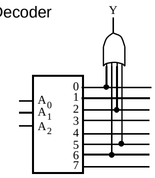
1 RECAP

• Decoder



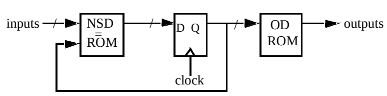
A_2	A_1	A_0	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

• ROM – compile time look-up table

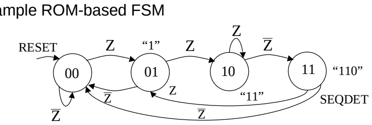
• RAM – programmable LUT → level sensitive and tri-state

2 ROM-BASED FSM

Why? – Eliminate random logic, which is had to maintain

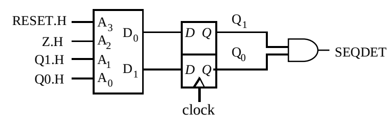


1. Example ROM-based FSM



2 ROM-BASED FSM (cont.)

2. State table – block diagram



Reset A3	Z A2	Q_1 A1	Q_0 A0	Next State Q_1=D_1	Q_0=D_0	Output
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	1	0
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	1	0	0
0	1	1	1	0	1	1
1	X	X	X	0	0	$Q_1 \cdot Q_0$

2 ROM-BASED FSM (cont.)

Address	Data	Address	Data
0	0	8	0
1	0	9	0
2	3	A	0
3	0	B	0
4	1	C	0
5	2	D	0
6	2	E	0
7	1	F	0

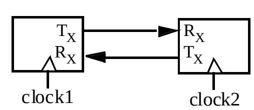
reset

Note: No logic to draw – just type in numbers

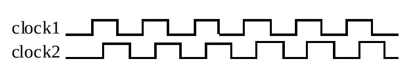
Xilinx ROM: Values written from 15 → 0, not 0 → 15 (in Hex)

3 SERIAL DATA TRANSMISSION WITH MULTIPLE INDEPENDENT CLOCKS

Minimum number of wires – 1 signal and ground or 1 optical

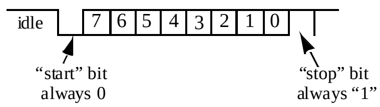


asynchronous clock frequency ~±5%



3 SERIAL DATA TRANSMISSION (cont.)

Serial data



Transmitter: easy – 10-bit SR (start + data + stop) and 10-state counter

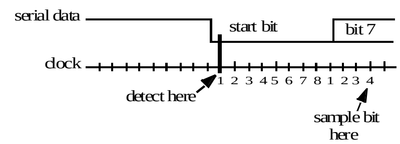
minimum 1-stop bit, no maximum number (idle ok)

successor to baudot code 5 bit electromechanical (75 baud ~1910)

3 SERIAL DATA TRANSMISSION (cont.)

3.1 Serial (Asynchronous) Receiver

Basic idea: Oversample to find start bit



Issues: 1. wait for start bit

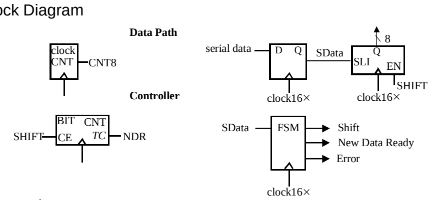
2. wait for middle of bit since frequency may be off

3. input data asynchronously

4. error detection: missing stop or false start

3 SERIAL DATA TRANSMISSION (cont.)

Block Diagram



State Diagram