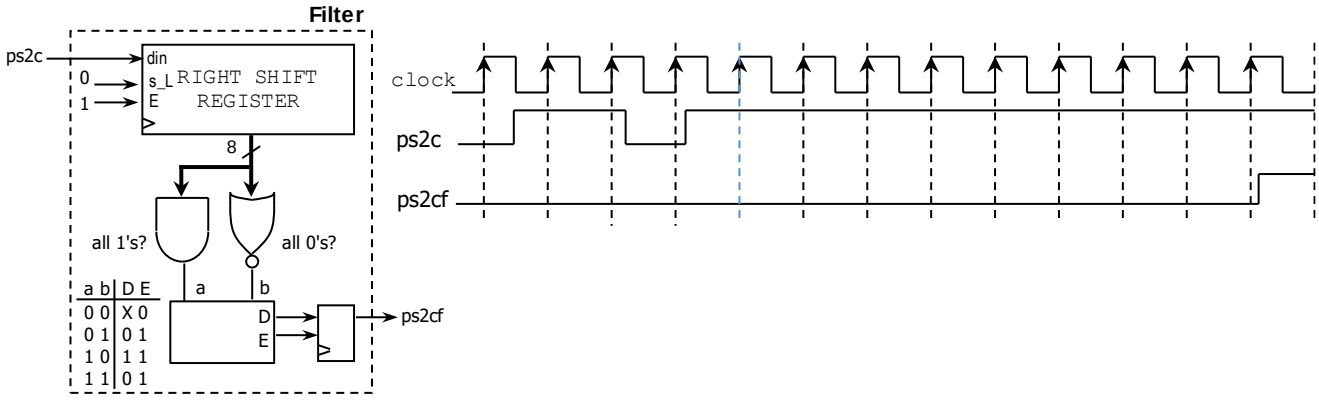
- This system can only receive data from the PIC24FJ128. Thus, it only reads the `PS2_CLK` and `PS2_DAT` signals.
- 10-bit output: `|STOP|parity|D7-D0|`.
- Counter: EQ=1 ⇒ Q ← Q+1. Note that once the maximum count is reached, asserting enable to '1' resets the count to 0.
- Falling Edge Detector. This FSM detects transitions from 1 to 0 on `ps2cf`.





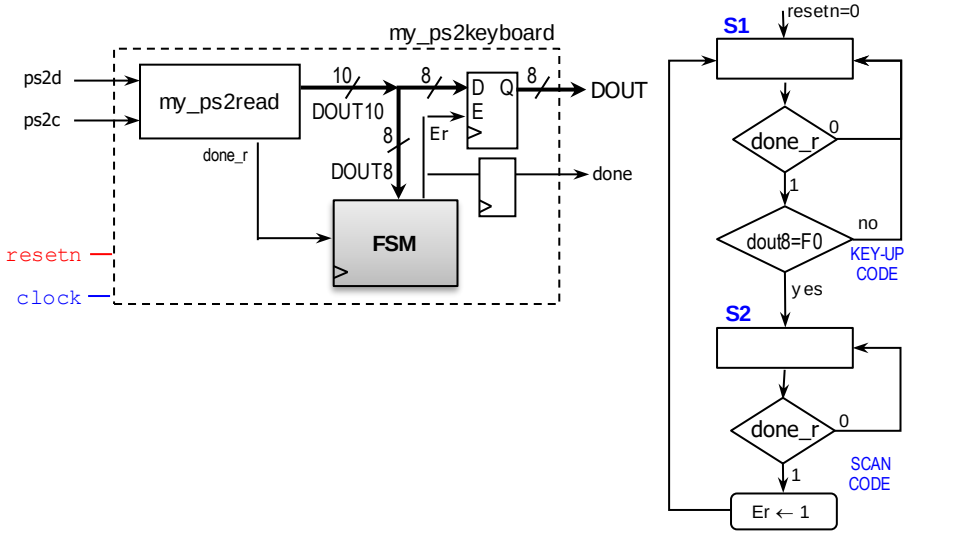


- **Filter**: It makes sure that `ps2c` (PS2_CLK) is constant for at least 8 clock cycles (FPGA operating frequency) before `ps2cf` is assigned a '1' or a '0'. This mitigates the presency of glitches that may have been interpreted as falling edges. The choice of 8 cycles is based on actual testing (use more cycles if you notice glitches affecting the functioning of the circuit.

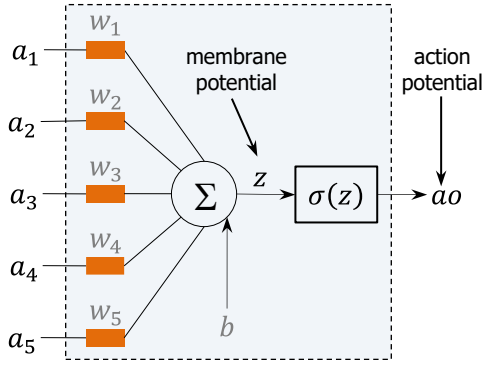


INTERFACING WITH A PS/2 KEYBOARD

- Data from the PS/2 keyboard is given as an 8-bit scan code (see Nexys4-DDR datasheet for a list of scan codes). The following is the protocol that is used when a key is pressed:
  - ✓ If a key is held, the scan code is sent repeatedly every 100 ms.
  - ✓ When the key is released, an `F0` key-up code is sent, followed by the scan code of the released key.
  - ✓ If a key can be shifted to produce a new character (like a capital letter), then a shift character is sent alongside the scan code. Example: `F0 12 [scan code]`.
  - ✓ Some keys, called extended keys, send an `E0` ahead of the scan code. When an extended key is released, an `E0 F0` key-up code is sent, followed by the scan code.
- The circuit presented here cannot read extended keys or shifted keys, only normal keys. It waits for the key-up code (`F0`), and then it captures the scan code. For example, for 'U', the PS/2 keyboard sends `|F0|3C|`, this circuit retrieves `3C`.
- The block `my_ps2read` outputs 10 bits when it receives any code from the PS/2 keyboard. Example:
  - ✓ `DOUT10 = 1111000000`, where parity bit is $\overline{1\oplus1\oplus0\oplus0\oplus0\oplus0\oplus0\oplus0\oplus0} = 1$
  - ✓ `DOUT10 = 1100111100`, where parity bit is $\overline{0\oplus0\oplus1\oplus1\oplus1\oplus1\oplus0\oplus0\oplus0} = 1$
  - ✓ `DOUT10 = 1100011100`, where parity bit is $\overline{0\oplus0\oplus0\oplus1\oplus1\oplus1\oplus0\oplus0\oplus0} = 0$
- The timing diagram shows when the 'U' key is pressed and released: `F0` (key-up code) is sent first, then `3C` (scan code).



EXAMPLE: ARTIFICIAL NEURON



- **Artificial neuron model**. The membrane potential $\boldsymbol{z}$ is a sum of products (input activations a_i by weights w_i) to which a bias term b is added. The action potential $\boldsymbol{ao}$ is modeled as a scalar function of $\boldsymbol{z}$. The figure depicts a neuron with 5 inputs. The bias and the weights are constant values.

$$ao = \sigma\left(\sum_i a_i \times w_i + b\right)$$

  - ✓ A popular and simple scalar function is the Rectified Linear Unit (ReLU):
    - $\sigma(z) = z$ if $z \geq 0$, otherwise $\sigma(z) = 0$.

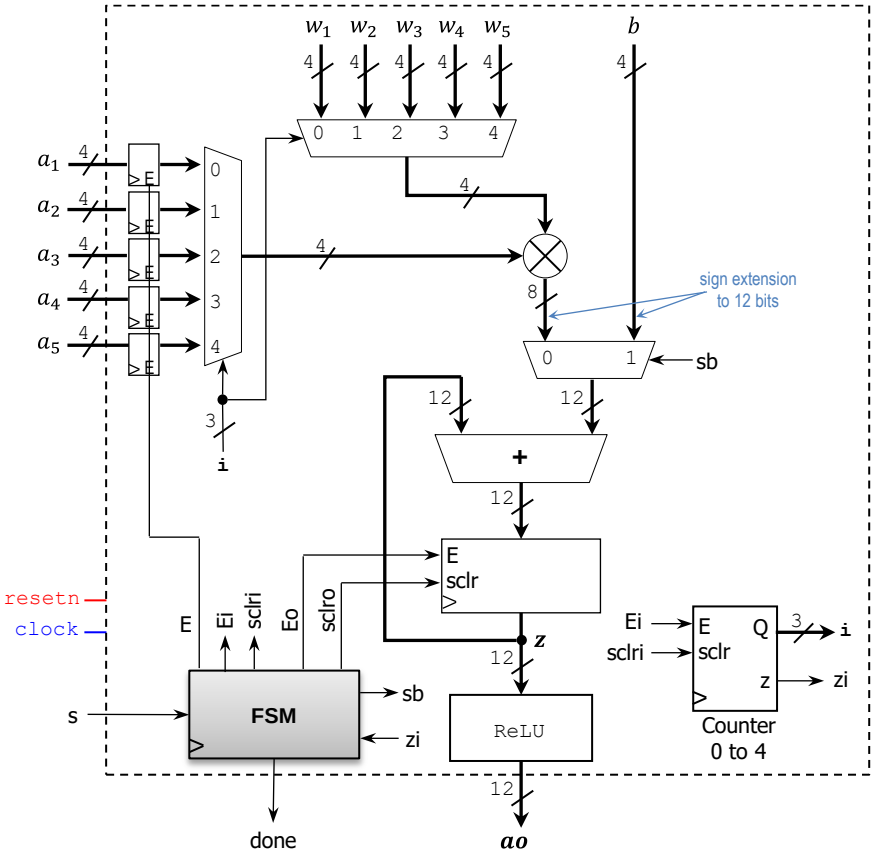
- **DIGITAL SYSTEM** (FSM + Datapath): An iterative architecture for a 5-input neuron is depicted. The circuit captures the input data $(a_1, a_2, a_3, a_4, a_5)$ and then computes $\boldsymbol{z}$ using a multiply-and-accumulate approach (see iterative algorithm). The output $\boldsymbol{ao}$ is computed by applying the ReLU function to $\boldsymbol{z}$.

  - ✓ All data is represented as signed integers:
    - Input activations $(a_1, a_2, a_3, a_4, a_5)$, weights $(w_1, w_2, w_3, w_4, w_5)$, bias (b): 4-bits wide.
      - Weights and biases: They are constant values.
    - Membrane potential $(\boldsymbol{z})$ and action potential $(\boldsymbol{ao})$: 12-bits wide (11 bits suffice, we select 12 for simplicity's sake).

  - ✓ Example (this is what appears in the timing diagram, where $\boldsymbol{z}$ and $\boldsymbol{ao}$ are in hexadecimal format):
    - $w_1 = 0{\times}4$, $w_2 = 0{\times}1$, $w_3 = 0{\times}2$, $w_4 = 0{\times}8$, $w_5 = 0{\times}2$. $\quad b = 0{\times}6$
    - If $a_1 = 0{\times}4$, $a_2 = 0{\times}E$, $a_3 = 0{\times}C$, $a_4 = 0{\times}5$, $a_5 = 0{\times}A$.
    - Then $\boldsymbol{z} = 4{\times}4 + -2{\times}1 + -4{\times}2 + 5{\times}(-8) + -6{\times}2 = -40 = 0{\times}FD8$. Finally, $\boldsymbol{ao} = 0{\times}000$

  - ✓ Components:
    - Counter 0 to 4: If E=1, sclr=1, then Q ← 0. If E=1, sclr=0, then Q ← Q+1. Also: z=1 if Q = 4, else z=0.
    - Register: If E=1, sclr=1→ Clear. If E=1, sclr=0 → Load data.
    - 4×4 Signed Multiplier: Combinational circuit, whose result is 8-bits wide (sign-extended to 12 bits).
    - ReLU: Combinational Block that implements the ReLU operation. For example, if $\boldsymbol{z} = 0{\times}FD8$, then $\boldsymbol{ao} = 0{\times}000$

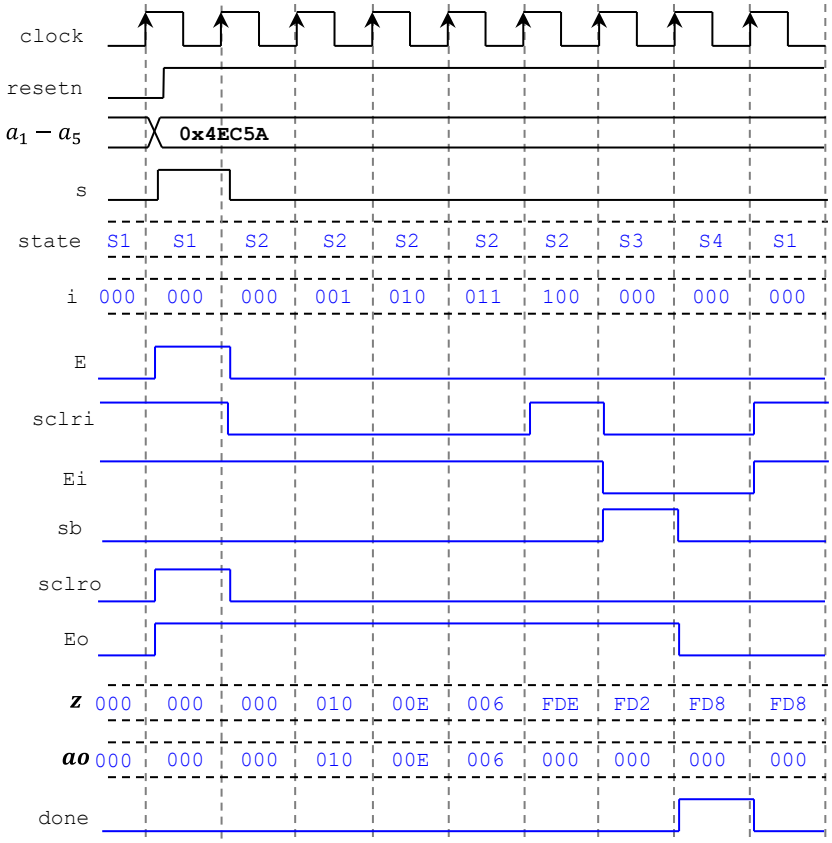
  - ✓ FSM: The process begins when s is asserted, at this moment we capture a_1, a_2, a_3, a_4, a_5 on the input registers. Then $\boldsymbol{z}$ is updated until the counter reaches its maximum value (4). The signal done is asserted when the final result is computed.



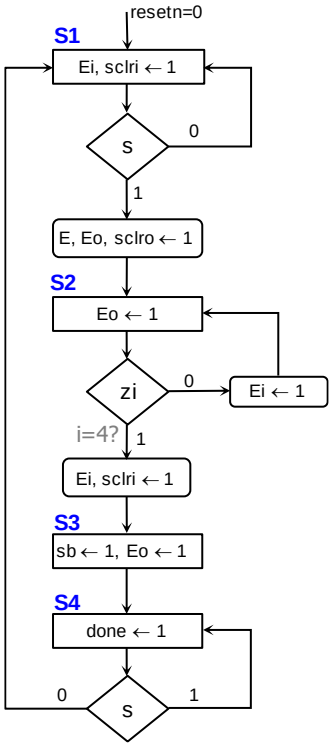
$$ao = \sigma_{ReLU}\left(\sum_{i=1}^{5} a_i \times w_i + b\right)$$

ALGORITHM

$z \leftarrow 0$
$for\ i = 0\ to\ 4$
$\quad z \leftarrow z + a_i \times w_i$
end
$z \leftarrow z + b$
$ao \leftarrow z\ \ if\ z \geq 0, else\ 0$


clock
resetn
$a_1 - a_5$
0x4EC5A
s
state S1 S1 S2 S2 S2 S2 S2 S3 S4 S1
i 000 000 000 001 010 011 100 000 000 000
E
sclri
Ei
sb
sclro
Eo
z 000 000 000 010 00E 006 FDE FD2 FD8 FD8
ao 000 000 000 010 00E 006 000 000 000 000
done

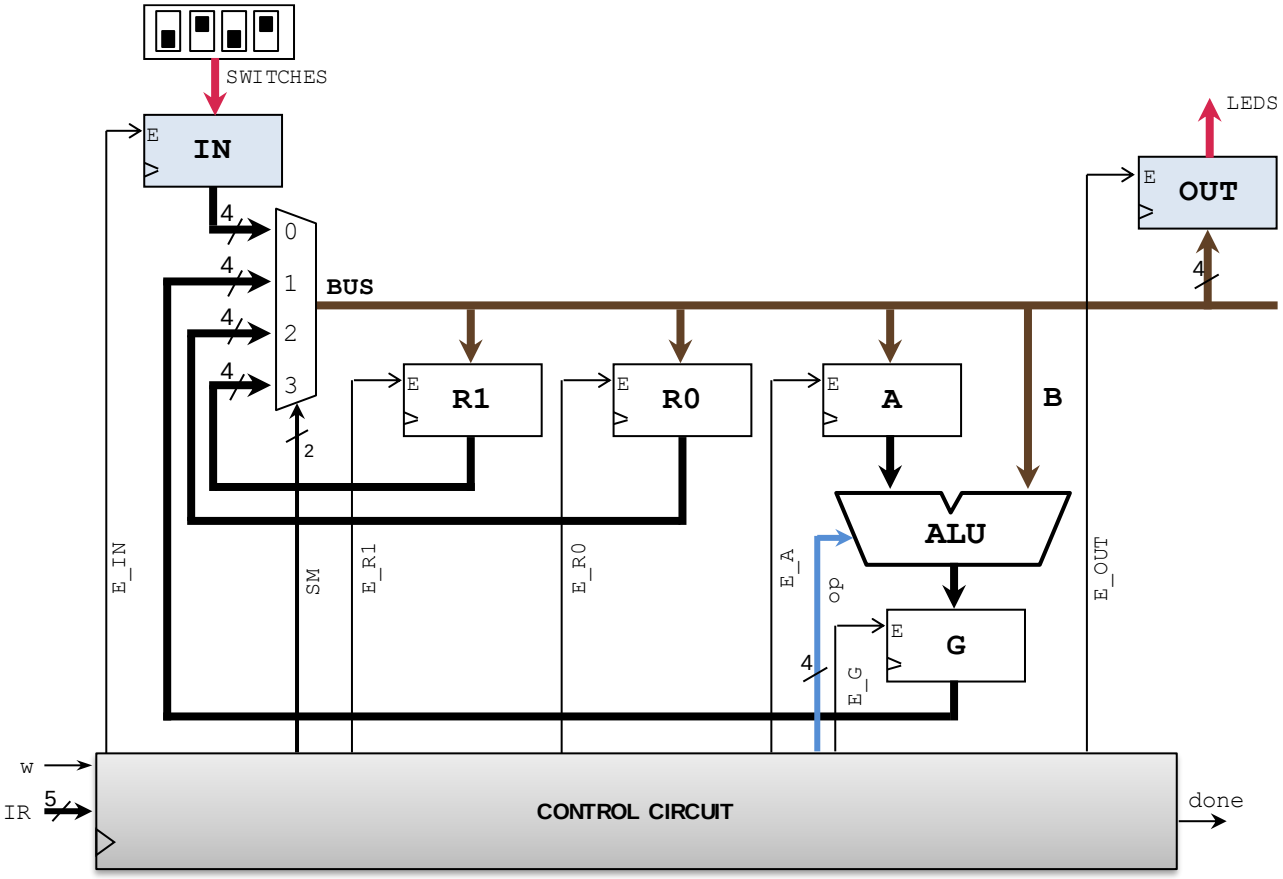
FSM:


resetn=0
S1
Ei, sclri ← 1
S
0
1
E, Eo, sclro ← 1
S2
Eo ← 1
zi
0
Ei ← 1
i=4?
1
Ei, sclri ← 1
S3
sb ← 1, Eo ← 1
S4
done ← 1
S
0
1

EXAMPLE: SIMPLE PROCESSOR

DIGITAL SYSTEM (FSM + Datapath circuit)

- This system is a basic Central Processing Unit (CPU). For completeness, a memory would need to be included.



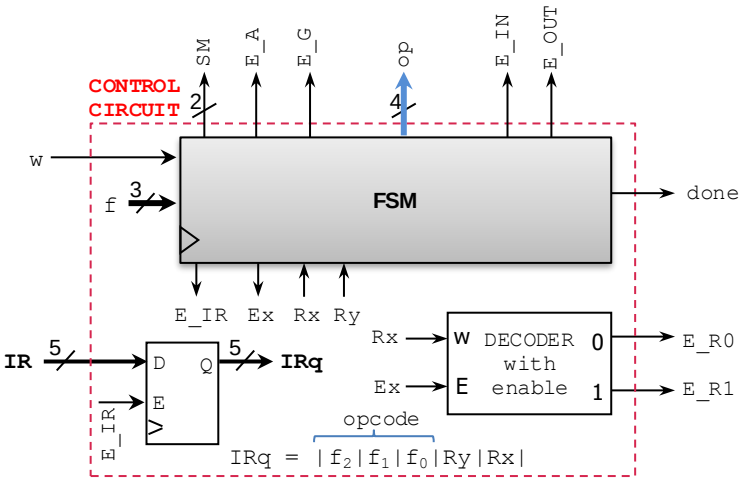
- **Instruction Set**

  Every time w=1, we grab the instruction from `IR` and execute it.
  `Instruction = |f2|f1|f0|Ry|Rx|`. This is called 'machine language instruction' or Assembly instruction.
  - ✓ Opcode (operation code): `IR(4..2)`. These bits specify the operation to be performed.
  - ✓ Operands: `IR(1..0)`. These bits specify the register indices to be used in the operation:
    - Rx: index of the register where the result of an operation is stored (we also read data from Rx). Rx can be R_0 or R_1.
    - Ry: index of the register where we only read operands from. Ry can be R_0 or R_1.

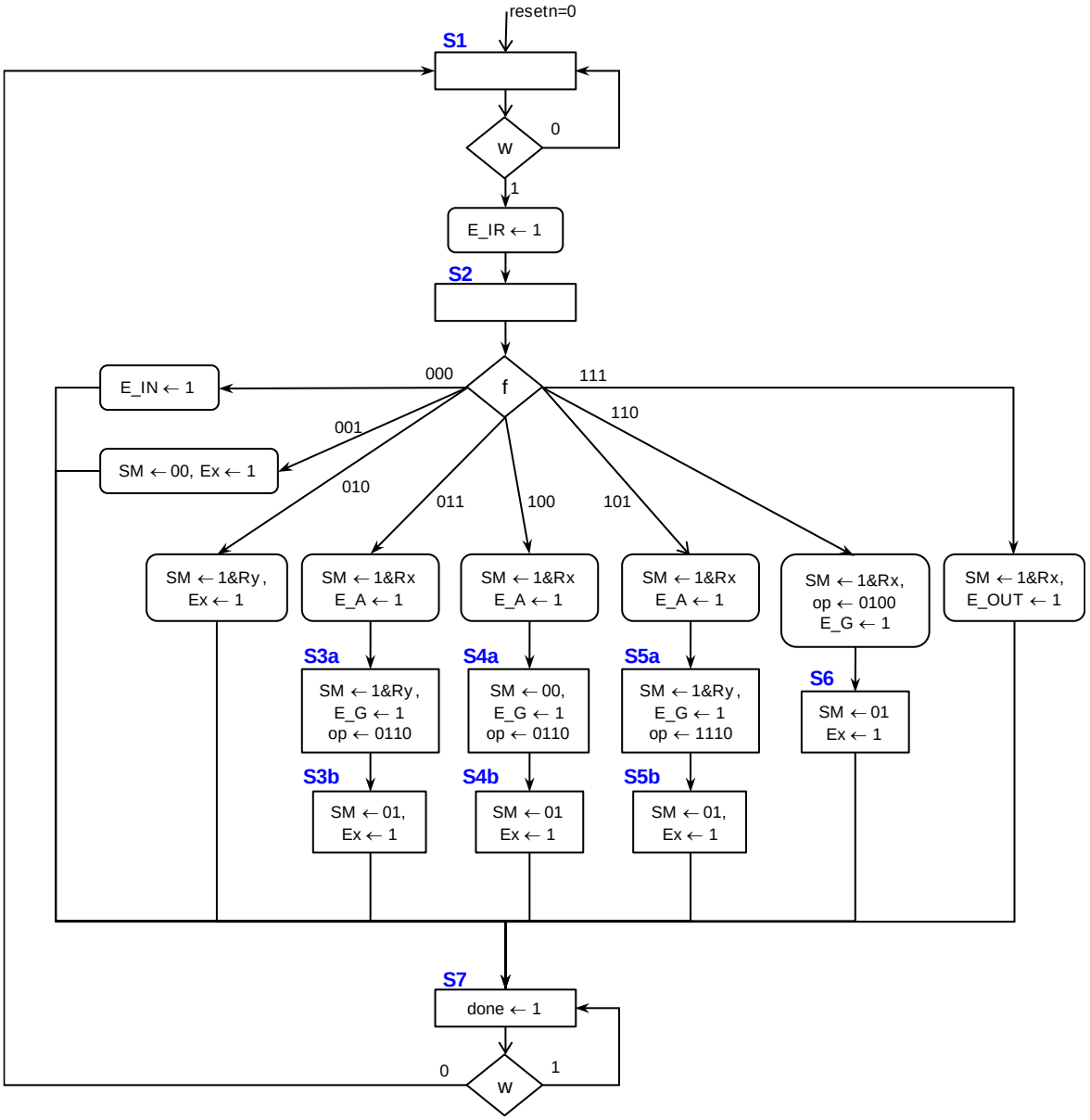
f (IR[4..2])	Operation	Function
000	load IN	IN ← Switches
001	load Rx, IN	Rx ← IN
010	copy Rx, Ry	Rx ← Ry
011	add Rx, Ry	Rx ← Rx + Ry
100	add Rx, IN	Rx ← Rx + IN
101	xor Rx, Ry	Rx ← Rx XOR Ry
110	inc Rx	Rx ← Rx + 1
111	load OUT, Rx	OUT ← Rx

- **Control Circuit**:
  Here, the Control Circuit could be implemented as an FSM. However, in order to simplify the FSM design, the Control Circuit was built out of some combinational units, a register, and an FSM. The 5-bit instruction is captured on the signal IRq.

  ✓ `Ex`: Every time we want to enable register Rx, the FSM only asserts `Ex` (instead of controlling `E_R0`, `E_R1` directly). The decoder takes care of generating the enable signal for the corresponding register Rx.



✓ Control Circuit: FSM



- Arithmetic Logic Unit (ALU):

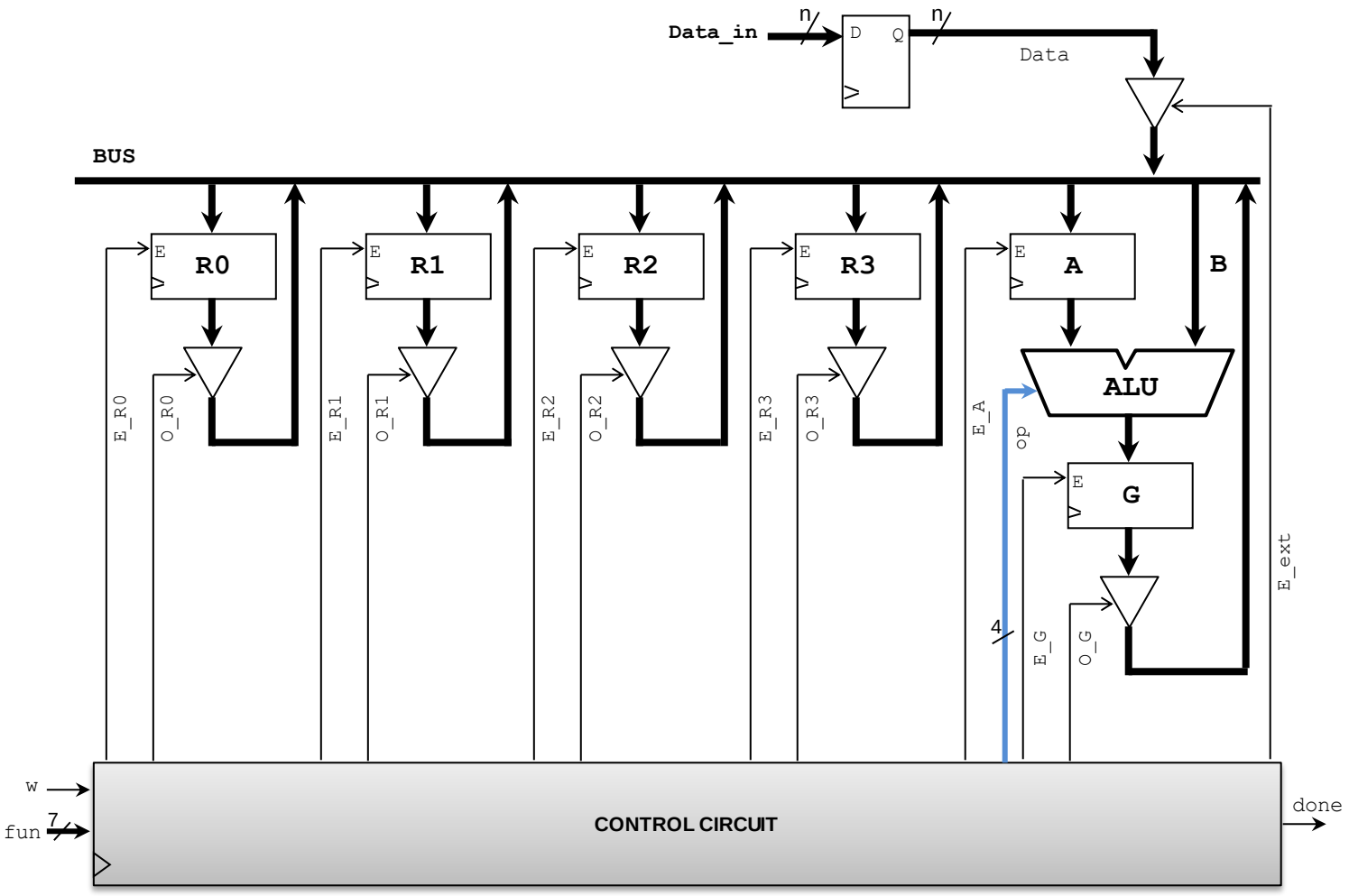
op	Operation	Function	Unit
0000	y <= A	Transfer 'A'	Arithmetic
0001	y <= A + 1	Increment 'A'	
0010	y <= A - 1	Decrement 'A'	
0011	y <= B	Transfer 'B'	
0100	y <= B + 1	Increment 'B'	
0101	y <= B - 1	Decrement 'B'	
0110	y <= A + B	Add 'A' and 'B'	
0111	y <= A - B	Subtract 'B' from 'A'	
1000	y <= not A	Complement 'A'	Logic
1001	y <= not B	Complement 'B'	
1010	y <= A AND B	AND	
1011	y <= A OR B	OR	
1100	y <= A NAND B	NAND	
1101	y <= A NOR B	NOR	
1110	y <= A XOR B	XOR	
1111	y <= A XNOR B	XNOR	

- Assembly code example:

```
load IN;       IN ← 0101 (SWs = 0101)
load R1, IN;   R1 ← 0101
copy R0, R1;   R0 ← 0101, R1 ← 0101
inc R1;        R1 ← 0110
xor R0, R1;    R0 ← 0101 xor 0110 = 0011
add R0, R1;    R0 ← 0011 + 0110 = 1001
load OUT, R0;  OUT ← 1001
```

EXAMPLE: SIMPLE PROCESSOR WITH 3-STATE BUFFERS

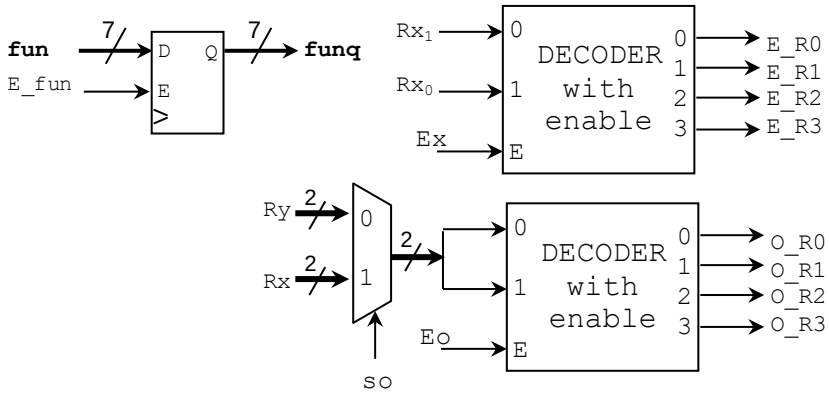
DIGITAL SYSTEM (FSM + Datapath circuit)



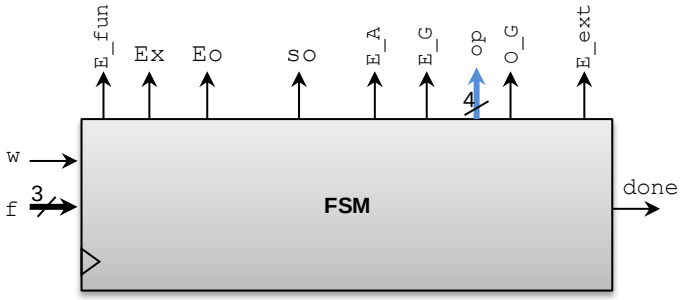
- **Instruction Set**
  Every time w = '1', we grab the instruction from fun and execute it.
  $Instruction = |f_2|f_1|f_0|Ry_1|Ry_0|Rx_1|Rx_0|$. This is called 'machine language instruction' or Assembly instruction:
  - ✓ $f_2f_1f_0$: Opcode (operation code). This is the portion that specifies the operation to be performed.
  - ✓ Rx: Register where the result of the operation is stored (we also read data from Rx). Rx can be R1, R2, R3, R4.
  - ✓ Ry: Register where we only read data from. Ry can be R1, R2, R3, R4.

f	Operation	Function
000	Load Rx, Data	Rx ← Data
001	Move Rx, Ry	Rx ← Ry
010	Add Rx, Ry	Rx ← Rx + Ry
011	Sub Rx, Ry	Rx ← Rx - Ry
100	Not Rx	Rx ← NOT (Rx)
101	And Rx, Ry	Rx ← Rx AND Ry
110	Or Rx, Ry	Rx ← Rx OR Ry
111	Xor Rx, Ry	Rx ← Rx XOR Ry

- **Control Circuit:**
  This is made out of some combinational units, a register, and a FSM:
  - ✓ `Ex`: Every time we want to enable register Rx, the FSM only asserts `Ex` (instead of controlling `E_R0`, `E_R1`, `E_R2`, `E_R3` directly). The decoder takes care of generating the enable signal for the corresponding register Rx.
  - ✓ `Eo`, `so`: Every time we want to read from register Ry (or Rx), the FSM only asserts `Eo` (instead of controlling `O_R0`, `O_R1`, `O_R2`, `O_R3` directly) and `so` (which signals whether to read from Rx or Ry). The decoder takes care of generating the enable signal for the corresponding register Rx or Ry.



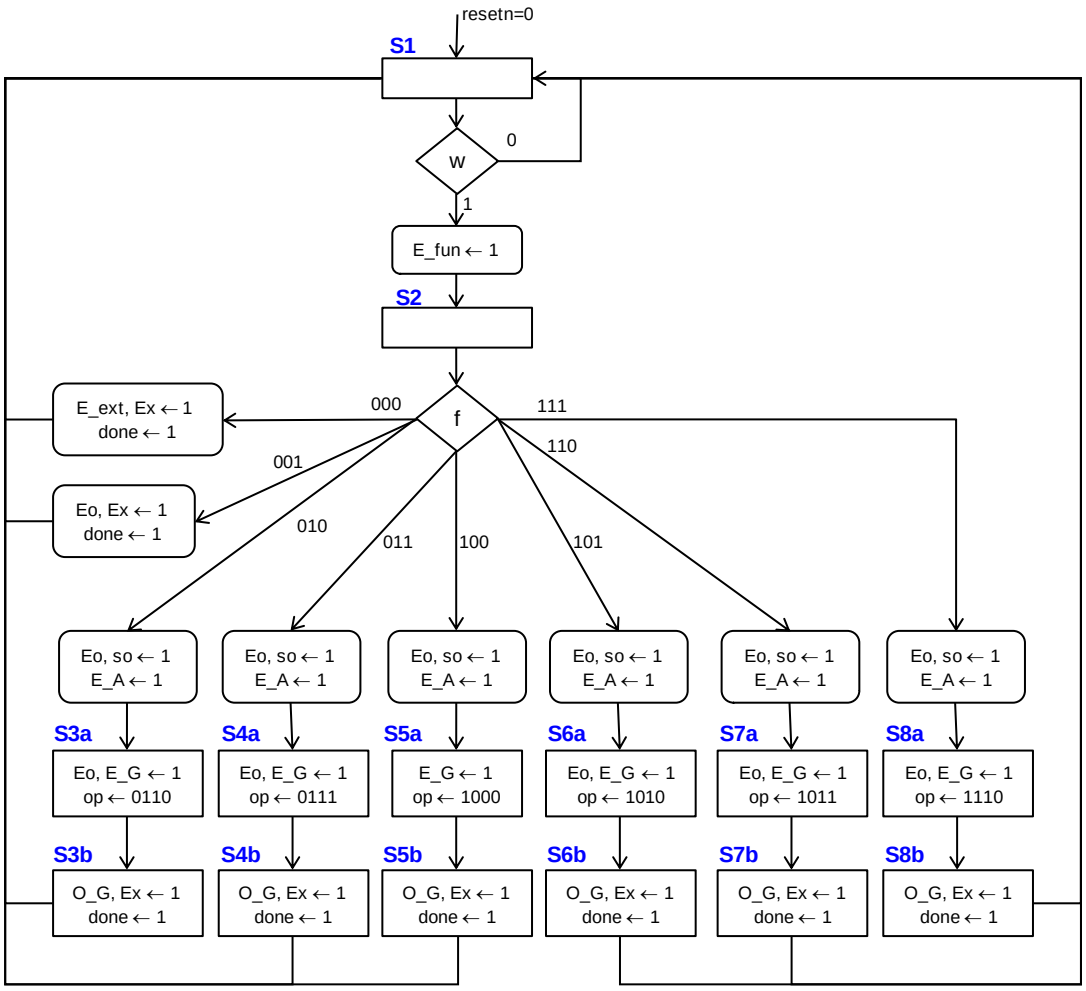
$$funq = |f_2|f_1|f_0|Ry_1|Ry_0|Rx_1|Rx_0|$$



- Arithmetic-Logic Unit (ALU):

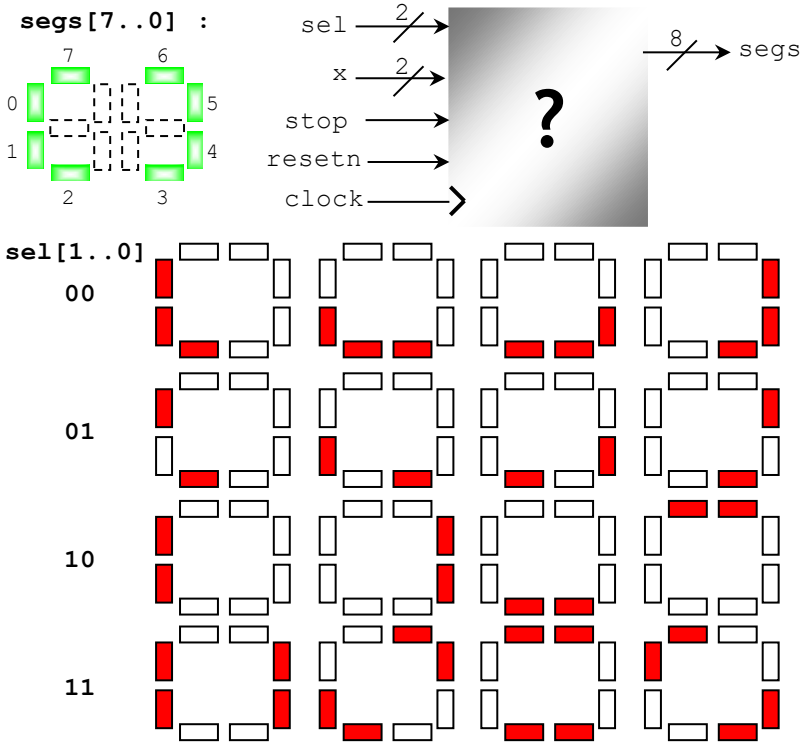
op	Operation	Function	Unit
0000	y <= A	Transfer 'A'	Arithmetic
0001	y <= A + 1	Increment 'A'	
0010	y <= A - 1	Decrement 'A'	
0011	y <= B	Transfer 'B'	
0100	y <= B + 1	Increment 'B'	
0101	y <= B - 1	Decrement 'B'	
0110	y <= A + B	Add 'A' and 'B'	
0111	y <= A - B	Subtract 'B' from 'A'	
1000	y <= not A	Complement 'A'	Logic
1001	y <= not B	Complement 'B'	
1010	y <= A AND B	AND	
1011	y <= A OR B	OR	
1100	y <= A NAND B	NAND	
1101	y <= A NOR B	NOR	
1110	y <= A XOR B	XOR	
1111	y <= A XNOR B	XNOR	

- **Algorithmic State Machine (ASM) Chart**:
  Every branch of the FSM implements an Assembly instruction.

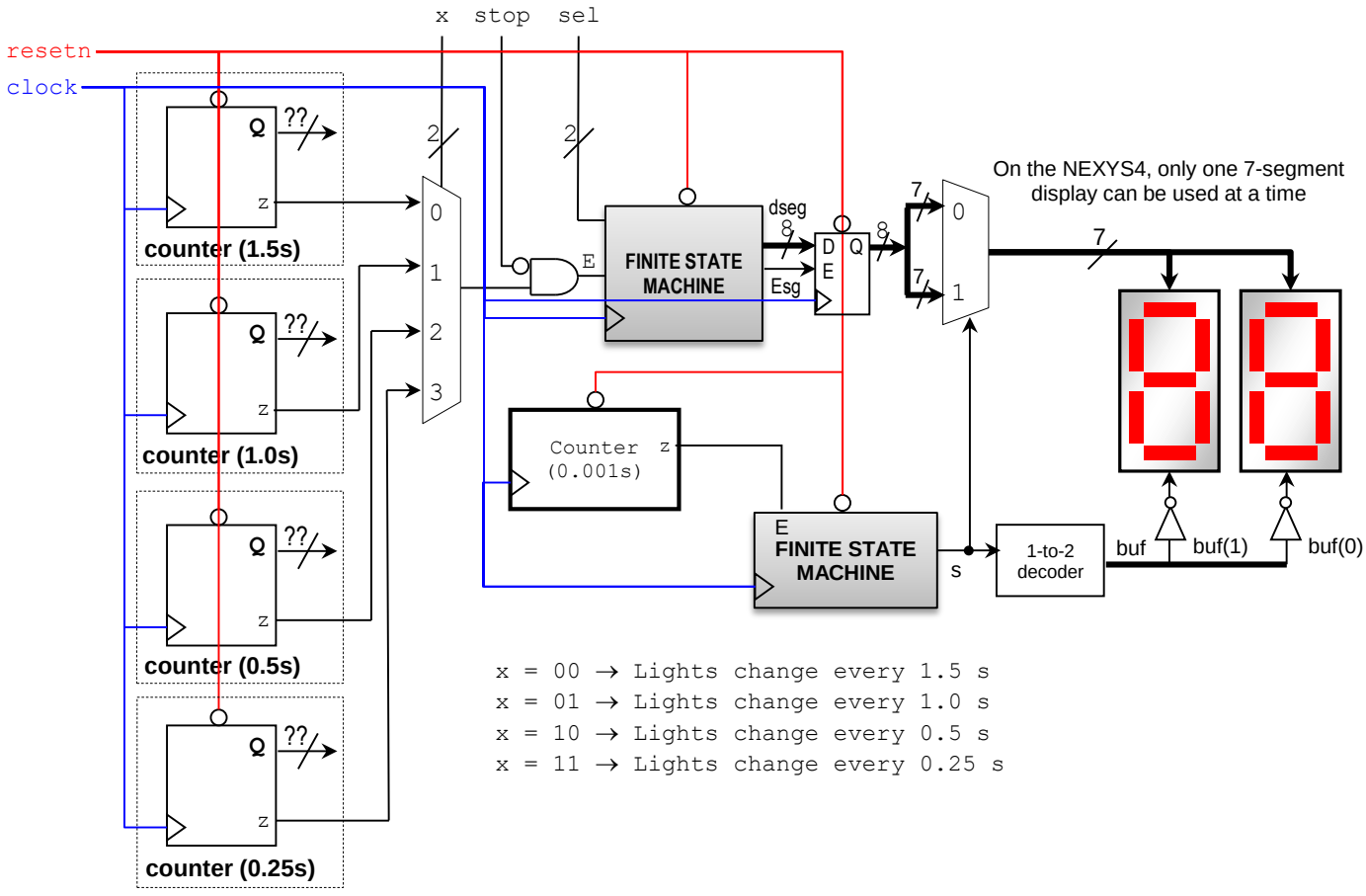


EXAMPLE: DISPLAYING PATTERNS ON 7-SEGMENT DISPLAYS

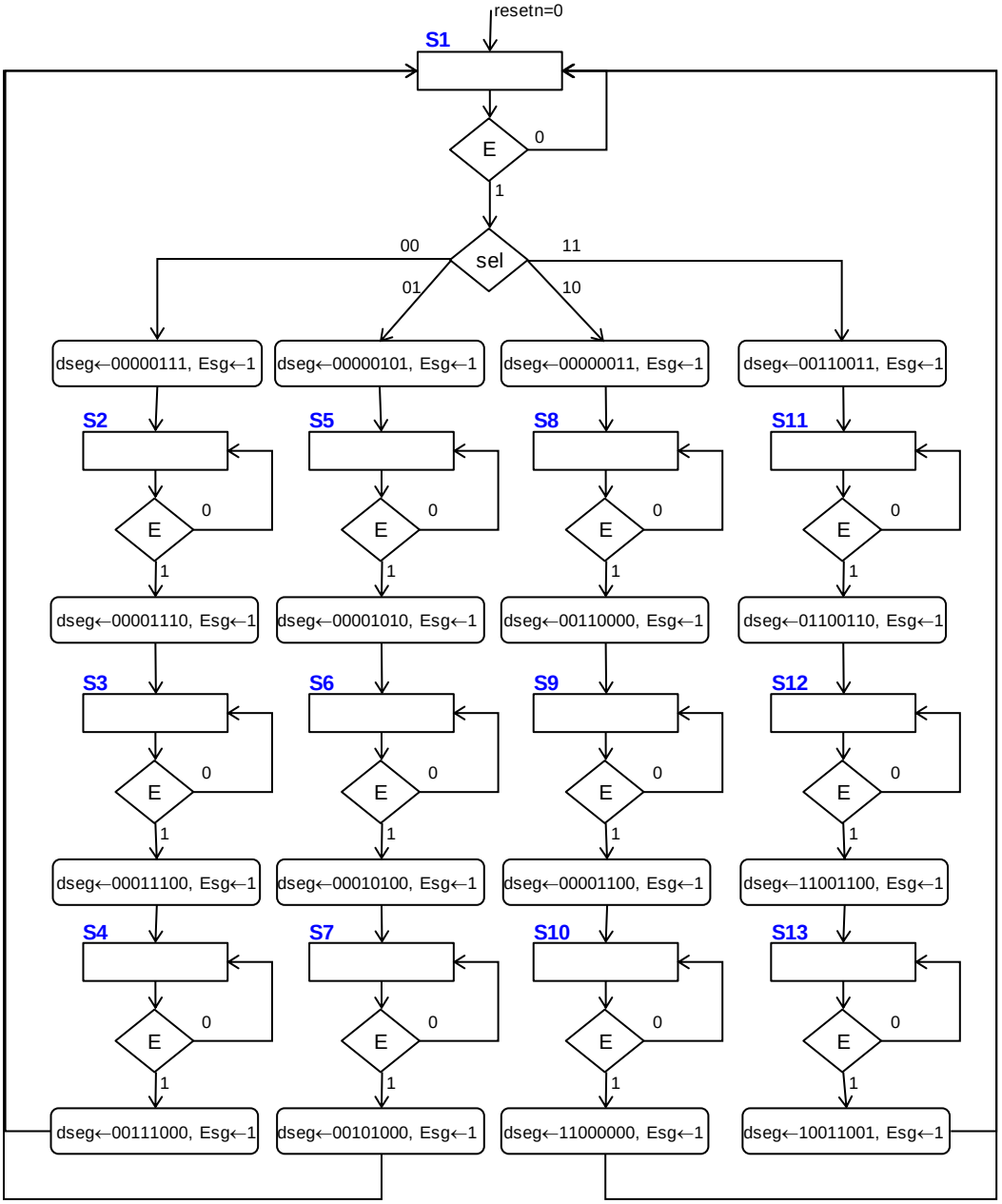
- Different patterns are shown based on the selector 'sel' signal. Two 7-segment displays are used.
- 'stop' input: If it is asserted (stop = 1), the lights' pattern freezes.
- The input 'x' selects the rate of change (every 1.5, 1.0, 0.5, or 0.25 seconds).



DIGITAL SYSTEM (FSM + Datapath circuit)



- **Algorithmic State Machine (ASM) chart:**



- **Algorithmic State Machine (ASM) chart:** This is the FSM that controls the output MUX