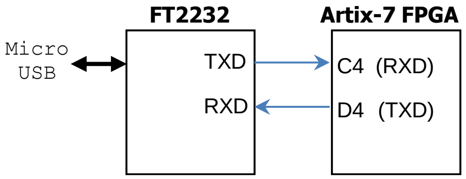
Unit 3- External Peripherals: Interfacing

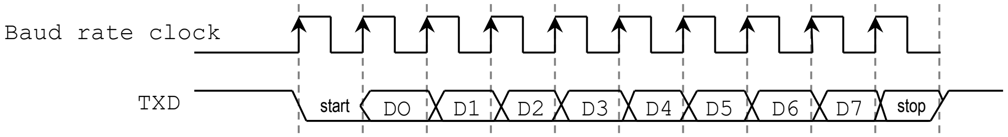
SERIAL COMMUNICATION

SERIAL DATA TRANSMISSION WITH UART

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 handles 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: UART TRANSMITTER (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.

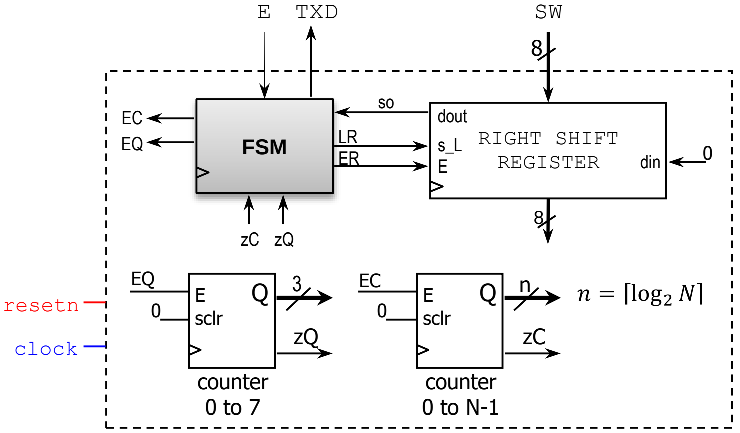
  Then: $N = \frac{1/9600}{10\ ns} = 10416$. We need a counter modulo-N in order to generate the proper time interval (bit time of 104.2 us).

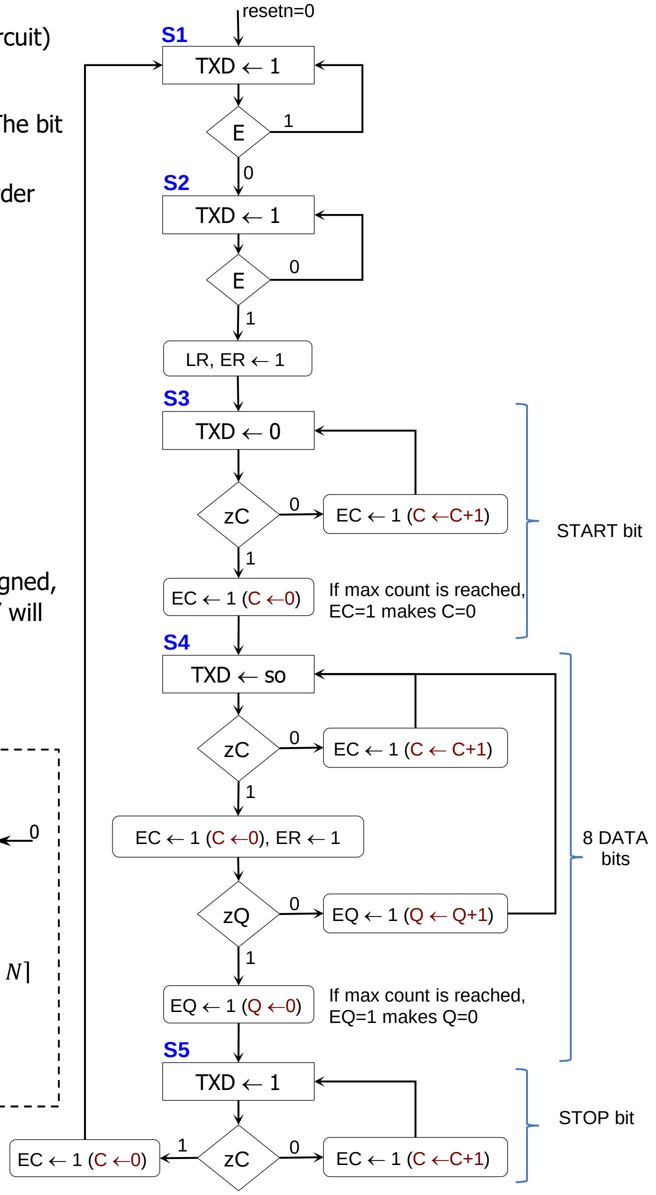
- Generic component (counter): Behavior on the clock tick:

  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.

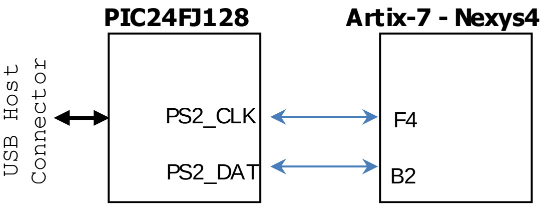




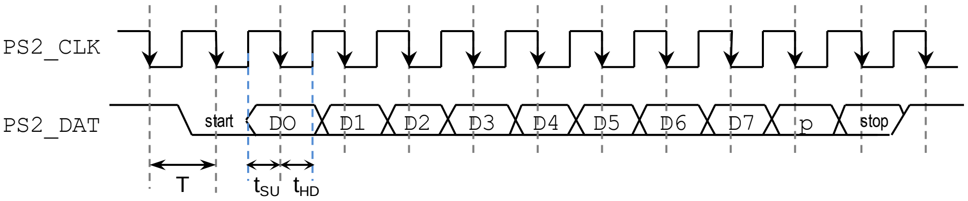
EXAMPLE: PS/2 INTERFACE FOR KEYBOARD

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.

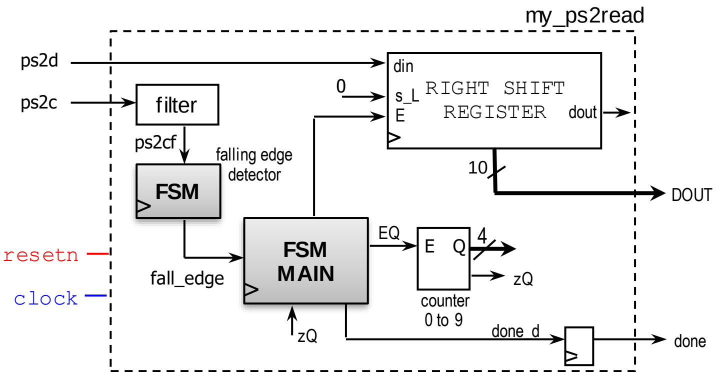


- Format of a Frame: Start bit ('0'), 8 data bits (LSB transmitted first), parity bit (odd), and a stop bit ('1').
- Timing Diagram: Data (1 byte) is captured on the falling edge. The following are times found in the Nexys-4 DDR datasheet: $T \in [60\text{ us},100\text{ us}]$, t_{SU}, $t_{HD} \in [5\text{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

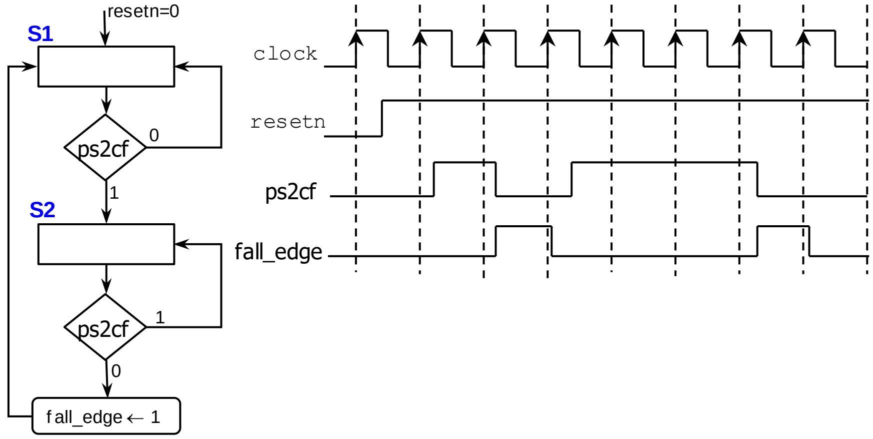


BYTE READ - DIGITAL SYSTEM (FSM + Datapath)

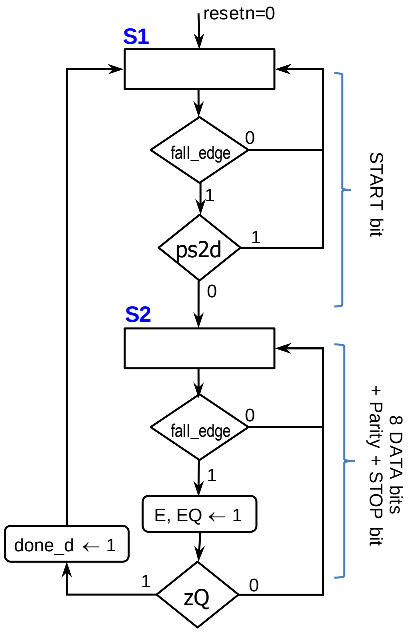
- This system can only receive data from the PIC24FJ128. Thus, it only reads the `PS2_CLK` and `PS2_DAT` signals.
- 10-bit output: `|STOP|parity|D7-D0|`.
- Counter: EQ=1 ⇒ Q ← Q+1. Note that once the maximum count is reached, asserting enable to '1' resets the count to 0.
- Falling Edge Detector. This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



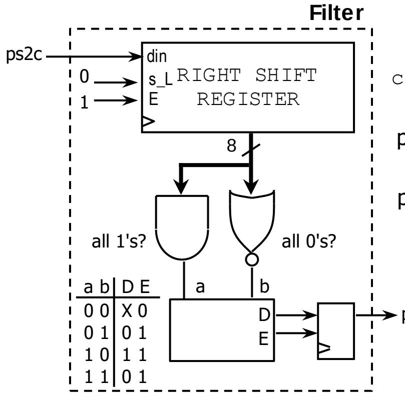
Falling Edge Detector

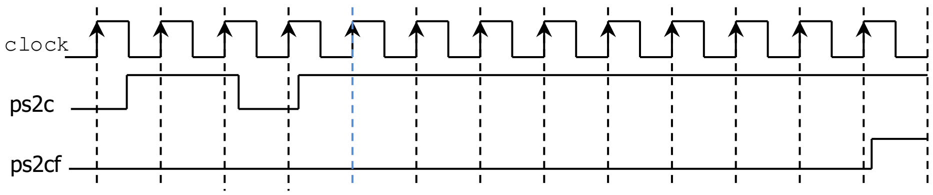


FSM MAIN



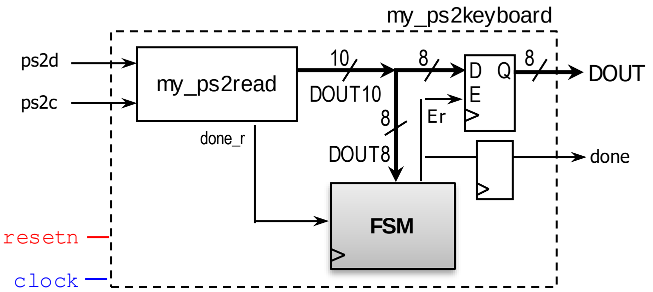
- **Filter**: It makes sure that `ps2c` (PS2_CLK) is constant for at least 8 clock cycles (FPGA operating frequency) before `ps2cf` is assigned a '1' or a '0'. This mitigates the presence of glitches that could be interpreted as falling edges. The choice of 8 cycles is based on actual testing (use more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

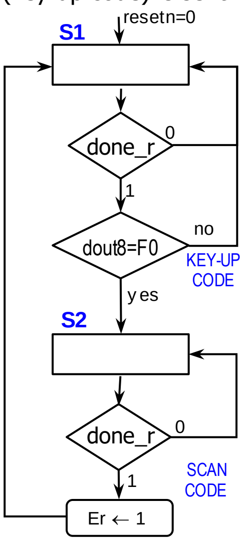


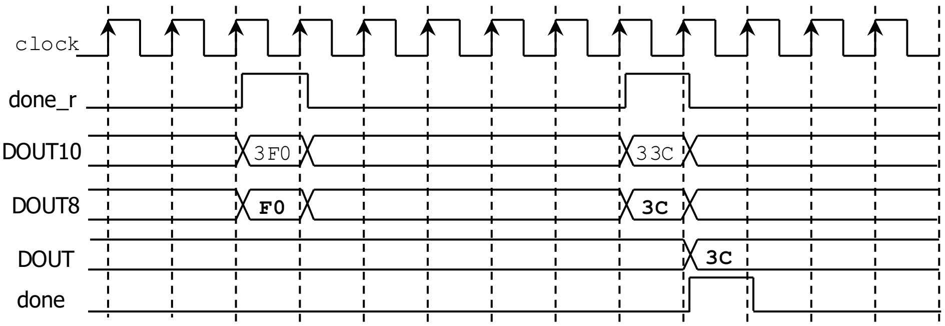


INTERFACING WITH A PS/2 KEYBOARD

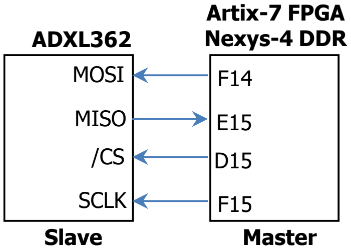
- Data from the PS/2 keyboard is given as an 8-bit scan code (see Nexys4-DDR datasheet for a list of scan codes). The following is the protocol that is used when a key is pressed:
  - ✓ If a key is held, the scan code is sent repeatedly every 100 ms.
  - ✓ When the key is released, an `F0` key-up code 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 $\overline{1\oplus1\oplus0\oplus0\oplus0\oplus0\oplus0\oplus0} = 1$
  - ✓ `DOUT10 = 1100111100`, where parity bit is $\overline{0\oplus0\oplus1\oplus1\oplus1\oplus1\oplus0\oplus0} = 1$
  - ✓ `DOUT10 = 1100011100`, where parity bit is $\overline{0\oplus0\oplus0\oplus1\oplus1\oplus1\oplus0\oplus0} = 0$
- The timing diagram shows when the 'U' key is pressed and released: `F0` (key-up code) is sent first, then `3C` (scan code).





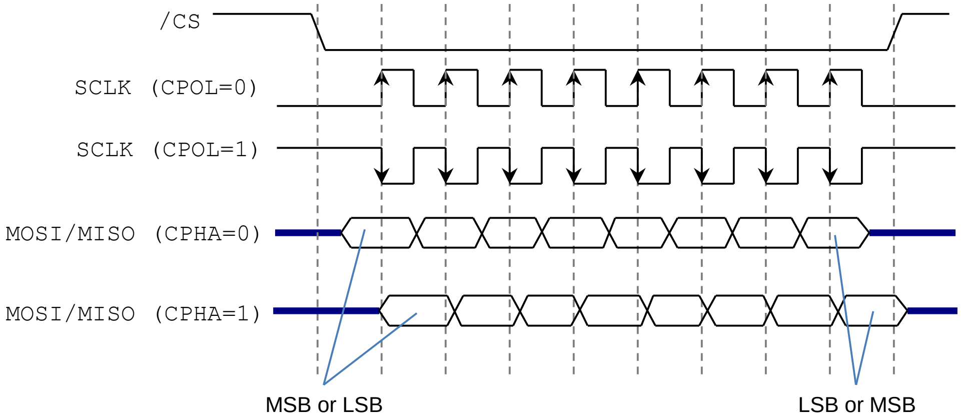


SPI (ACCELEROMETER)



SPI INTERFACE

- Simple 4-wired synchronous (clock is transmitted) serial interface.
- SPI logic signals:
  - ✓ SCLK: Serial clock. Generated by Master.
  - ✓ MOSI: Master Output, Slave Input. Generated by Master.
  - ✓ MISO: Master Input, Slave Output. Generated by Slave.
  - ✓ /CS: Chip select (or Slave Select). Generated by Master.
- Messages are supported that are multiple of 8 bits.
- Clock polarity (CPOL) and Phase (CPHA):



- CPOL = 0: Base value of SCLK is 0.
  - ✓ CPHA=0: Data is captured on rising edge, data is output on falling edge.
  - ✓ CPHA=1: Data is captured on falling edge, data is output on rising edge.
- CPOL = 1: Base value of SCLK is 1.
  - ✓ CPHA=0: Data is captured on falling edge, data is output on rising edge.
  - ✓ CPHA=1: Data is captured on rising edge, data is output on falling edge.

- It is commonly used for short distance communications within embedded systems. Microcontrollers and FPGA designs use SPI to communicate with internal/external peripherals. Large variety of SPI-capable peripherals available: sensors (e.g.: temperature, pressure), ADCs, DACs, touchscreens, memories, LCDs, SD cards.

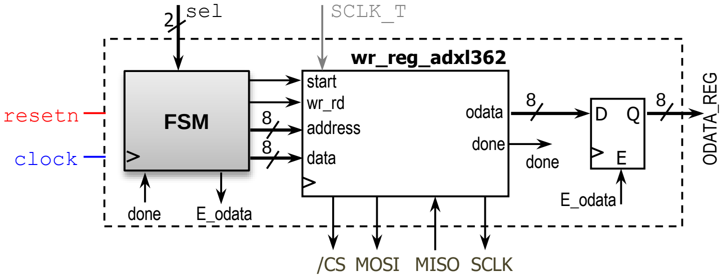
ACCELEROMETER ADXL362

- This 3-axis MEMS device operates as a SPI slave device. We read/write data via a register-based interface: we can write/read a byte or many bytes per bus transaction.
- ADXL362 parameters (range, resolution, ODR are selectable):
  - ✓ Range: ± 2g (default at reset), ± 4g, ± 8g.
  - ✓ Resolution: 1mg/LSB (default at reset), 2 mg/LSB, 4 mg/LSB
  - ✓ Output data rate (ODR): 12.5 – 400 Hz. Default at reset: 100 Hz.
  - ✓ Output resolution: 12 bits. Representation: signed.
- CPOL = 0, CPHA = 0. Many SPI devices work very similarly, although we need to comply with specific timing parameters.

Accelerometer: Basic Controller (code available here)

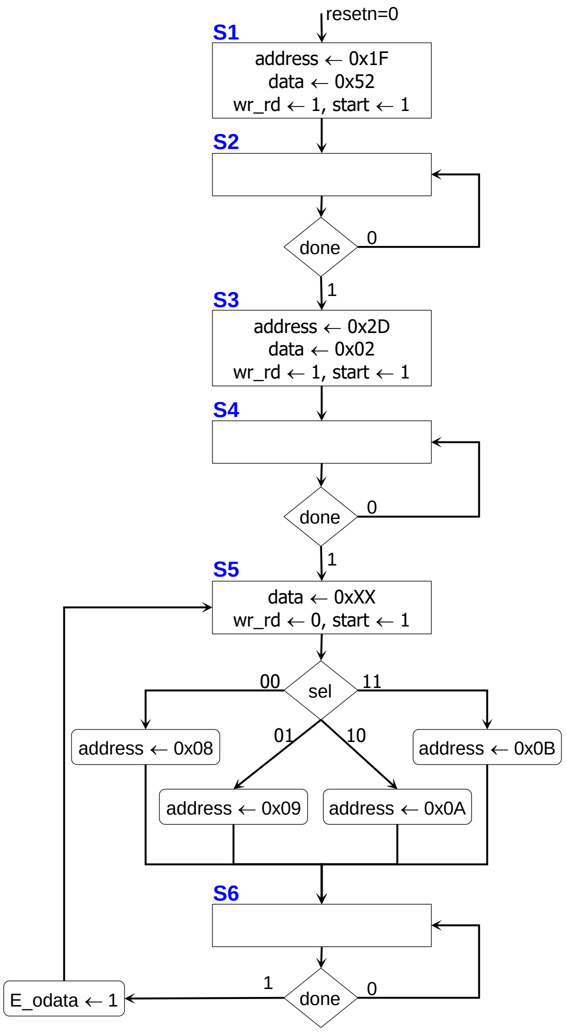
- **Operation:** We first configure the appropriate ADXL362 registers and then proceed to read 8-bit registers. A simple operation mode is listed here. Refer to the ADXL362 datasheet for a complete list of registers and operation modes.
  - ✓ Reset the ADXL362. Write `0x52` on SOFT_RESET (`0x1F`) register.
  - ✓ Activate measurement mode. Write `0x02` on POWER_CTL (`0x2D`) register.
  - ✓ Read any 8-bit register (one per bus transaction). See ADXL362 datasheet for complete list.

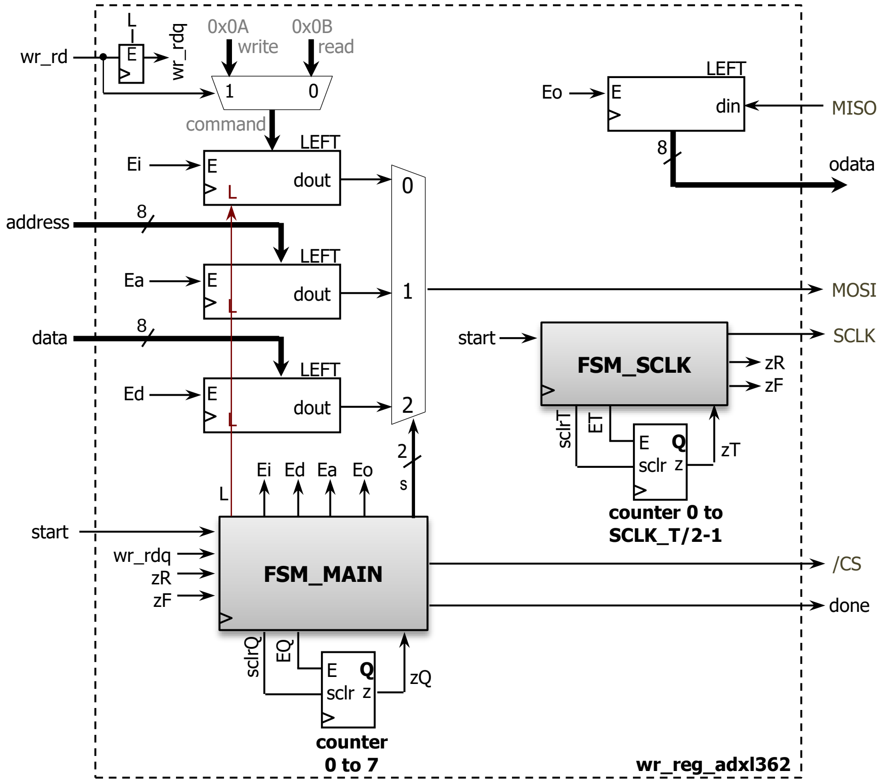
- The basic controller is depicted on the right. The block **wr_reg_adxl362** is the most important: it handles the SPI communication based on address, data and write/read decision. Asserting the *start* signal initiates a transaction. When the operation is completed, the signal *done* is asserted for one clock cycle. If reading data, it appears on *odata*. A new transaction can be started on the next cycle after $done = 1$.

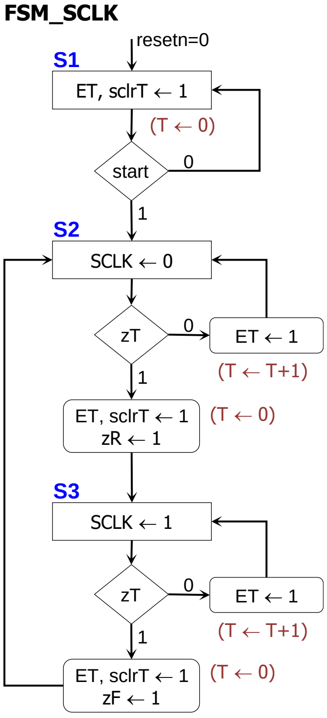


- **FSM**: It issues commands to configure the 2 ADXL362 registers and then read (cyclically) from one of four 8-bit ADXL362 registers (selected by the `sel` input). Data is fetched on the output register. Here, we can read the low-precision 8-bit X, Y, Z measurements (`0x08`, `0x09`, `0x0A`) and the Status Register (`0x0B`).

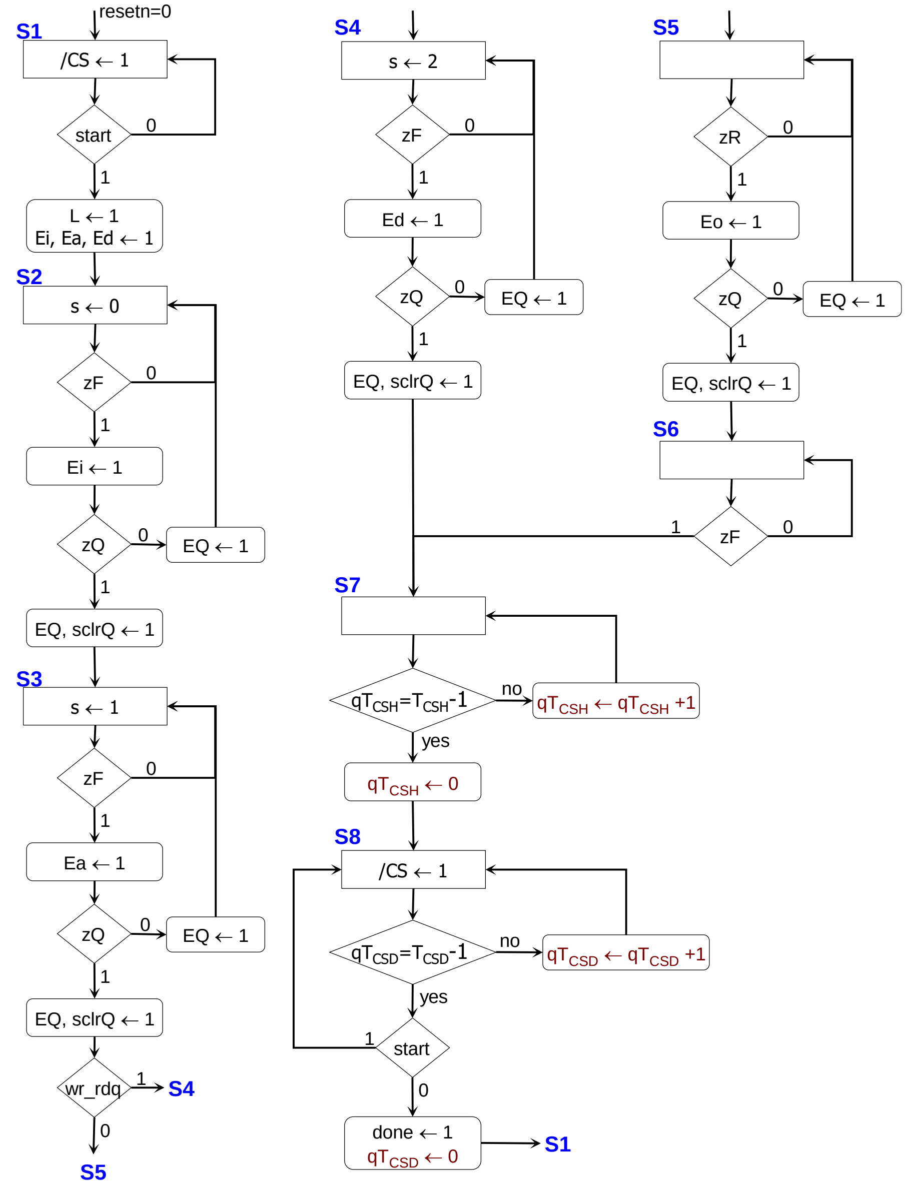
- **wr_reg_adxl362:** This circuit handles the SPI communication with the ADXL362. The user provides address, data, and read/write. Then, a read/write SPI transaction is executed. At every transaction, we write or retrieve 8 bits of data. When writing to the ADXL362, 3 bytes are transmitted: `|command|address|data|`. When reading from the ADXL362, 2 bytes are transmitted `|command|address|`, and 1 byte is read (data) and placed on `odata`.
  - ✓ Circuit Design: It involves implementing the SPI protocol and complying with the ADXL362 timing parameters (see datasheet):
    - C_{SS} (/CS Setup Time): 100 ns
    - t_{CSH} (/CS Hold Time): 20 ns
    - t_{CSD} (/CS Disable Time): 20 ns
    - t_{SU} (Data Setup Time) = t_{HD} (Data Hold Time): 20 ns
    - f_{SCLK}: 2.4 (only when using FIFO) – 8000 KHz.
    - t_{HIGH} (SCLK High Time) = t_{LOW} (SCLK Low Time) = 50 ns. Note that these times only constrain the duty cycle when using large frequencies. The maximum frequency is 8 MHz.
  - ✓ SCLK: not defined by the standard (usually a few MHz). This is specified by the Slave Device (ADXL362: $f_{SCLK} \leq 8000$ KHz).
  - ✓ This design uses a free running SCLK. To comply with the timing parameters: $T_{SCLK}/2\text{-}(t_{CSD}+t_{CSH}) \geq C_{SS} \rightarrow T_{SCLK} \geq 280$ ns ($f_{SCLK} \leq 3.57$ MHz). For T_{SCLK}=280 ns, we have `SCLK_T = 28` (at clock=100 MHz) as the minimum possible value.
  - ✓ To display data on LEDs or 7-segment displays, you need an appropriate refreshment rate. We can choose T_{SCLK}=1 ms (f_{SCLK}=1 KHz) → `SCLK_T`$=10^6$. Since there are 24 SCLK periods in a reading transaction, data is refreshed at 24 ms per sample.
  - ✓ FSM_SCLK: It generates a free running clock of period `SCLK_T` and 50% DC along with rising and falling edge detectors.





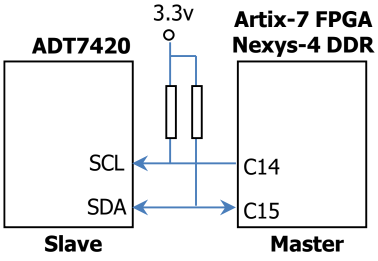


- ✓ **FSM_MAIN**: handles the SPI communication and complies with the ADXL362 timing parameters. Command, address, data (MSB is sent first), same when reading. Note that T_{CSD} = T_{CSH} = 2 (20 ns).
  To comply with the timing parameters, we always wait until the last falling edge in a reading or writing cycle, then wait for T_{CSH} cycles, set /CS=1 for T_{CSD} cycles and then we are back in State S1 for a new transaction.
  Note how we embed the counters for qT_{CSD} and qT_{CSH} inside the FSM.
- ✓ Other approaches do not have a free running SCLK, but instead they only activate it when /CS=0. This approach might make the controlling of the timing parameters simpler (depending on the timing parameters).

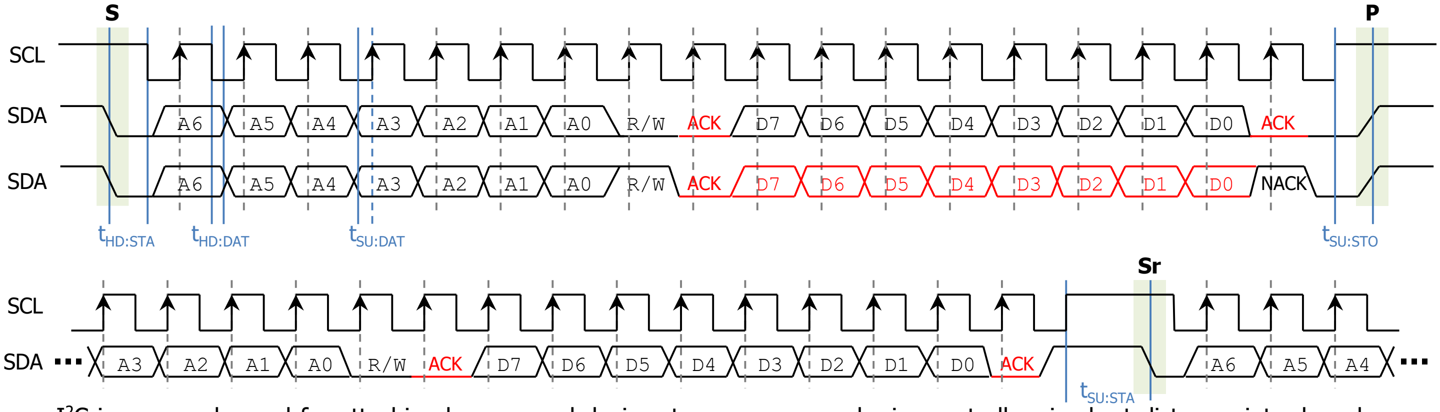


I²C (TEMPERATURE SENSOR)

I²C (Inter-Integrated Circuit) INTERFACE



- Simple 2-wired synchronous (clock is transmitted) serial interface.
- I²C logic signals:
  - ✓ SCL: Serial clock. Generated by Master, defined by the Slave device. The standard specifies a Fast Mode (up to 400 KHz), a High Speed Mode (up to 3.4 MHz), and an Ultra-Fast Mode (up to 5 MHz).
  - ✓ SDA: Bi-directional serial data.
- In general, SCL and SDA are open-drain. There can be one Master and many Slaves. The Master device puts the slave address on the bus, and the slave device with the matching address acknowledges the Mater.
- Slave Address: Unique identifier of a device. 7-bits wide.
- Communication on the I²C bus:
  - ✓ It starts when the master puts the START condition (S) on the bus (a high-to-low transition on SDA while SCL is high). The bus is considered to be busy until the Master puts a STOP condition (P) on the bus (a low-to-high transition on SDA while SCL is high).
  - ✓ I²C data: Transferred in 8-bit packets. There is no restriction to the number of bytes transmitted per data transfer. Each byte transferred must be followed by an acknowledge signal (ACK). ACK (0) is generated by the Slave.
  - ✓ After a START condition (S), the Master writes the 7-bit Slave Address followed by a Read/Write bit, then ACK. Then, the Master writes/reads bytes of data, each byte followed by an ACK. When writing, after the last written byte (followed by ACK), data transmission is terminated by the Master with a STOP condition (P). When reading, only on the last byte, the Master must generate a NACK (Not acknowledge) bit, and then a STOP condition (P). The Master can also generate a repeated START condition (Sr) without first generating a STOP condition (P) to signal that the bus is still busy.
  - ✓ Data bits are read on the SCL rising edge. We must comply with the Slave device timing parameters: $t_{SU:DAT}$, $t_{HD:DAT}$, $t_{SU:STA}$, $t_{HD:STA}$, $t_{SU:STO}$. Unlike a flip flop, $t_{HD:DAT}$ (hold time) is defined as the time the data bit should be on the bus after SCL is high (i.e., after the falling edge).



- I²C is commonly used for attaching lower-speed devices to processors and microcontrollers in short-distance, intra-board communication. Large variety of I²C-capable peripherals available: sensors (e.g.: temperature, acceleration, pressure), ADCs, DACs, touchscreens, memories, LCDs, SD cards.

TEMPERATURE SENSOR ADT7420

- This high accuracy digital temperature sensor operates as an I²C slave device. We read/write data via a register-based interface: we can write/read a byte or two bytes per bus transaction.
- ADT7420 parameters (resolution is selectable):
  - ✓ Output resolution: 13 bits (default at reset), 16 bits. Representation: FX signed.
  - ✓ Resolution: 0.0625°C per LSB (13-bit mode, default at reset), 0.0078125°C per LSB (16-bit mode).
    - ▫ 16-bit mode: FX Format [16 7]. Temperature (°C): $\frac{-2^{15}b_{15}+\sum_{i=0}^{14}b_i 2^i}{2^7}$
    - ▫ 13-bit mode: FX Format [13 4]. This is just the 13 MSBs of the 16-bit result. Temperature (°C): $\frac{-2^{12}b_{12}+\sum_{i=0}^{11}b_i 2^i}{2^4}$
    - ▫ According to the formulas, the temperature range is $[-256°\text{C}, 256°\text{C})$. However, in practice the ADT7420 is guaranteed to measure temperature between -40°C and 150°C.
  - ✓ Slave Address: $10010A_1A_0$. A_1A_0 bits are configurable. Nexys-4 DDR-Board: $A_1A_0 = 11$ → Slave Address: `0x4B`.

Temperature Sensor: Basic Controller (code available here)

- **Operation:** We first configure the appropriate ADT7420 registers and then proceed to read 8-bit registers. A simple operation mode is listed here. Refer to the ADT7420 datasheet for a complete list of registers and operation modes.
  - ✓ Configure the 16-bit mode. Write `0x80` on CONFIG (`0x03`) register.
  - ✓ Read any 8-bit register (one per bus transaction).

- The Basic Controller interacts with the following registers: (refer to the ADT7420 datasheet for a complete list of registers).

Reg. Address	Name	Reg. Address	Name
0x00	TEMP_H	0x03	CONFIG
0x01	TEMP_L		
0x02	STATUS	0x0B	ID

  ✓ 13-bit mode: This requires to write 0x00 on CONFIG register. The 13-bit data will be located in the 13 MSBs of the 16-bit sequence: TEMP_H|TEMP_L.

  ✓ Reading from the ID register results in 0xCB (manufacturer's setting)

- **Communication Protocol:** This protocol runs on top of I^{2}C. Writing/reading here refer to the process of writing/reading to/from a register. This is a bit different from writing/reading data onto the I^{2}C bus (what the I^{2}C protocol specifies). RA: ADT7420 internal Register Address of ADT7420. AD: Slave (ADT7420) I^{2}C Address (0x4B). NACK: Not Acknowledge (1), set by Master, ACK: Acknowledge (0). W: Write bit (0). R: Read bit (1). For AD, RA, DATA, MSB is sent first.

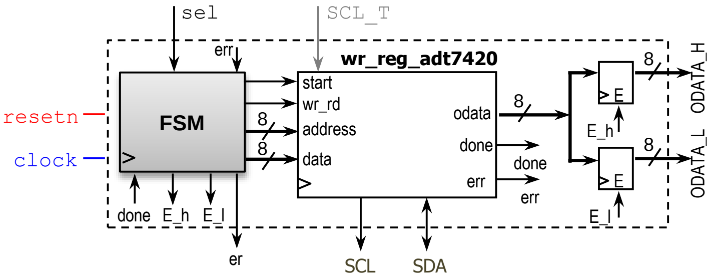
  ✓ Single-byte Write Sequence:

Master	S	AD (7-bit)	W		RA (8-bit)		DATA (8-bit)		P
Slave				ACK		ACK		ACK	

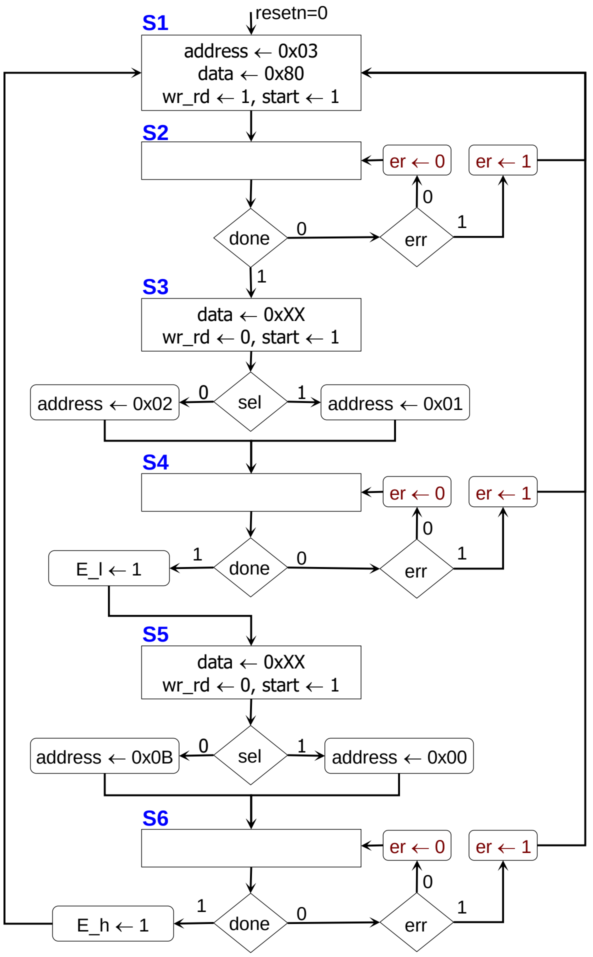
  ✓ Single-byte Read Sequence:

Master	S	AD (7-bit)	W		RA (8-bit)		Sr	AD (7-bit)	R			NACK	P
Slave				ACK		ACK				ACK	DATA (8-bit)		

- The basic controller is depicted on the right. The block **wr_reg_adt7420** is the most important: it handles the I^{2}C communication based on address, data and write/read decision. Asserting the *start* signal initiates a transaction. When the operation is completed, the signal *done* is asserted for one clock cycle. If reading data, it appears on *odata*. A new transaction can be started on the next cycle after $done = 1$.



- **FSM**: It issues commands to configure one ADT7420 register (CONFIG) and then read (cyclically) from two of four 8-bit ADT7420 registers (selected by the sel input). Data is fetched on the output registers. Here, we can read the STATUS and ID registers (0x02, 0x0B), or TEMP_H and TEMP_L (0x01, 0x00).



- **wr_reg_adt7420:** This circuit handles the I^{2}C communication with the ADT7420. The user provides address, data, and read/write. Then, a read/write SPI transaction is executed. At every transaction, we write or retrieve 8 bits of data.

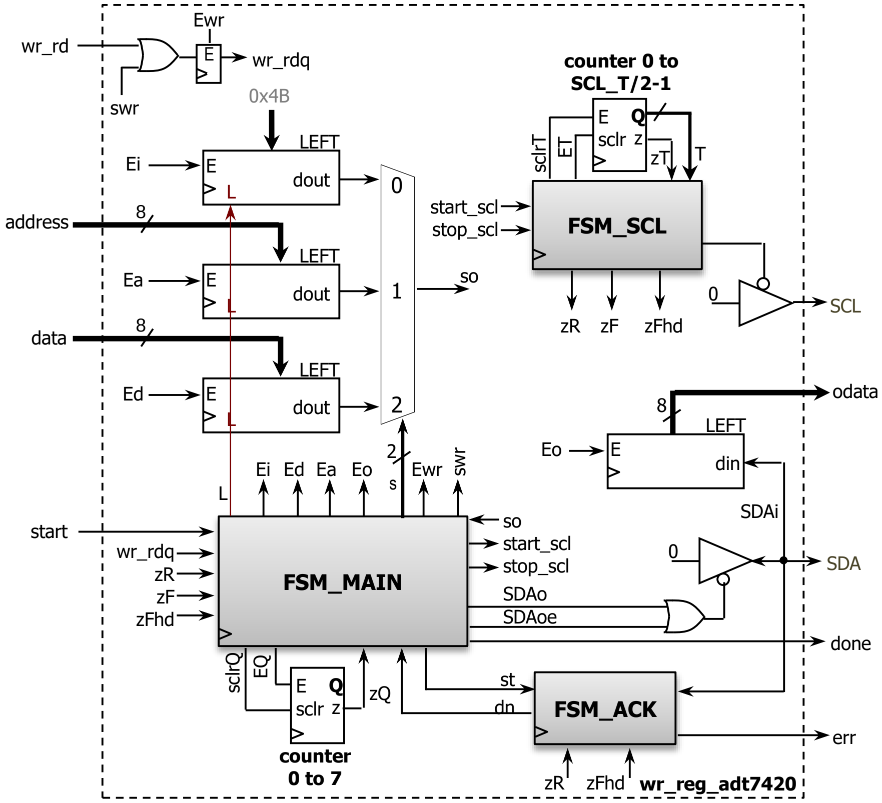
  ✓ Circuit Design: It involves implementing the I^{2}C protocol and satisfying the ADT7420 timing parameters (see datasheet):

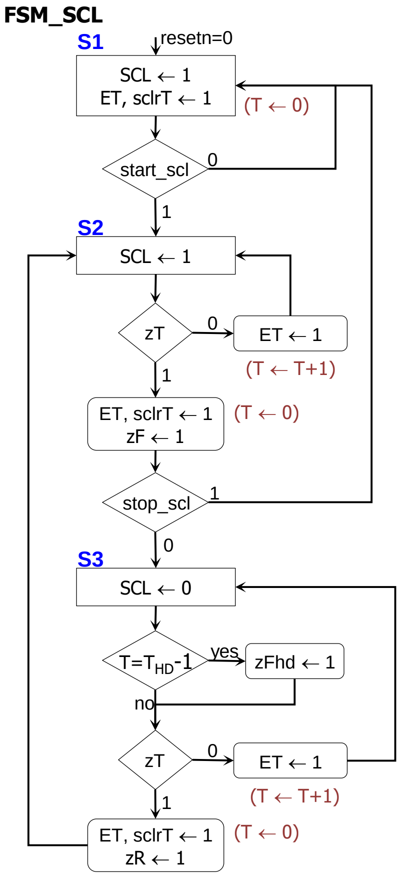
  - $t_{SU:DAT}$ (Data Setup Time): 0.02 us.
  - $t_{HD:DAT}$ (Data Hold Time): 0.03 us.
  - $f_{SCLK} \leq$ 400 KHz.
  - $t_{HD:STA}$ (Hold Time – Start Condition): 0.6 us. Time SCL must be 1 after SDA falling edge.
  - $t_{SU:STA}$ (Setup Time – Start Condition): 0.6 us. Time SCL must be 1 before SDA falling edge.
  - $t_{SU:STO}$ (Setup Time – Stop Condition): 0.6 us. Time SCL must be 1 before SDA rising edge.
  - t_{BUF} (Bus-Free Time ÷ Start and Stop Condition): 1.3 us.

  ✓ For $f_{SCL} \leq$ 400 KHz ($T_{SCL} \geq$ 2.5 us), we have SCL_T ≥ 125 (at clock = 100 MHz).

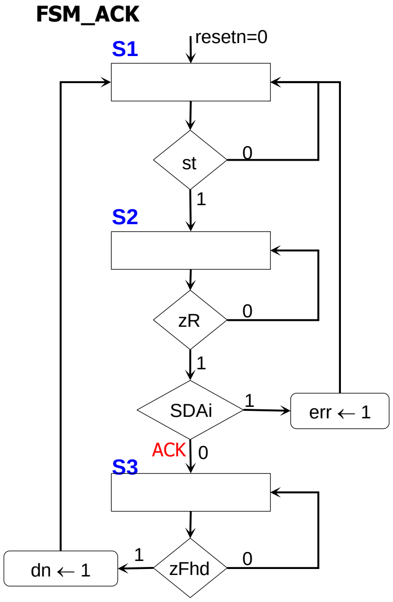
  ✓ To display data on LEDs or 7-segment displays, you need an proper refreshment rate. We pick T_{SCLK}=1 ms (f_{SCLK}=1 KHz) → SCLK_T=50×10^3. There are about 35 SCL periods in a reading transaction, thus data is refreshed at 35 ms per sample.

  ✓ FSM_SCL: It generates a clock of period SCL_T and 50% DC along with rising and falling edge detectors. It also issued a delayed falling edge detection signal zFhd: This is to allow data to be kept for $t_{HD:DAT}$ after the falling edge. The clock stops after the STOP condition (P) is issued.

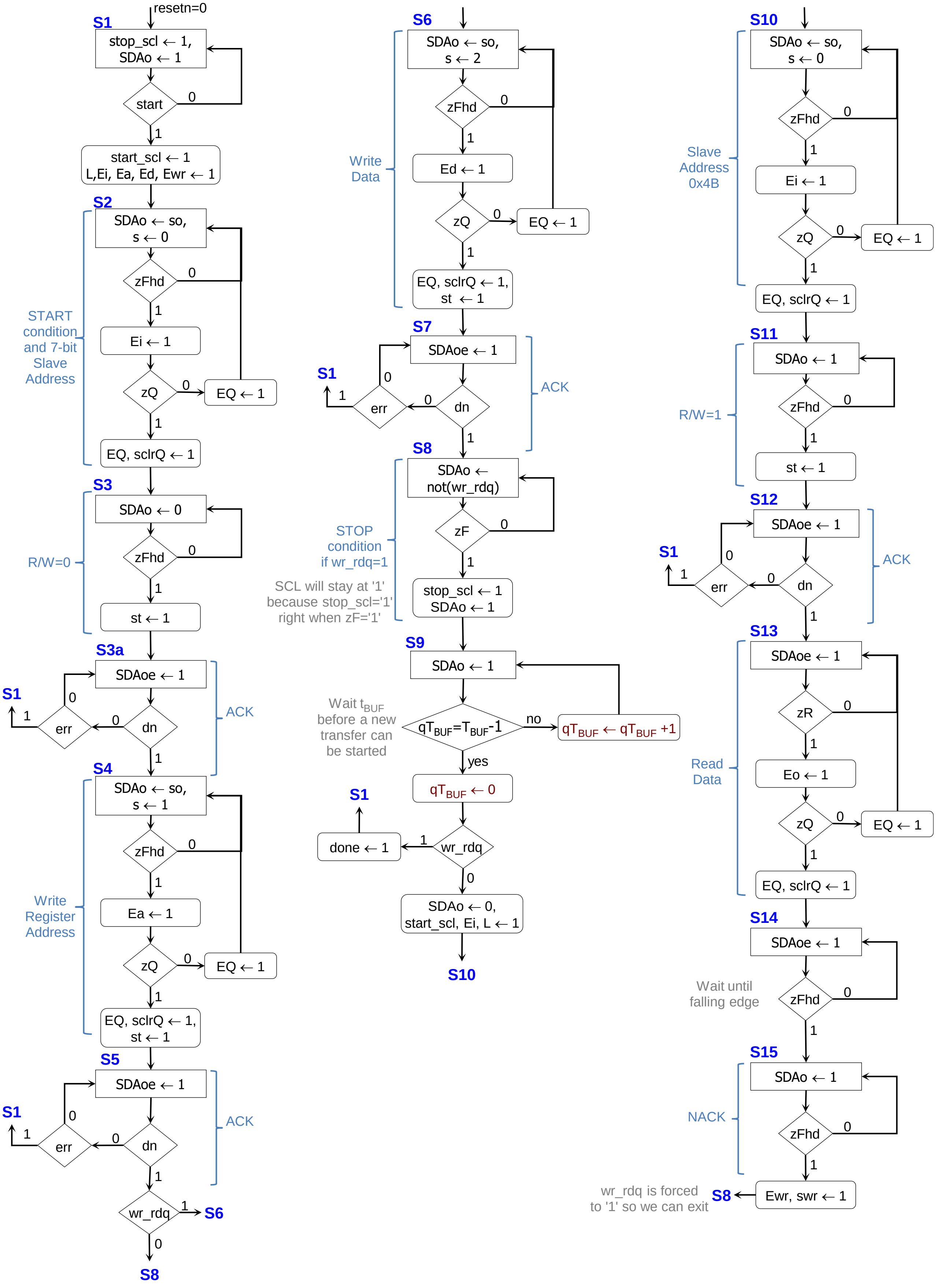




- ✓ **FSM_ACK**: It handles the detection of the Acknowledge bit (ACK), which is generated by the Slave. This operation is repeated at many points in the design, thus we decided to have a separate FSM.



- ✓ **FSM_MAIN**: It handles the I^{2}C communication and complies with the ADT7420 timing parameters. Note that T_{BUF} = 3 (30 ns). Also, data is kept for $t_{HD:DAT}$ after the falling edge of SCL (that is why we have the signal z_{Fhd}, which is issued after $t_{HD:DAT}$). Note that when the Slave is writing data, `SDAoe=1`.

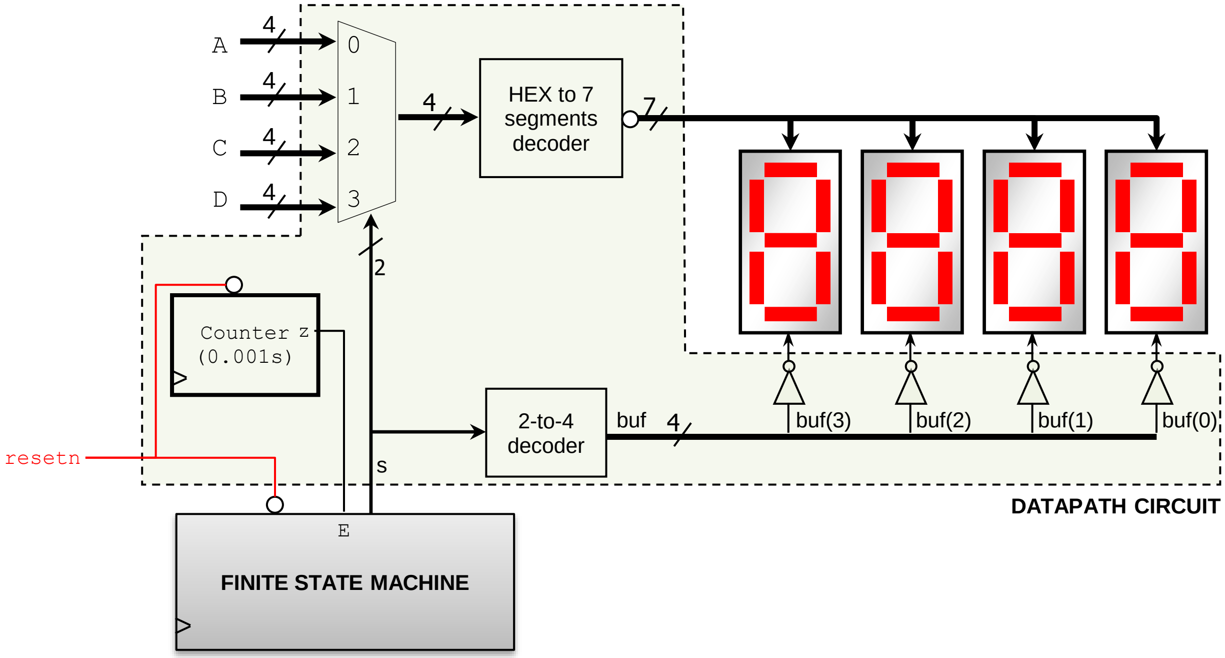


I/O: DISPLAY AND KEYPAD

7-SEGMENT SERIALIZER (VHDL CODE)

DIGITAL SYSTEM (FSM + Datapath circuit)

- Most FPGA Development boards 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 (say 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) digits. 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).

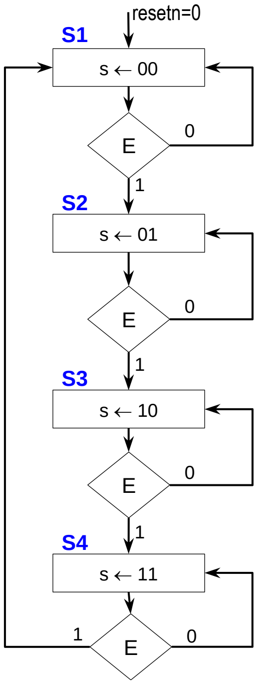


- Generic Component: Behavior on the clock tick.

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

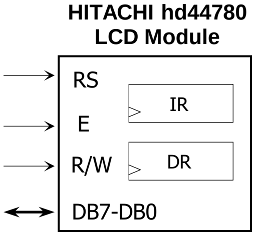
```
if Q = 10^5 - 1 then
      Q ← 0
   else
      Q ← Q+1
  end if;
end if;

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

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



LIQUID CRYSTAL DISPLAY (HD44780 COMPATIBLE)

- LCD modules: LCD and its controller. The Hitachi HD44780 is one of the most popular. It can display one 8-character line or two 8-character lines. For detailed information, refer to the datasheet.
- HD44780: It includes two 8-bit registers:
  - ✓ Instruction Register (IR): It stores instruction codes (e.g.: display clear, cursor shift).
  - ✓ Data Register (DR): It temporarily stores data to be written/read into/from DDRAM (display data RAM) or CGRAM (character generator RAM).
- Digital Interface I/O:
  - ✓ RS: Selects register.
    - ▫ 0: Instruction register (for write). Busy flag (for read).
    - ▫ 1: Data Register (for write and read)
  - ✓ R/W: Selects read (1) or write(0)
  - ✓ E: Enable signals. It starts data read/write cycle.
  - ✓ DB7-DB0: Bidirectional tristate data pins. Used for transfer data and receive between a hardware interface in an FPGA or ASIC (or MPU in a microcontroller) and the HD44780. *There is a 4-bit mode where data is carried on DB7-DB4.



INTERFACING THE HD44780 TO THE FPGA

- IR, DR: Controlled via the interface (RS, E, DB). These signals make up the HD44780 instructions. Instruction categories: i) Designate HD44780 functions, e.g.: display format, data length, ii) Set internal RAM addresses, iii) Perform data transfer with internal RAM, and iv) Perform miscellaneous functions.
- Instruction codes (they apply to IR and DR) are shown below:

Instruction	Code										Description	Execution Time
	RS	R/W	DB$_7$	DB$_6$	DB$_5$	DB$_4$	DB$_3$	DB$_2$	DB$_1$	DB$_0$		
Clear display	0	0	0	0	0	0	0	0	0	1	Clear display, returns cursor to home position (set maddress to 0)	1.64 ms
Cursor home	0	0	0	0	0	0	0	0	1	-	Returns cursor to home position without changing DDRAM contents	1.64 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Set cursor move direction and display shift (during data read/write)	40 us
Display on/off ctrl	0	0	0	0	0	0	1	D	C	B	Sets on/off of display, cursor, and blink of cursor position character.	40 us
Cursor/ display shift	0	0	0	0	0	1	S/C	R/L	-	-	Moves cursor and shifts display without changing DDRAM contents	40 us
Function set	0	0	0	0	1	DL	N	F	-	-	Sets interface data length (DL), display lines (N), character font (F)	40 us
Set CGRAM address	0	0	0	1	CGRAM address						Sets CGRAM address. CGRAM data is sent/received after this setting	40 us
Set DDRAM address	0	0	1	DDRAM address							Set DDRAM address. DDRAM data is sent/received after this setting	40 us
Read busy flag & AC	0	1	BF	CGRAM/DDRAM address							Reads busy flag (BF) and address counter (AC)	0 us
Write data	1	0	Data (to be written)								Writes data into DDRAM or CGRAM	40 us
Read data	1	1	Data (read)								Reads data from DDRAM or CGRAM	40 us

- ✓ Notes:

I/D: 0 (decrement cursor position), 1 (increment cursor position)	S: 0 (no display shift), 1 (display shift)	
S/C: 0 (move cursor), 1 (display shift)	R/L: 0 (shift to the left), 1 (shift to the right)	
D: 0 (display off), 1 (display on)	C: 0 (cursor off), 1 (cursor on)	B: 0 (cursor blink off), 1 (cursor blink on)
DL: 0 (4 bits), 1 (8 bits)	N: 0 (1 line), 1 (2 lines)	F: 0 (5x8 dots), 1 (5x10 dots)
BF: 0 (can accept instructions), 1 (internal operation in progress.		
After execution of CGRAM/DDRAM data write or write instruction, the RAM AC (address counter) is incremented or decremented by 1. The RAM AC is updated after the busy flag (BF) turns off.		

Display Data RAM (DDRAM)

- It stores display data represented in 8-bit character codes. Capacity: 80x8 bits (80 ASCII characters). Only 8 characters per line shown; you can use Display shift (here, 7-bit DDRAM address shifts) to display more characters.
  - ✓ 1-line display (N=0): 8 characters x 1 line.
    DDRAM addresses (80): `0x00, 0x01, …, 0x4F`.

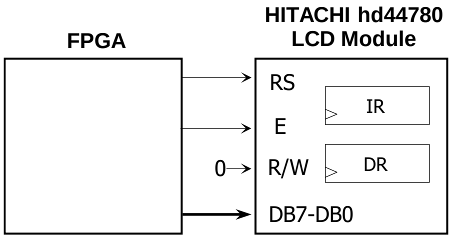
Display position	1	2	3	4	5	6	7	8
DDRAM address	00	01	02	03	04	05	06	07
shift left	01	02	03	04	05	06	07	08
shift right	4F	00	01	02	03	04	05	06

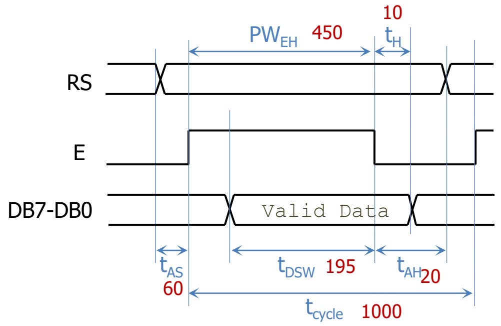
  - ✓ 2-line display (N=1): 8 characters x 2 lines.
    DDRAM addresses (80): 1st line: `0x00, 0x01, …, 0x27`.
    2nd line: `0x40, 0x41, …, 0x67`.

Display position	1	2	3	4	5	6	7	8
DDRAM address	00	01	02	03	04	05	06	07
	40	41	42	43	44	45	46	47
shift left	01	02	03	04	05	06	07	08
	41	42	43	44	45	46	47	48
shift right	27	00	01	02	03	04	05	06
	67	40	41	42	43	44	45	46

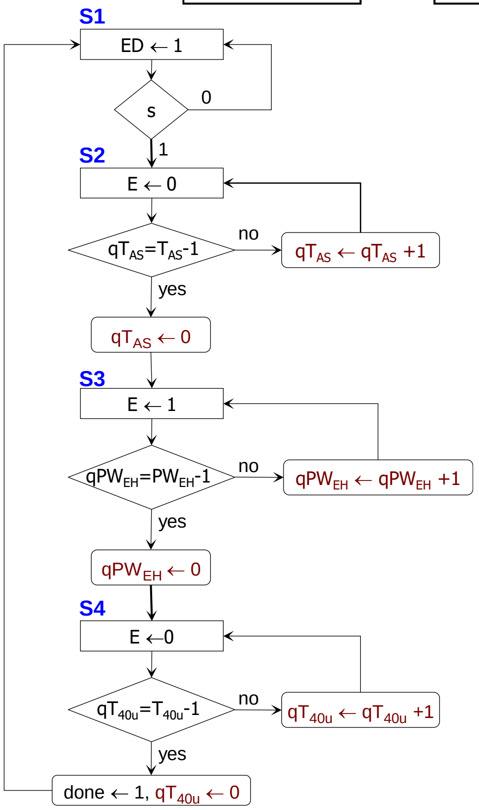
Simplified interface (R/W = 0)

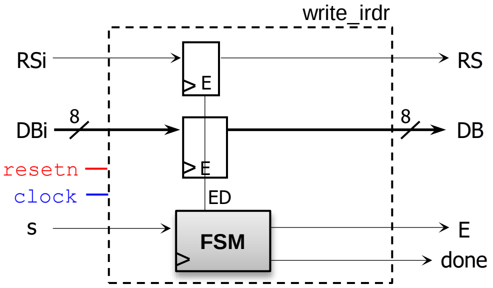
- With R/W=0, we only write on IR and DR. Data transfer occurs in one direction.
- The write timing diagram is depicted. Certain timing parameters (specified in ns) must be satisfied for successful LCD operation.
- `write_irdr`: It reads input data (8-bit DBi, RSi: write on DR or IR), and issues the signals DB, RS, E while meeting the timing parameters. Note: when E goes back to 0, we wait 40 us (not T_{cycle} - PW_{EH} +t_H as per the timing diagram). This ensures instructions are completed without having to read the BF signal.





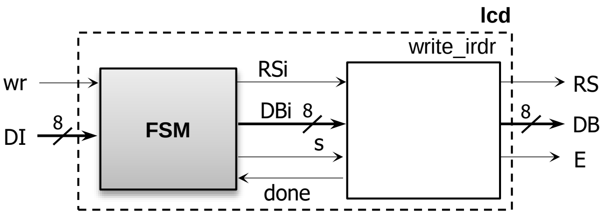
T_{AS}=6 (for T=10ns)

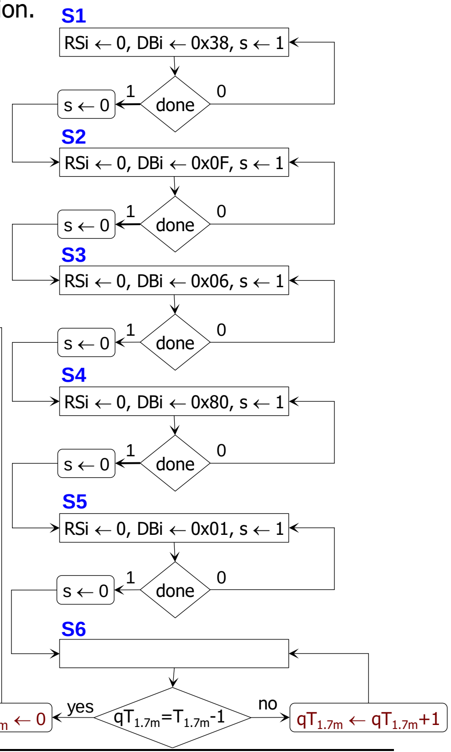


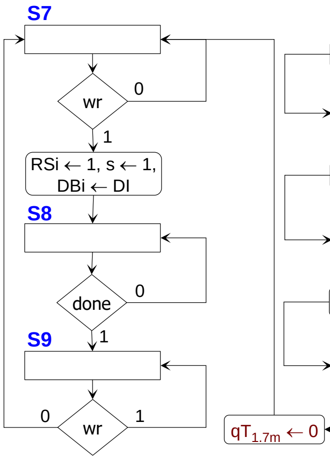


- Based on '`write_irdr`', we implement a basic LCD interface block that initializes the LCD (write set of instructions on IR), and then writes data characters (ASCII, we only write on the DDRAM). This is usually the job of a microprocessor.
  - ✓ Note: 'Clear display' and 'Cursor Home' are the only two instructions that take 1.64 ms. We must account for this delay in the LCD interface. The table shows a basic set of instructions for LCD configuration.

	Instruction Description	RS	DB
Initial LCD Config.	Function set: 8-bit operation, 2-line display, 5x8 character font	0	0x38
	Display on/off control (turns on display and cursor blink)	0	0x0F
	Entry mode set: address inc., cursor shift right, no display shift	0	0x06
	Set DDRAM address: set address to 0 (write to DDRAM after)	0	0x80
	Clear Display (wait 1.64 ms)	0	0x01
Data write	Write data (this can be continuously):	1	8-bit
	Move to 2nd line (set DDRAM address to 0xC0)	0	0xC0
	Move to 1nd line (set DDRAM address to 0x80)	0	0x80



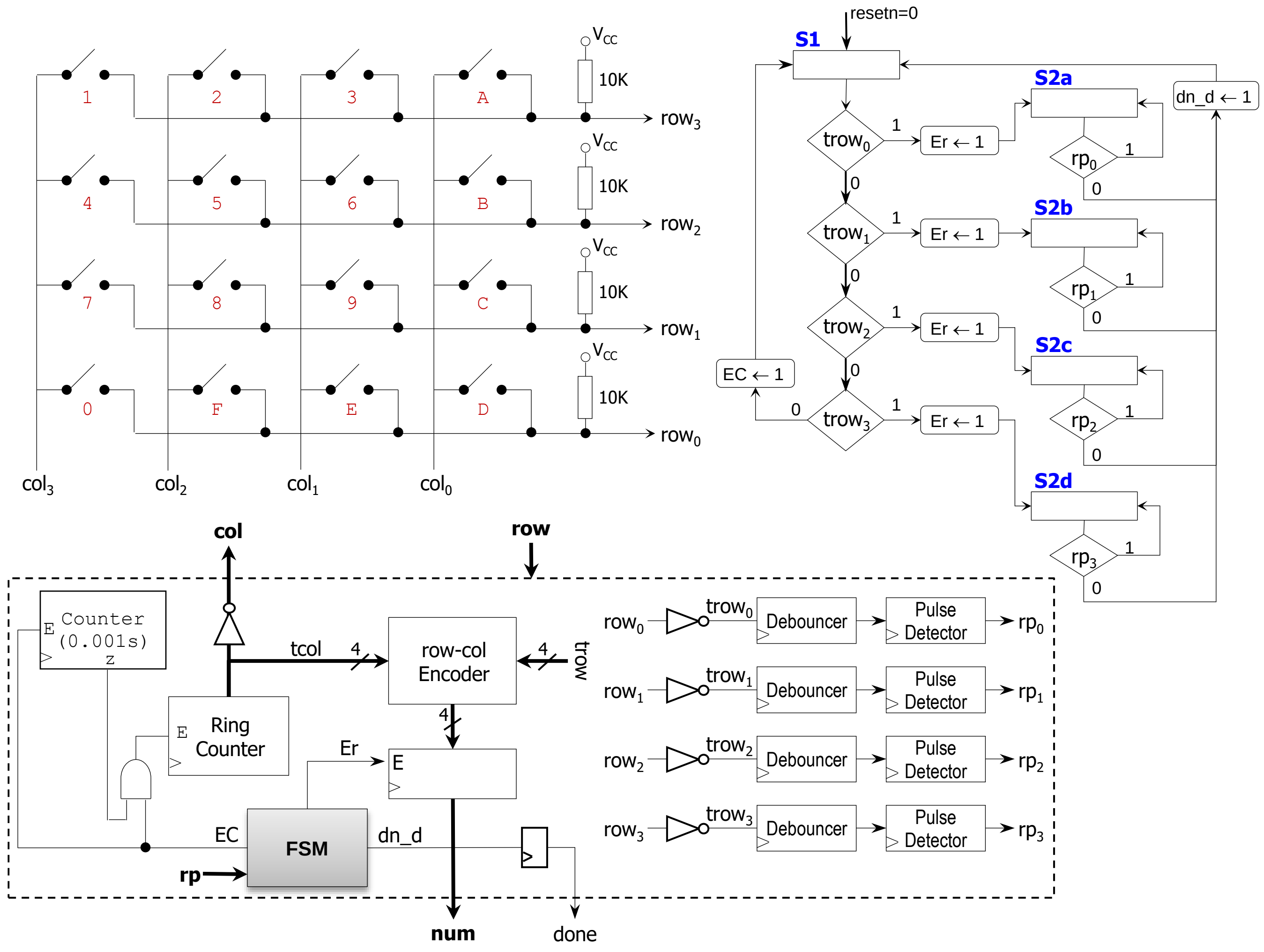

S1
RSi ← 0, DBi ← 0x38, s ← 1
s ← 0
1
done
0
S2
RSi ← 0, DBi ← 0x0F, s ← 1
s ← 0
1
done
0
S3
RSi ← 0, DBi ← 0x06, s ← 1
s ← 0
1
done
0
S4
RSi ← 0, DBi ← 0x80, s ← 1
s ← 0
1
done
0
S5
RSi ← 0, DBi ← 0x01, s ← 1
s ← 0
1
done
0
S6
yes
qT_1.7m=T_1.7m-1
no
qT_1.7m ← qT_1.7m+1



16-BUTTON KEYPAD

- To read data from 16 push buttons in a keypad, a straightforward approach is to use 16 wires.
- A more efficient approach is to arrange the 16 buttons into a 4x4 array. This requires only 8 wires, and a scanning method is required to read data (we write on 4 wired, and we read from 4 wires). This is a more optimal approach ($2N$ vs N^2).
- **Scanning Method**:
  - ✓ 4-bit **col** signal: We only activate one wire at a time (for 1 ms). Then, we read the 4-bit **row** signal, and detect which one is activated (due to the configuration, at most only one row is activated). By detecting which row and col were activated, we can determine which button the user pressed.
  - ✓ Mechanical bouncing: This requires a debouncer on every **row** signal. However, this introduces a delay (longer than 1 ms), so we will most likely lose the detection of an activated **row** signal, as the **col** signal will have changed.
    - ▫ Solution: the moment we detect a **row** signal that is activated (at that moment, we also capture the resulting button value), we freeze the 4-bit **col** signal. We then wait until the user releases the push button to continue with the updating of the **col** signal.

- **Digital System**: The figure depicts the digital system (FSM + Datapath)
  - ✓ The wiring interconnection of the push buttons causes the 4-bit signals **row** and **col** to be active low.
  - ✓ Ring Counter: it activates only one bit of the signal **col** every 1 ms (this is controlled by a 1 ms counter)
  - ✓ Each of the 4 bits of **row** is debounced and a one-cycle pulse is issued (to indicate the user released the button).
  - ✓ FSM: As soon as one **row** signal becomes active, we go into a state where we wait until the corresponding one-cycle pulse is issued.
  - ✓ **row**-**col** encoder: This combinational circuit generates the value of the button that was pressed based on the current **col** value and the **row** value.
  - ✓ If a bit of the row signal becomes activated at the same moment the 1 ms counter issues its pulse (rare occurrence), then the ring counter will work (at the operating frequency) regardless of whether we stop the 1 ms counter. To avoid this, the AND gate ensures that when $\mathrm{EC}=0$, both the 1 ms and the Ring Counter are frozen.
  - ✓ The output data (**num**) is issued with a one-cycle 'done' signal.



PULSE-WIDTH MODULATION (PWM)

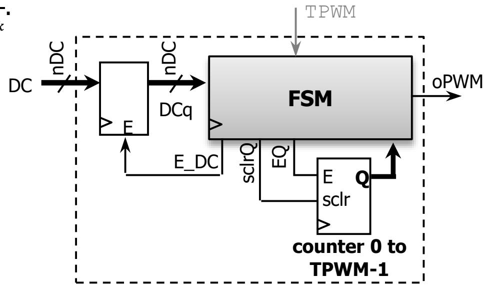
DEFINITION

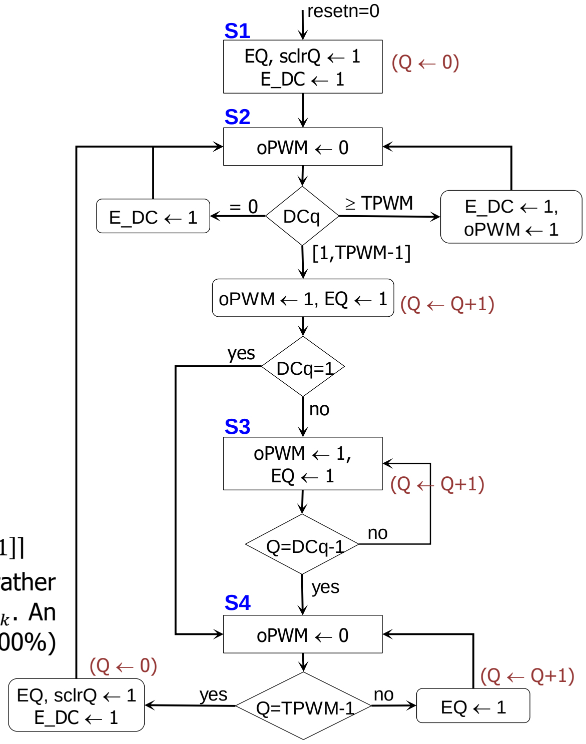
- We generate a square wave where we control the Duty Cycle. Duty Cycle is specified as a percentage: from 0 to 100%.
- PWM can be used to vary the average voltage on an output pin. This can be useful (in lieu of a DAC) to control the brightness of an LED, speed of a DC motor, volume of a tone in a speaker, etc.

DIGITAL SYSTEM FOR PWM (code available here)

- $TPWM$ (Period of PWM signal in units of T_{clock}): This is a parameter in the VHDL code. $TPWM > 2$

  $T_{PWM} = TPWM * \frac{1}{f_{clock}}$.





$$nDC = \lceil \log_2(TPWM+1) \rceil$$

$$If\ DC = 0 \rightarrow oPWM = 0$$

$$If\ DC = TPWM \rightarrow oPWM = 1$$

For $f_{clock} = 100\ MHz$:
$TPWM = 500 \rightarrow f_{PWM} = 200\ KHz$
$TPWM = 50000 \rightarrow f_{PWM} = 2\ KHz$

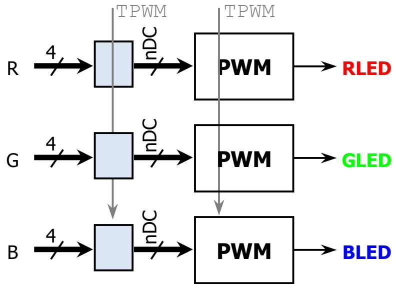
- DC (Duty Cycle): Input signal with nDC bits. $nDC = \lceil \log_2[TPWM+1] \rceil$

  DC ∈ [0, TPWM]. Note that DC is not specified from 0 to 100%, but rather from 0 to TPWM. Note that the "step" of the DC depends on f_{clock}. An external circuit can retrieve the Duty Cycle in standard terms (0-100%) and convert it to 0 to TPWM.

TRI-COLOR LEDs

RGB color can be controlled by varying (via PWM) the brightness of a Red, Green, and Blue LEDs. We want to control the DC of each color component using NB=4 bits. So, we need to map a signal from 0 to 2^{NB}-1 to a signal from 0 to TPWM. Mapping formula:

$$DC(0 \rightarrow TWPM) = \left\lfloor \frac{TPWM}{2^{NB}-1} \times DC(0 \rightarrow 2^{NB}-1) \right\rfloor \approx \left\lfloor \frac{TPWM \times DC(0 \rightarrow 2^{NB}-1)}{2^{NB}} \right\rfloor$$

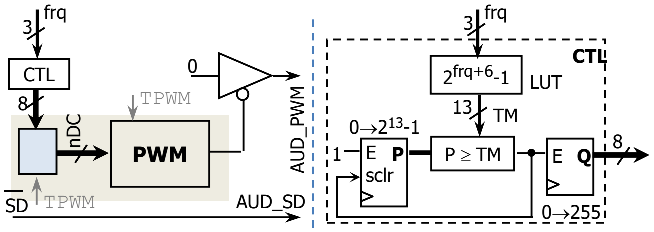


DIGITAL CIRCUIT (code available here)

- **Mapping circuit**: The approx. formula optimizes hardware: we multiply and then drop NB LSBs. DC (0-TPWM) never reaches TPWM, but the approx. is good enough.
- PWM frequency: 2 KHz (TPWM=50000, $f_{clock} = 100\ MHz$) provides a good color variation. A high frequency breaks the linearity between the brightness and the DC.
- We can use more bits per color component, but we need more input signals. For NB=4, refer to hex tables (higher nibble).

MONO AUDIO OUTPUT

- Nexys-4 (DDR) Board: An analog low pass filter (connected after AUD_PWM) turns a PWM signal with varying DC (DC goes from 0 to 100% and back) into a sinusoid. Use NB=8 bits.



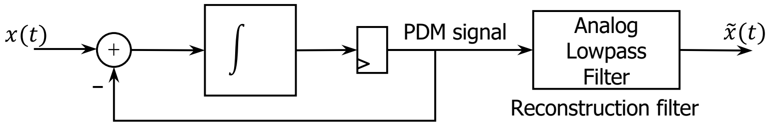
DIGITAL CIRCUIT (code available here)

- Shaded circuit: It generates a square wave and it can be connected to a buzzer or speaker, though we can only vary DC (≡ volume). Only frequency can change the tone, i.e., we need a new circuit where TWPM is an input signal.
- CTL: It produces a varying 8-bit DC (0→255→0, …). This allows the integrator to generate a sinusoidal wave. The variation rate is controlled by `frq`, i.e., we can pick from 8 sinusoidal frequencies.
- AUD_PWM: Open-drain output. AUD_SD: Analog filter shutdown input (via the AD8592 opamps). TPWM = 1000 (100 KHz).

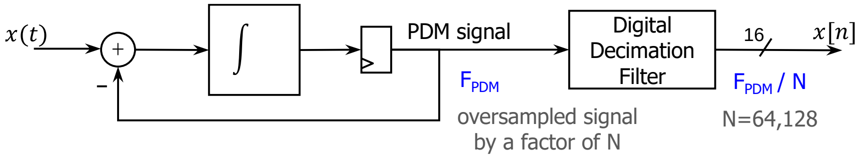
PULSE DENSITY MODULATION (PDM)

DEFINITION

- Popular in mobile devices, only 1 bit is required. 1-bit signal is oversampled.
- The amplitude of a signal is represented by the relative density of the pulses: the closer the pulses are, the larger the amplitude. Unlike PWM, the frequency of the pulses is not fixed.
- A PDM signal can be generated from an analog signal by using a sigma-delta modulator.
- Once PDM data is obtained, the analog signal can be recovered by passing the signal through an analog low-pass filter:



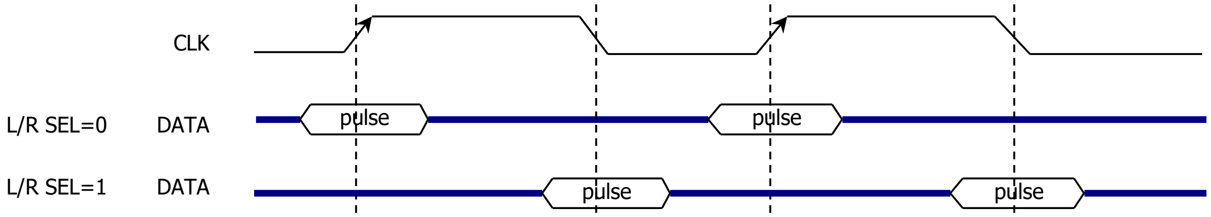
- If we want to get the PCM (pulse-code modulation)-coded signal to apply digital signal processing operations, we require a digital decimation filter. The figure depicts a PDM signal oversampled by a factor of N (over the Nyquist rate). The decimation filter outputs a signal $x[n]$ (16 bits per sample) sampled at the Nyquist rate. To recover the analog signal from $x[n]$, a DAC (digital-to-analog converter) is required.



MICROPHONE

- **ADMP421:** MEMS Microphone with PDM output
  - ✓ CLK: 1 – 3 MHz. Recommended: 2.4 MHz.
  - ✓ DATA: PDM signal (oversampled data)
  - ✓ L/R: Left right stereo input control. L/R=0: Data captured on CLK rising edge. L/R=1: Data captured on CLK falling edge.
  - ✓ Many MEMS microphones (e.g.: ADMP521, MP34DT02) feature a similar synchronous interface.

- **ADMP421:** Synchronous interface. Make sure to comply with the timing parameters (see ADMP421 datasheet).



- Once PDM data is obtained, the audio signal can be played back by passing the signal through an analog low-pass filter.

AUDIO OUT

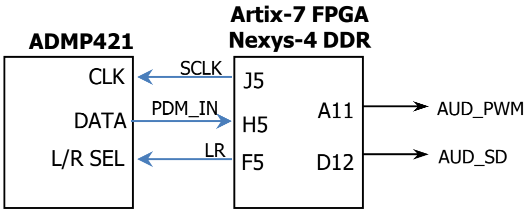
- Nexys-4/Nexys-4 DDR: The on-board audio jack is driven by an analog low-pass filter. The input then can be a PDM or PWM signal. The cut-off frequency is about 12 KHz. Stereo output is not supported.
- AUD_PWM: Open-drain output. AUD_SD: Analog filter shutdown input (via the AD8592 opamps).

AUDIO CAPTURE AND PLAYBACK ON THE NEXYS-4 DDR BOARD

- The figure depicts the connection between the MEMS microphone, the Artix-7 FPGA, and the mono audio output.

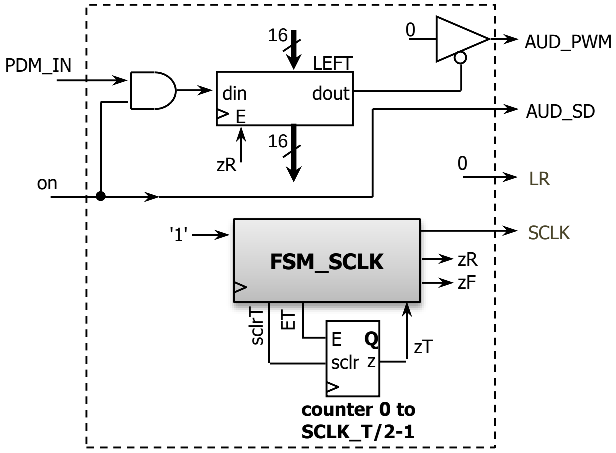
DIGITAL CIRCUIT

- As stereo output is not supported, we only retrieve a mono audio input from the ADMP42 microphone (e.g. L/R = 0).
- Main frequency (Nexys-4 DDR Board): $f_{clock} = 100\,MHz$, $T_{clock} = 10\,ns$.



Basic approach

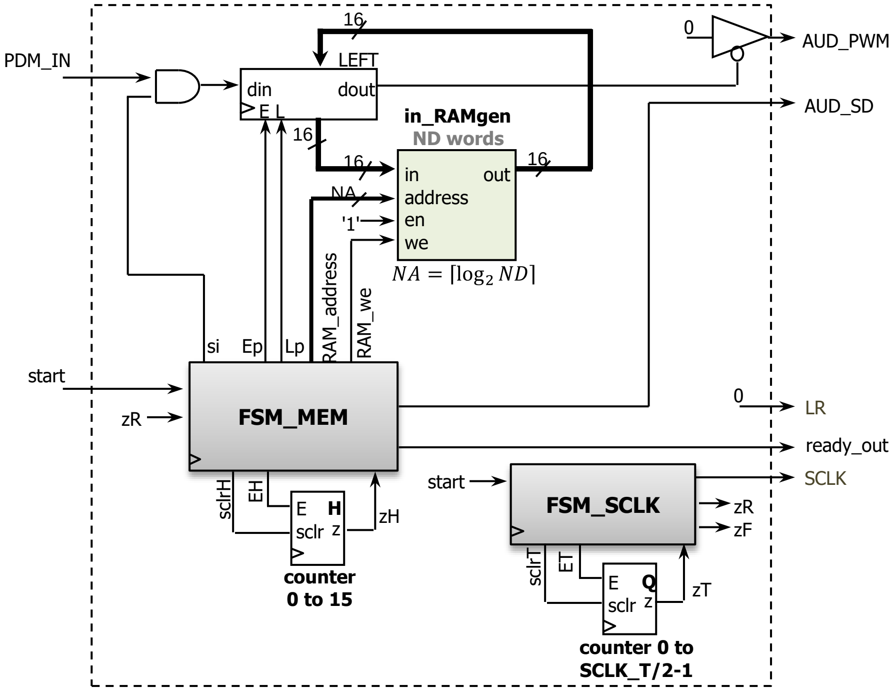
- The figure depicts a simple circuit that reads data in a shift register and immediately outputs the data. The rate at which data is shifted in and out is given by SCLK. Be aware of feedback when using this circuit.
- FSM_SCLK: It generates a free running clock of period `SCLK_T` and 50% DC along with rising and falling edge detectors. With an input clock of 100 MHz, we have that:
  - ✓ For SCLK = 1 MHz → $SCLK\_T = \frac{1}{1\times10^{6}}\frac{1}{10\times10^{-9}} = 100.$
  - ✓ For SCLK = 3 MHz → $SCLK\_T = \frac{1}{3\times10^{6}}\frac{1}{10\times10^{-9}} \approx 34.$
  - ✓ For SCLK = 2.4 MHz → $SCLK\_T = \frac{1}{2.4\times10^{6}}\frac{1}{10\times10^{-9}} \approx 42.$

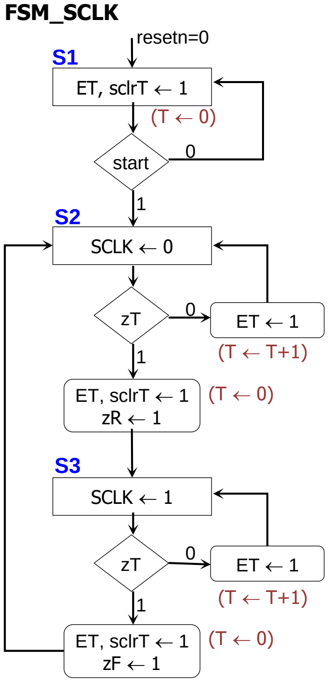


Memory-based approach

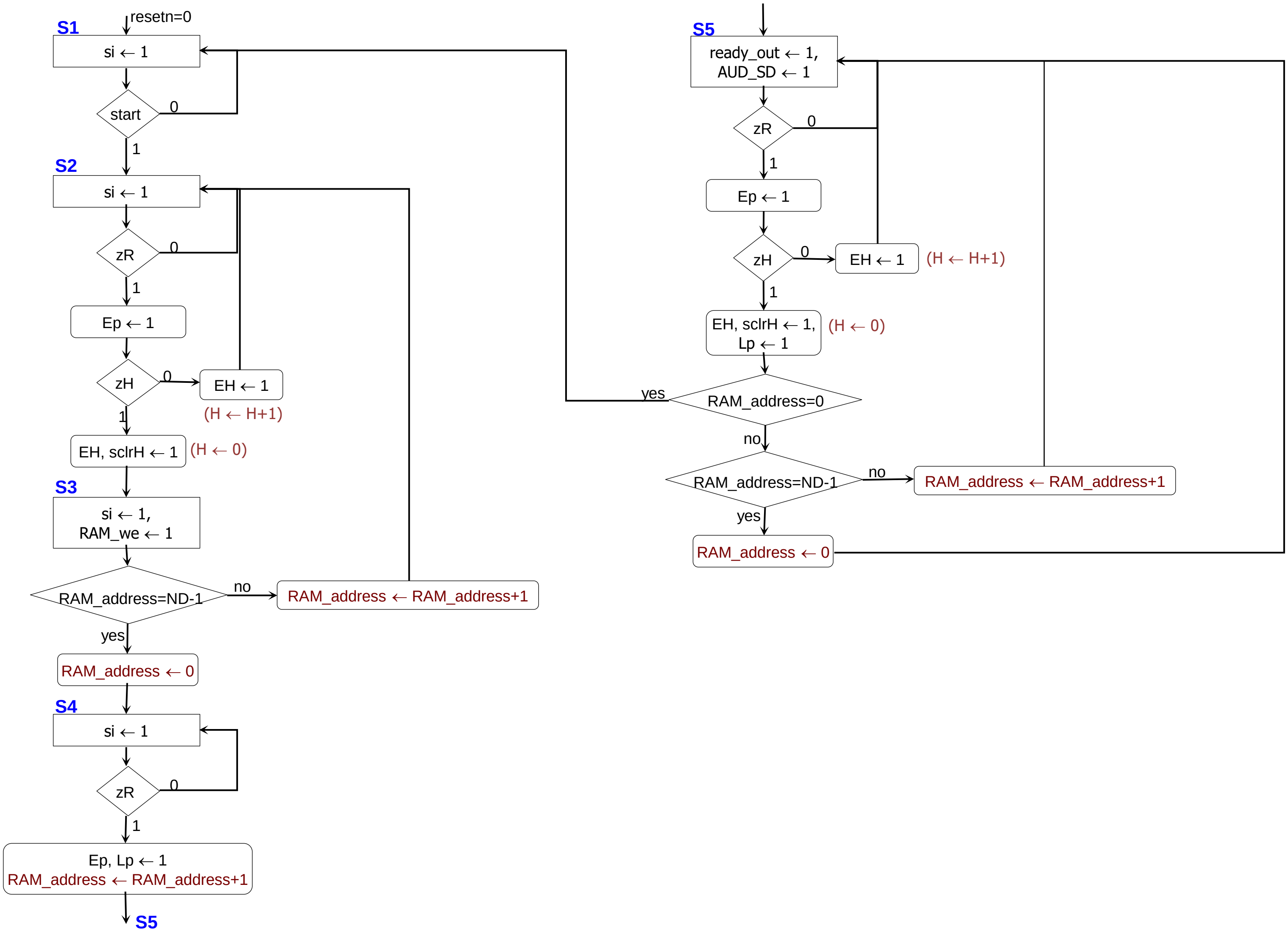
- Here, data is read into the shift register and then stored it in memory. We can then control when we shift data out. We might also store several audio sequences and select when to play them. Data is shifted in and out at the rate given by SCLK.
- Memory: It can store up to ND 16-bit words.
  - ✓ Address size: $NA = \lceil \log_2 ND \rceil$.
  - ✓ Total number of bits: $ND \times 16$ bits.
  - ✓ Duration of the stored sequence: $ND \times 16 \times SCLK\_T \times T_{clock}$. For example if $ND = 2^{18}$ and `SCLK_T=42` we have 1.7616s. To increase the duration, we can increase `SCLK_T` (`SCLK_T ≤ 100`), or we can increase the memory size.
- The main control circuit (FSM_MEM) varies according to the type of memory used. The memory might not operate at the same frequency or might have different input/output ports than the ones shown. For example:
  - ✓ On-chip memory (BlockRAMs inside Artix-7 FPGAs): Simple to use (we will use this one).
    - ▫ BlockRAMs operate at the same frequency (100 MHz).
    - ▫ BlockRAM I/O ports: specified in the figure.
    - ▫ A BlockRAM behaves as a collection of registers: data requested/written is available on the next clock cycle. But the capacity is limited (~ 0.5 MB in the XCA100T Artix-7 FPGA).
  - ✓ External memories (e.g.: DDR2 RAM, Flash, SRAM): They require a different I/O interface and operating frequency; however, they can hold much more data.

- The circuit requires a 16-bit shift register, a memory, and two state machines. The FSM_SCLK is also depicted.





- **FSM_MEM** (using BlockRAMs inside Artix-7 FPGAs): Note how we embed the counter for `RAM_address` inside the FSM, as well as counter `H` (this is optional).


resetn=0
S1
si ← 1
start
0
1
S2
si ← 1
zR
0
1
Ep ← 1
zH
0
EH ← 1
(H ← H+1)
1
EH, sclrH ← 1
(H ← 0)
S3
si ← 1,
RAM_we ← 1
RAM_address=ND-1
no
RAM_address ← RAM_address+1
yes
RAM_address ← 0
S4
si ← 1
zR
0
1
Ep, Lp ← 1
RAM_address ← RAM_address+1
S5
S5
ready_out ← 1,
AUD_SD ← 1
zR
0
1
Ep ← 1
zH
0
EH ← 1
(H ← H+1)
1
EH, sclrH ← 1,
Lp ← 1
(H ← 0)
yes
RAM_address=0
no
RAM_address=ND-1
no
RAM_address ← RAM_address+1
yes
RAM_address ← 0