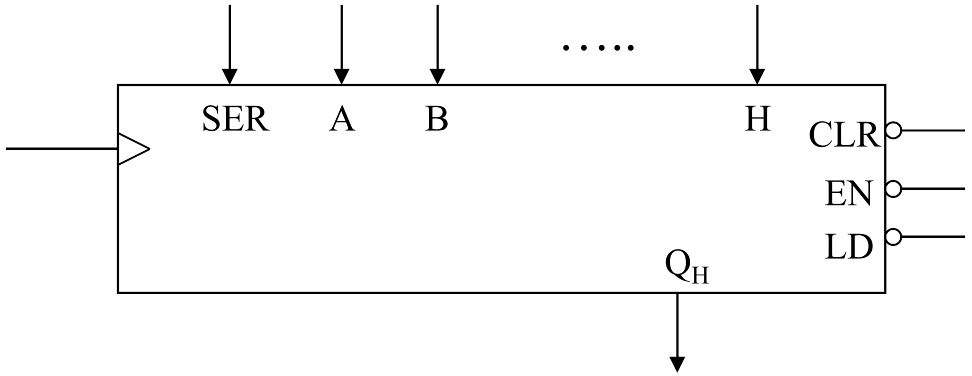
# Chapter 9.
# Sequential Circuit Design Practice

# Example 1: Problem Statement

- **Design 74x166 – 8 bit parallel-in, serial-out shift register with enable**



CLR – Asynchronous

EN – when asserted -> according to LD

otherwise -> hold

LD – when asserted -> LD

otherwise -> Shift

# Step 1: State/Output Table

- **A state table (or diagram) isn't very helpful since LD can take you from any state to any other -> messy!**

|  |  |  |  |
|---|---|---|---|
| – Inputs: | A-H | 8 | $2^{20}$ rows! |
|  | SER | 1 |  |
|  | CLR | 1 |  |
|  | EN | 1 |  |
|  | LD | 1 |  |
| State variables: | Qs | 8 |  |

# Alternatives

- **CLR asynchronous – not needed**
- **EN – take care in special way**
  - initially assume always asserted
- **Think 2 bits rather than 8 bits and then generalize**
- **Variables: SER, A, B, LD, $Q_A$, $Q_B$ → still $2^6$=64 rows!**
  **→ "Variable Entered Table"**

# Variable-Entered Table

| $Q_A$ | $Q_B$ | LD | $Q_A$ | $Q_B$ |
|---|---|---|---|---|
| 0 | 0 | 0 | SER | 0 |
| 0 | 0 | 1 | A | B |
| 0 | 1 | 0 | SER | 0 |
| 0 | 1 | 1 | A | B |
| 1 | 0 | 0 | SER | 1 |
| 1 | 0 | 1 | A | B |
| 1 | 1 | 0 | SER | 1 |
| 1 | 1 | 1 | A | B |

# Steps 2-6

- **Step 2: state minimization – not relevant**
- **Step 3: state assignment – not relevant**
- **Step 4: Transit/output table – we already have**
- **Step 5: Choose f/f – D f/f**
- **Step 6: Excitation table – same as transit table**

# Steps 7-8: Excitation/Output Eqs. (Variable Entered Map)

$D_A$

| $Q_AQ_B$ \ LD | 0 | 1 |
|---|---|---|
| 00 | SER | A |
| 01 | SER | A |
| 11 | SER | A |
| 10 | SER | A |

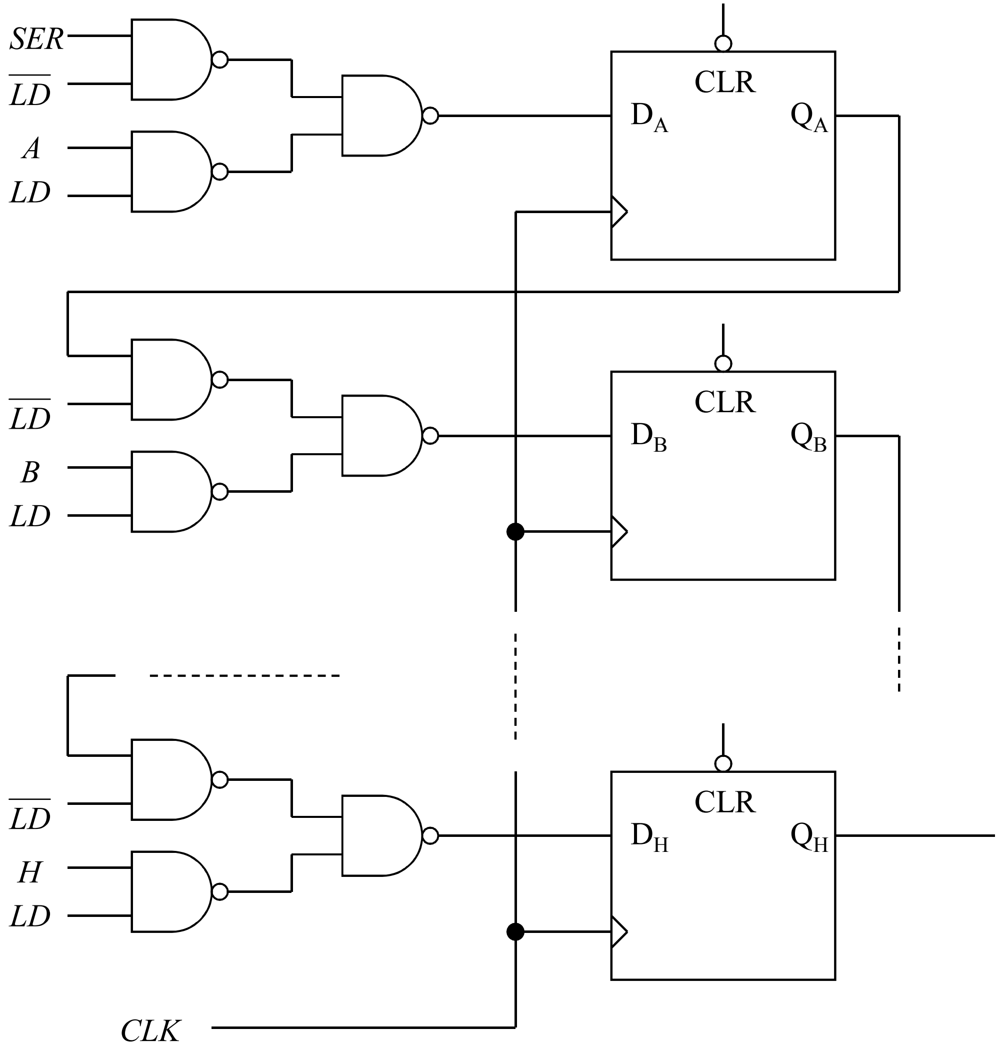
$$D_A = \overline{LD} \cdot SER + LD \cdot A$$

$D_B$

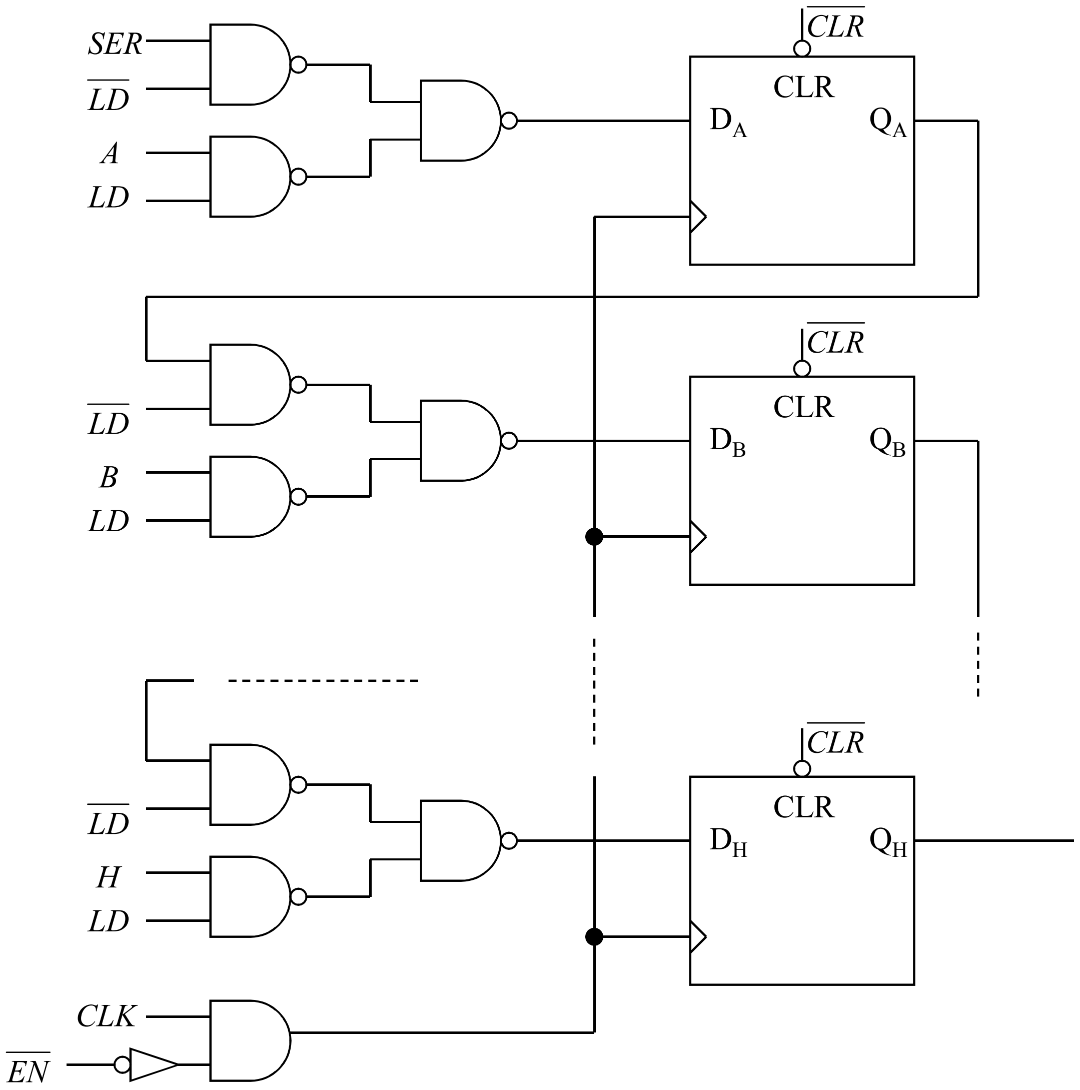
| $Q_AQ_B$ \ LD | 0 | 1 |
|---|---|---|
| 00 | 0 | B |
| 01 | 0 | B |
| 11 | 1 | B |
| 10 | 1 | B |

$$D_B = \overline{LD} \cdot Q_A + LD \cdot B$$

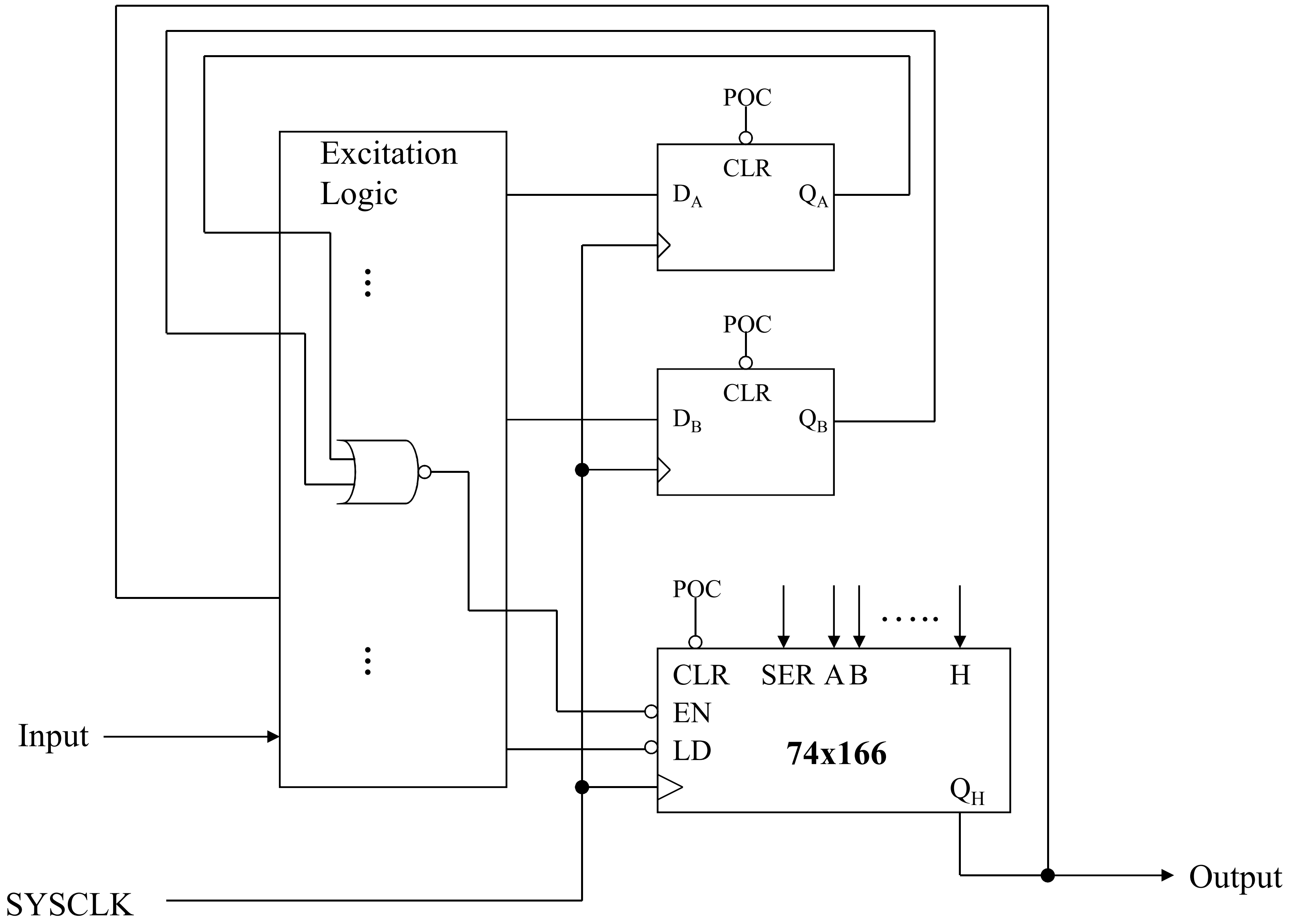
# Steps 9: Logic Diagram



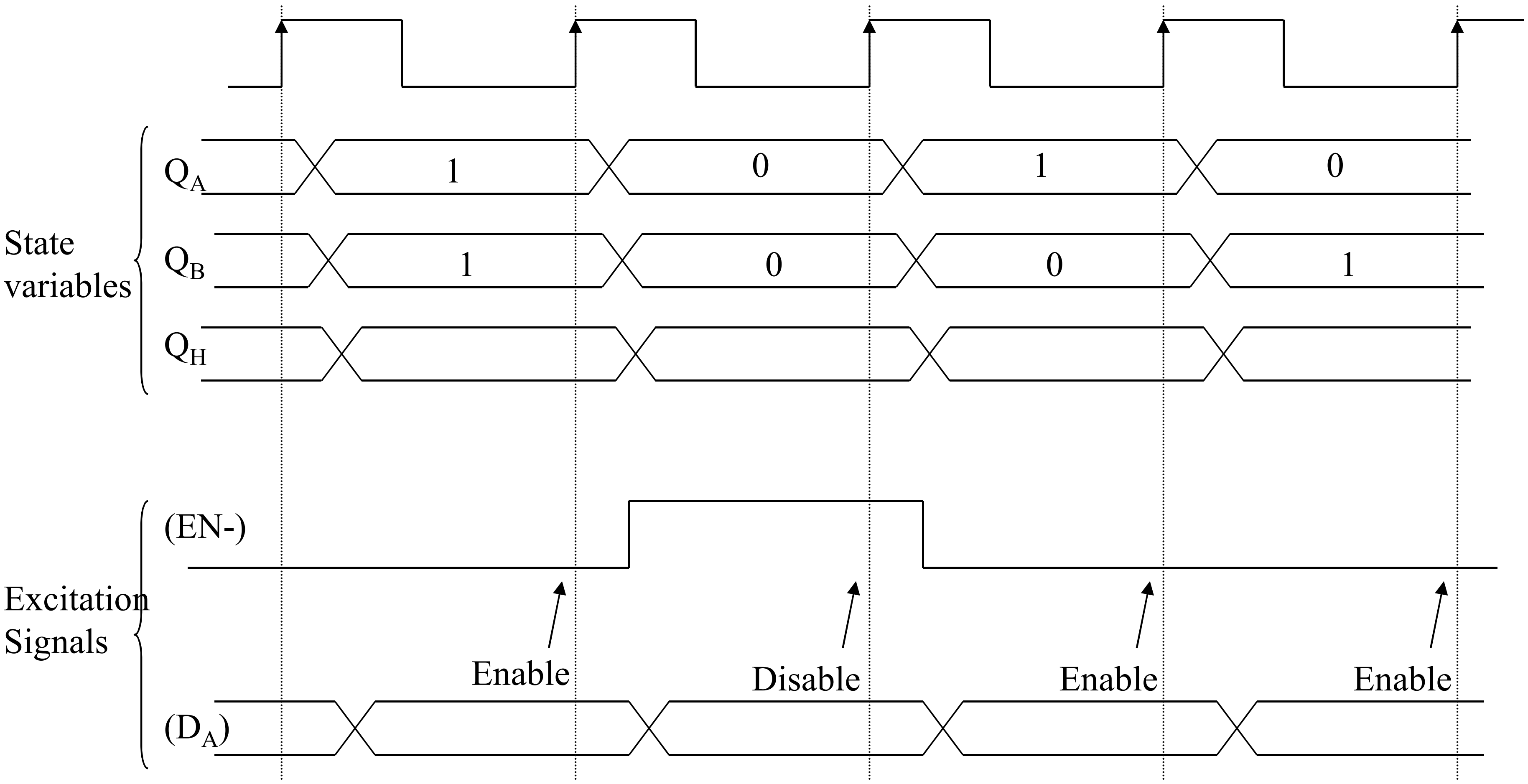
# Steps 9: Logic Diagram



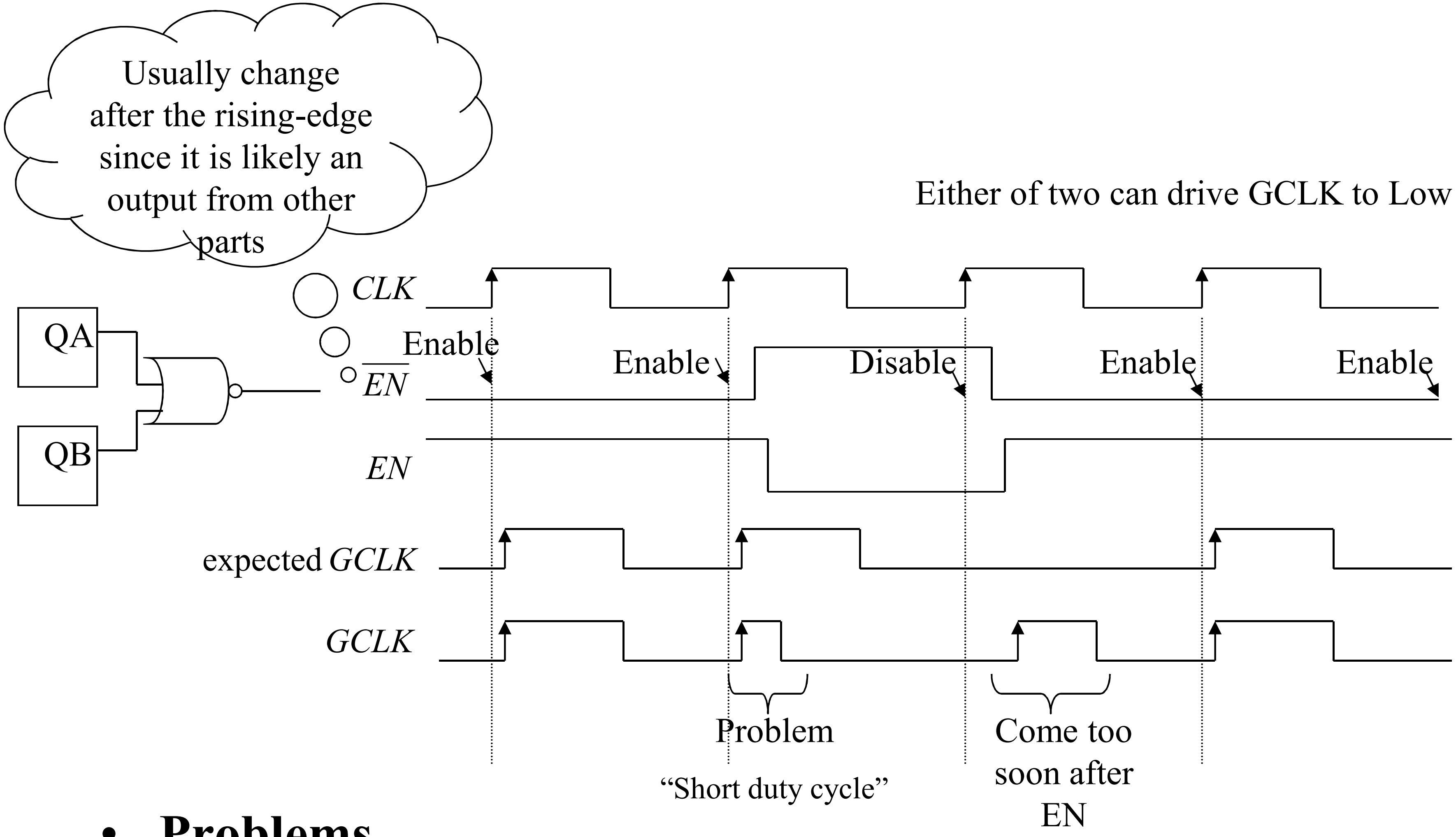
# Remind
# (General Sequential Circuit Structure)



# Remind
# (General Sequential Circuit Structure)



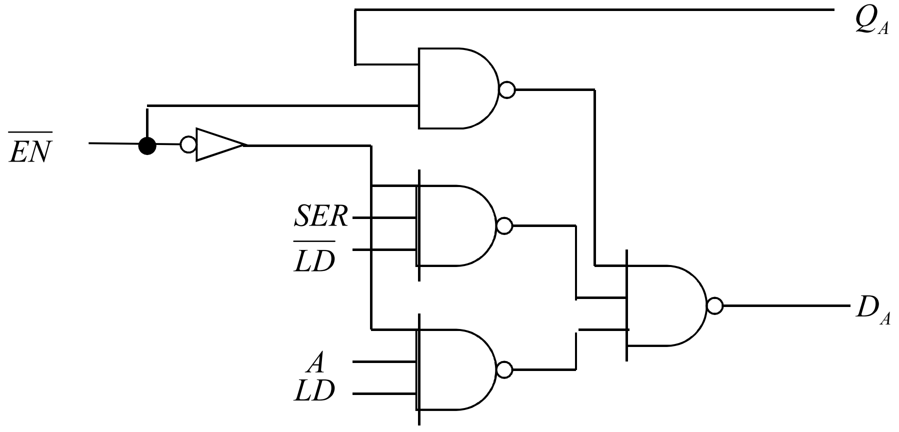
# Gating the CLK



- **Problems**
  - Short GCLK pulse – dependent on delay
  - Comes too soon (or late) after asserting enable

# Different approach to holding

- **Delay in CLK line is not good design practice**
  - Puts in CLK skew
  - All f/f CLKs don't triggered at same time → eliminates the good points to use synchronous design (we can ignore a lot of difficult timing issues)
- **Desired: "Synchronous function-enable input"**
  - EN should sampled along with data at CLK edge

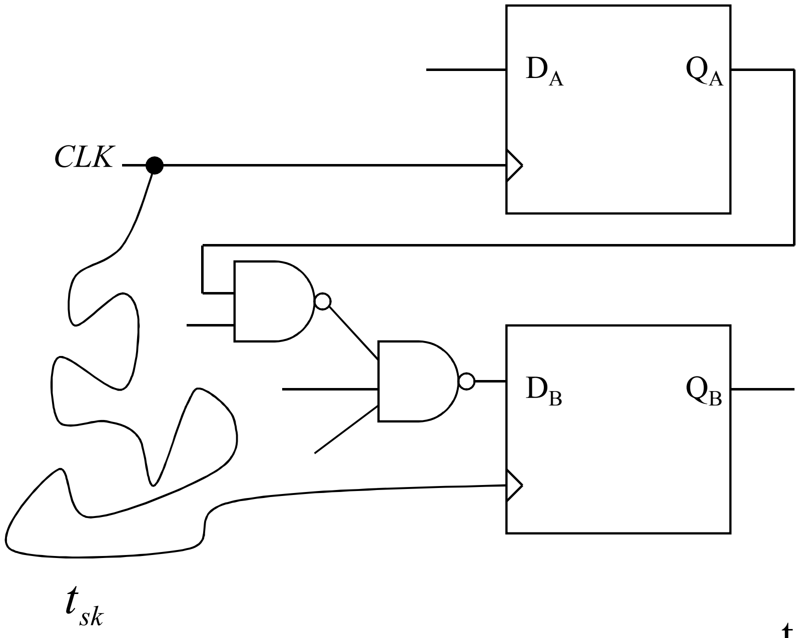


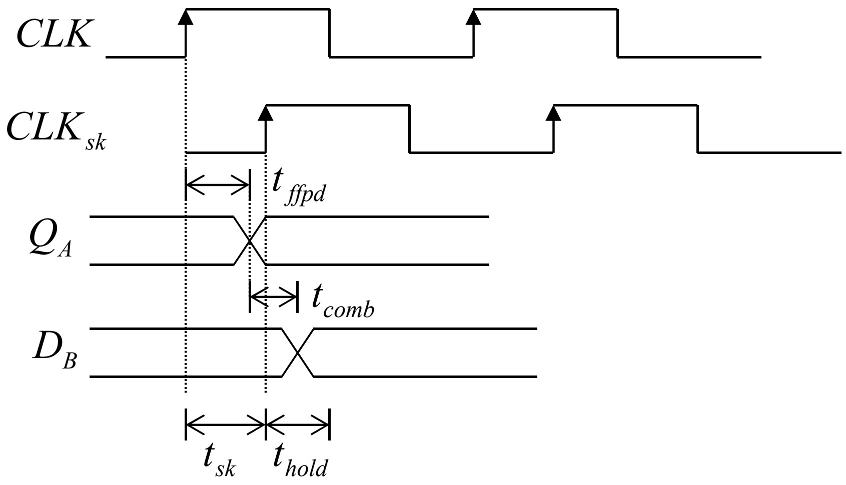
# Clock Skew

- **Clock skew: difference between arrival times of CLK at different devices**
- **Caused by**
  - Gating in the CLK line
  - Delay along long lines (1ns/ft – speed of light) – CAD serial routing

# Clock Skew

- **Problem**
  - If $t_{sk}$ too long – CLK edge get to B f/f after $D_B$ changes => Wrong operations
  - In general – hold time on $D_B$ can be violated





$$t_{ffpd}(\min)+t_{comb}(\min)>t_{sk}+t_h$$

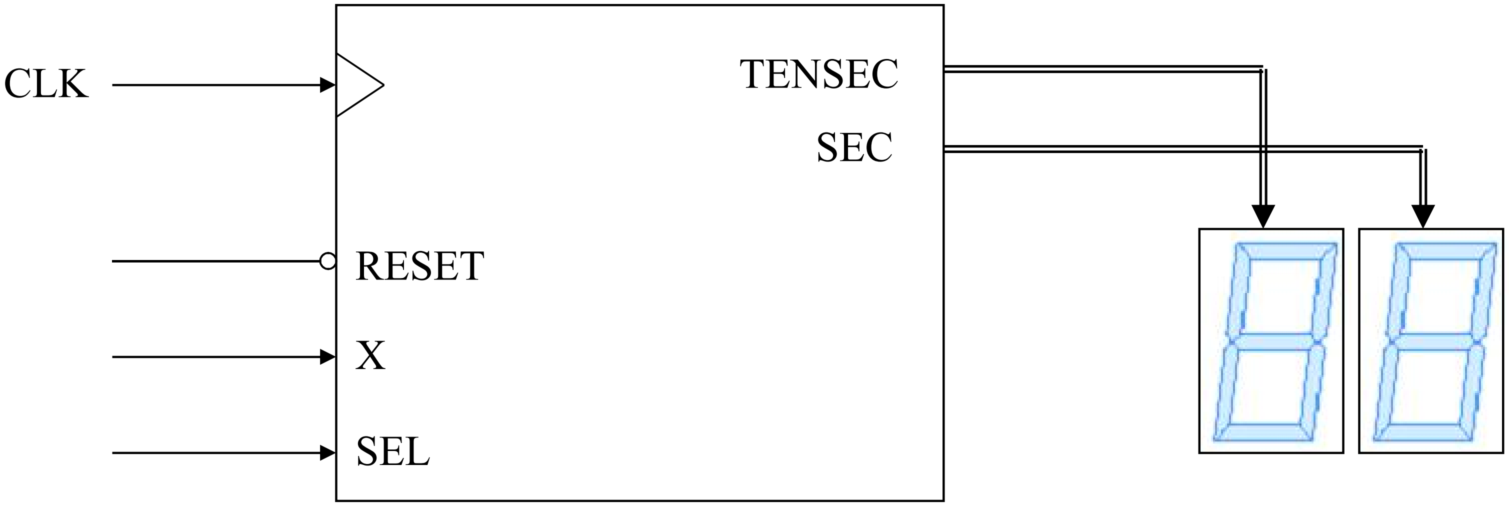
$t_{sk}$ max?: use timing specs

x74 (25ns) + x00 (9ns) + x10 (9ns) – x74hold (5ns) = 38ns

# Example 2: Problem Statement (Stop Watch: System Controller Approach)

- **Design a “stop watch” that can measure times taken for two events**
  - Inputs
    - CLK = 16 Hz
    - RESET: Asynchronously reset everything
    - X: comes from push button
      - First Push: Start timer
      - Second Push: Store the time taken for the first event
      - Third Push: Store the time taken for the second event
    - SEL: select output (High: first event time, Low: second event time)
  - Outputs
    - Two decimal digits to be connected to two seven segment displays
      - Can display 00 ~ 99

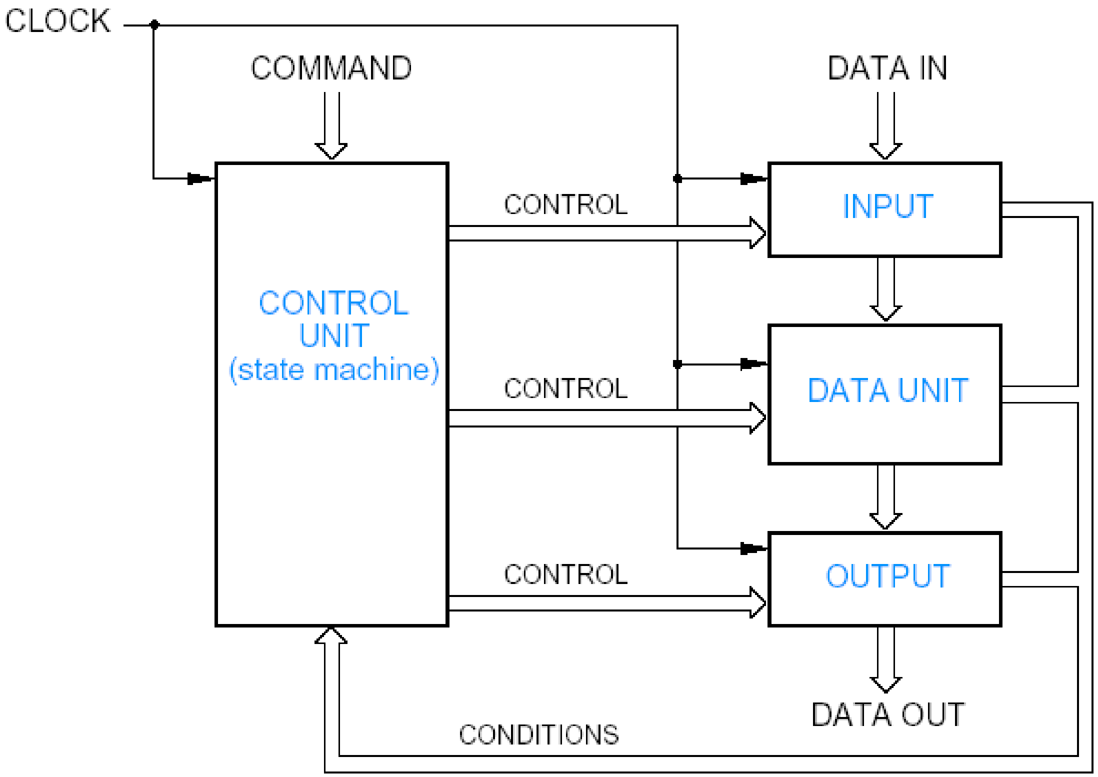
# Overall Architecture Design

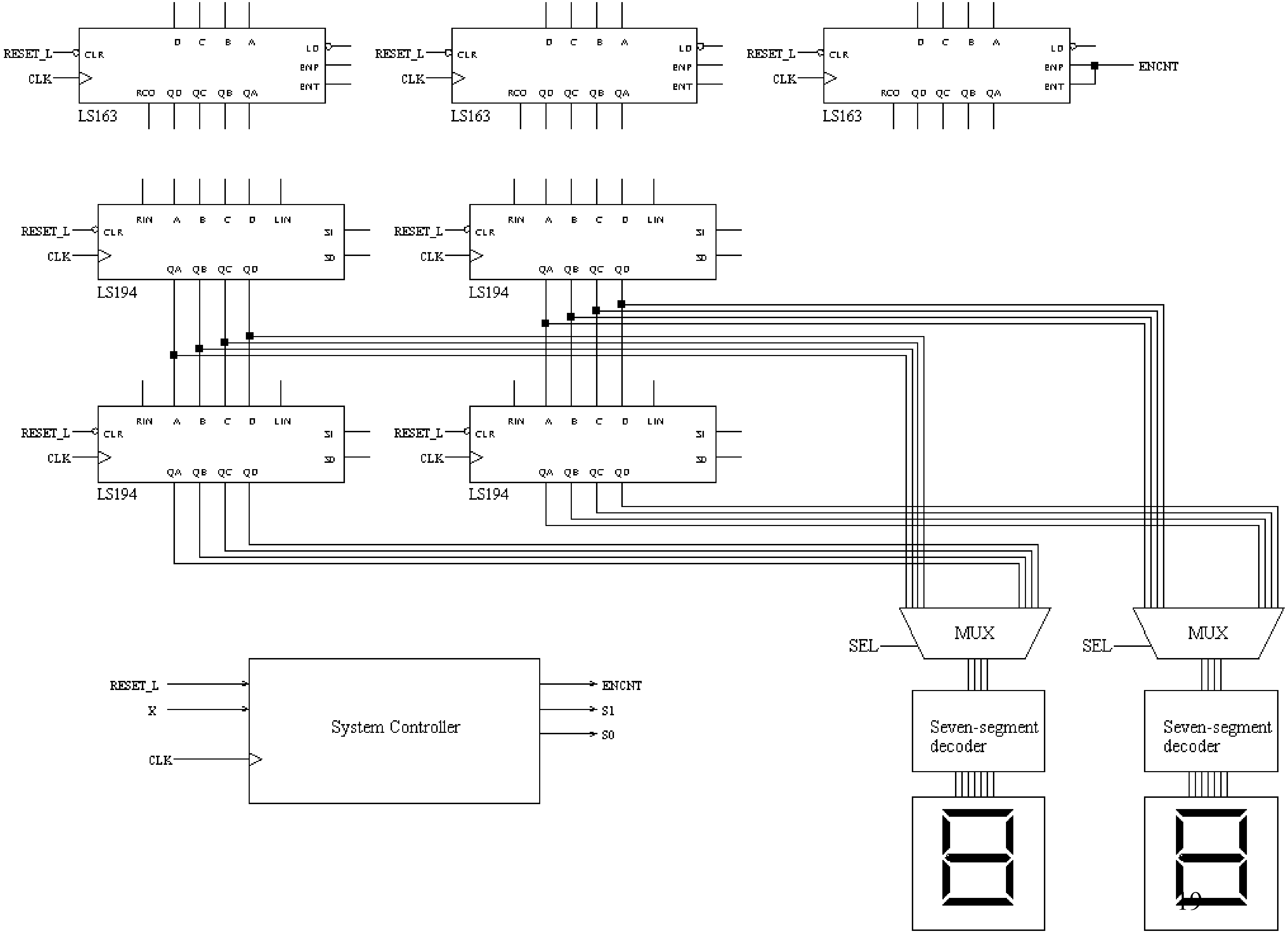


- Which LBB (among those we learn in the class) and how many?
- How to decompose the whole system into data part and control part?

# Design System Architecture

- **First step: divide the system into a control unit and data unit**
  - Data unit – stores, routes, combines, and generally process data
  - Control unit – starting & stopping the process, testing conditions, and deciding what to do next




RESET_L
CLK
CLR
D
C
B
A
LD
ENP
ENT
RCO
QD
QC
QB
QA
LS163
ENCNT
RIN
LIN
SI
SO
LS194
SEL
MUX
Seven-segment decoder
System Controller
X
S1
S0

# System controller design

# Table (Variable Entered Table)

|  | P.S. | N.S. | Output |
|---|---|---|---|
| State symbol | Q1 Q0 | D1 D0 | S1 S0 ENCNT |
| A | 0 0 |  |  |
| B | 0 1 |  |  |
| C | 1 1 |  |  |
| unused | 1 0 |  |  |

# Excitation Eqs, Output Eqs

| Q1 \ Q0 | 0 | 1 |
|---|---|---|
| 0 | | |
| 1 | | |

D1=

| Q1 \ Q0 | 0 | 1 |
|---|---|---|
| 0 | | |
| 1 | | |

D0=

| Q1 \ Q0 | 0 | 1 |
|---|---|---|
| 0 | | |
| 1 | | |

S1=

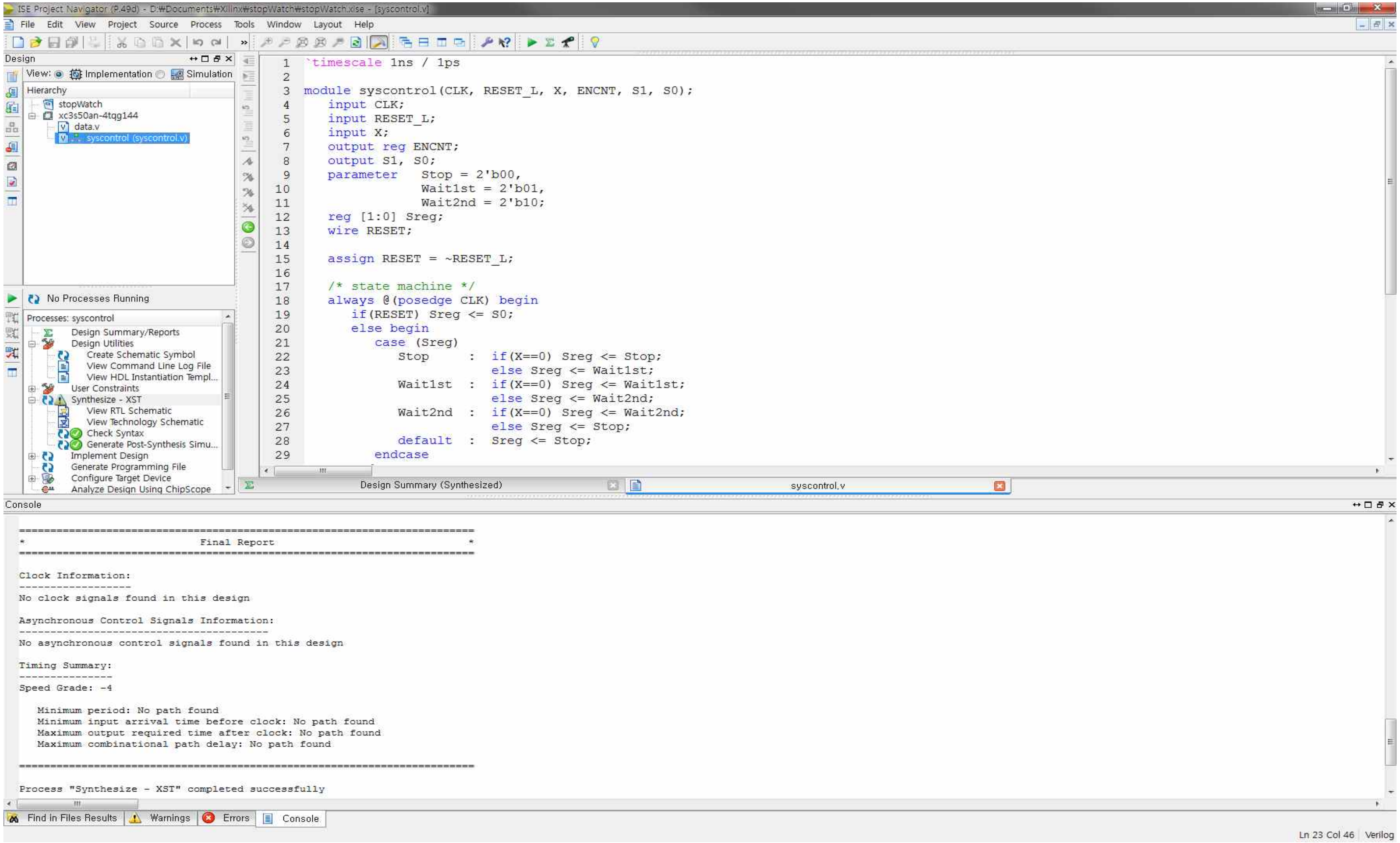
| Q1 \ Q0 | 0 | 1 |
|---|---|---|
| 0 | | |
| 1 | | |

S0=

| Q1 \ Q0 | 0 | 1 |
|---|---|---|
| 0 | | |
| 1 | | |

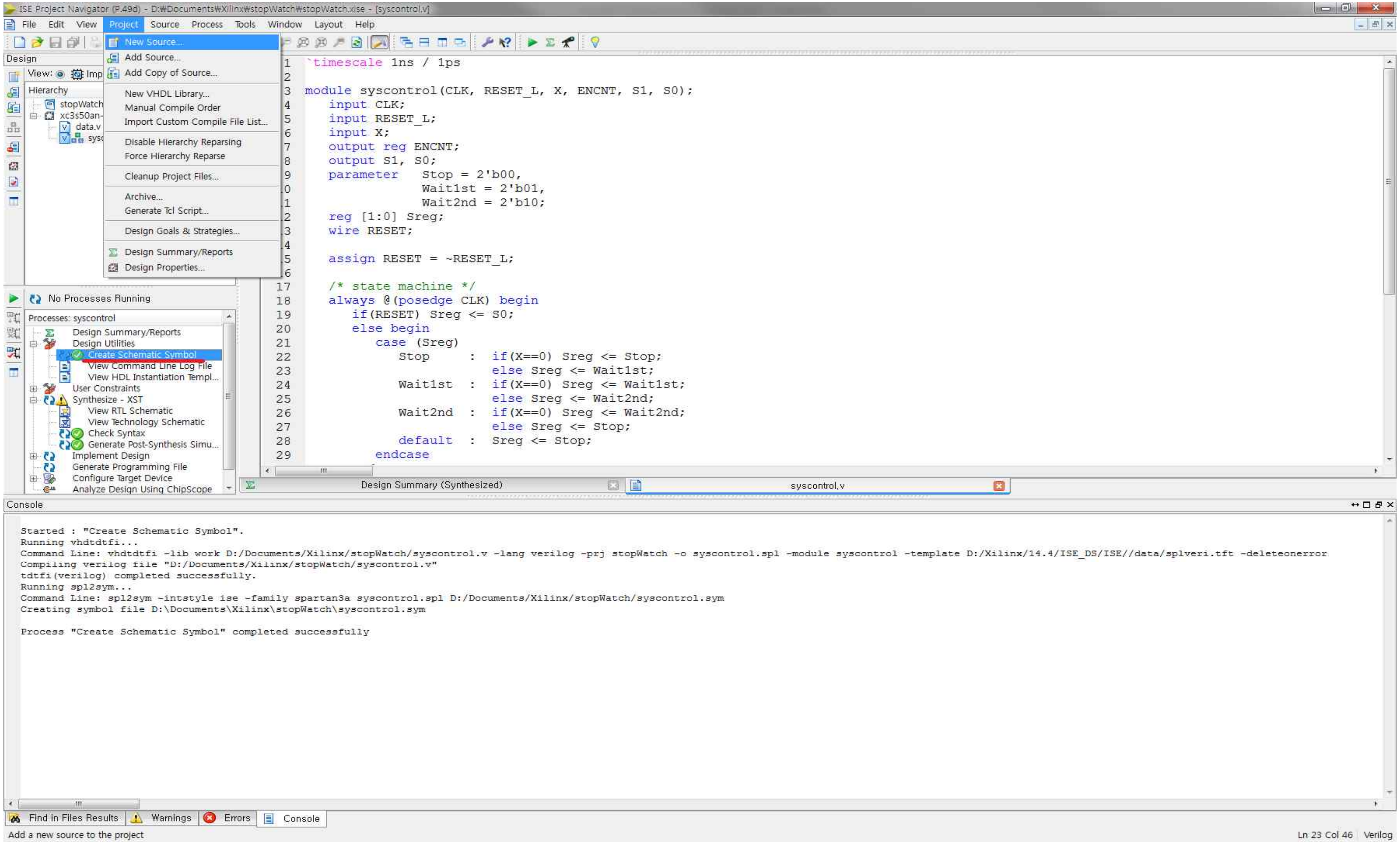
ENCNT=

# Alternative (Verilog program)



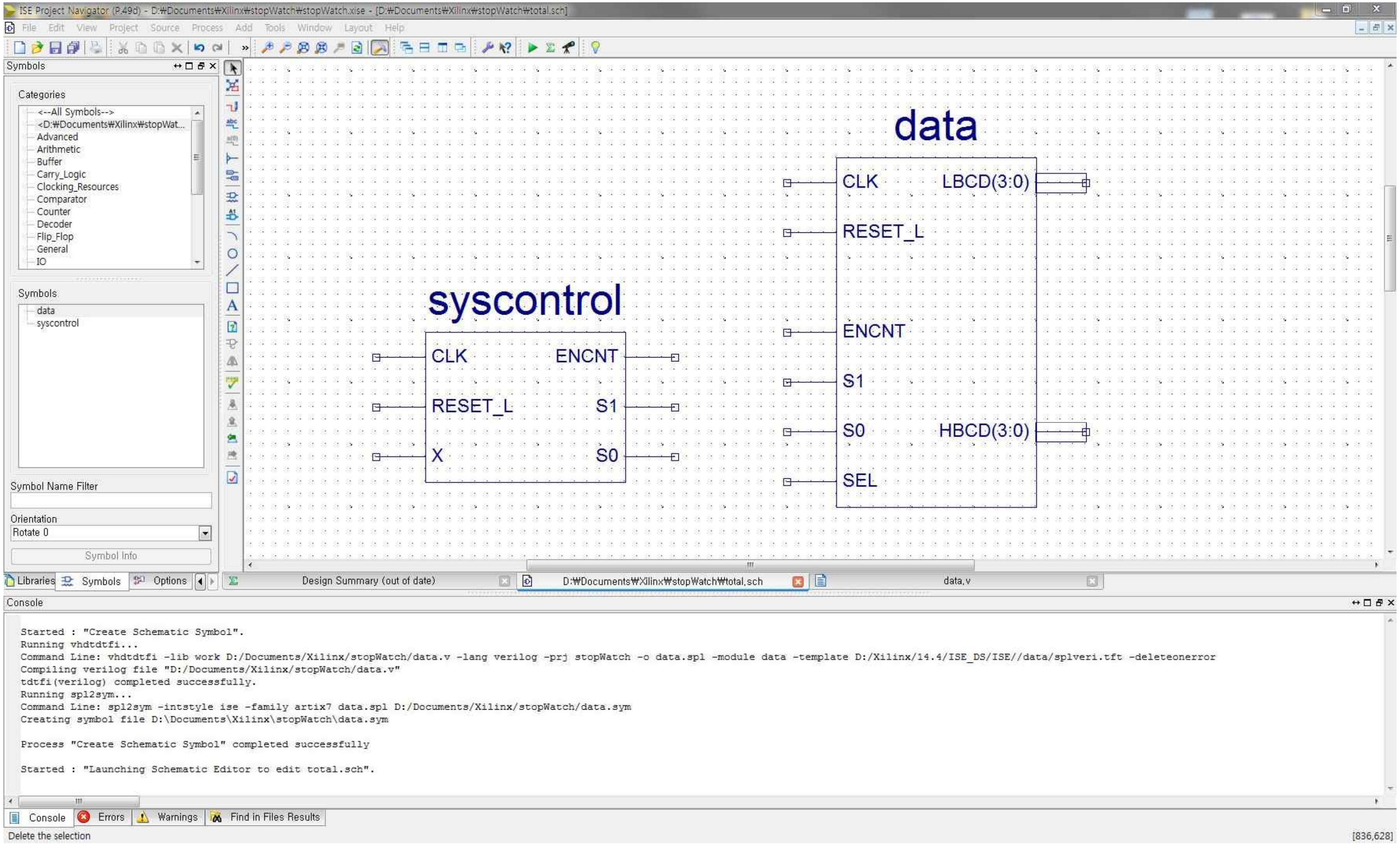
```verilog
`timescale 1ns / 1ps

module syscontrol(CLK, RESET_L, X, ENCNT, S1, S0);
   input CLK;
   input RESET_L;
   input X;
   output reg ENCNT;
   output S1, S0;
   parameter   Stop = 2'b00,
               Wait1st = 2'b01,
               Wait2nd = 2'b10;
   reg [1:0] Sreg;
   wire RESET;

   assign RESET = ~RESET_L;

   /* state machine */
   always @(posedge CLK) begin
      if(RESET) Sreg <= S0;
      else begin
         case (Sreg)
            Stop     :  if(X==0) Sreg <= Stop;
                        else Sreg <= Wait1st;
            Wait1st  :  if(X==0) Sreg <= Wait1st;
                        else Sreg <= Wait2nd;
            Wait2nd  :  if(X==0) Sreg <= Wait2nd;
                        else Sreg <= Stop;
            default  :  Sreg <= Stop;
         endcase
```

```
=========================================================================
*                            Final Report                               *
=========================================================================

Clock Information:
------------------
No clock signals found in this design

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -4

   Minimum period: No path found
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: No path found

=========================================================================

Process "Synthesize - XST" completed successfully
```

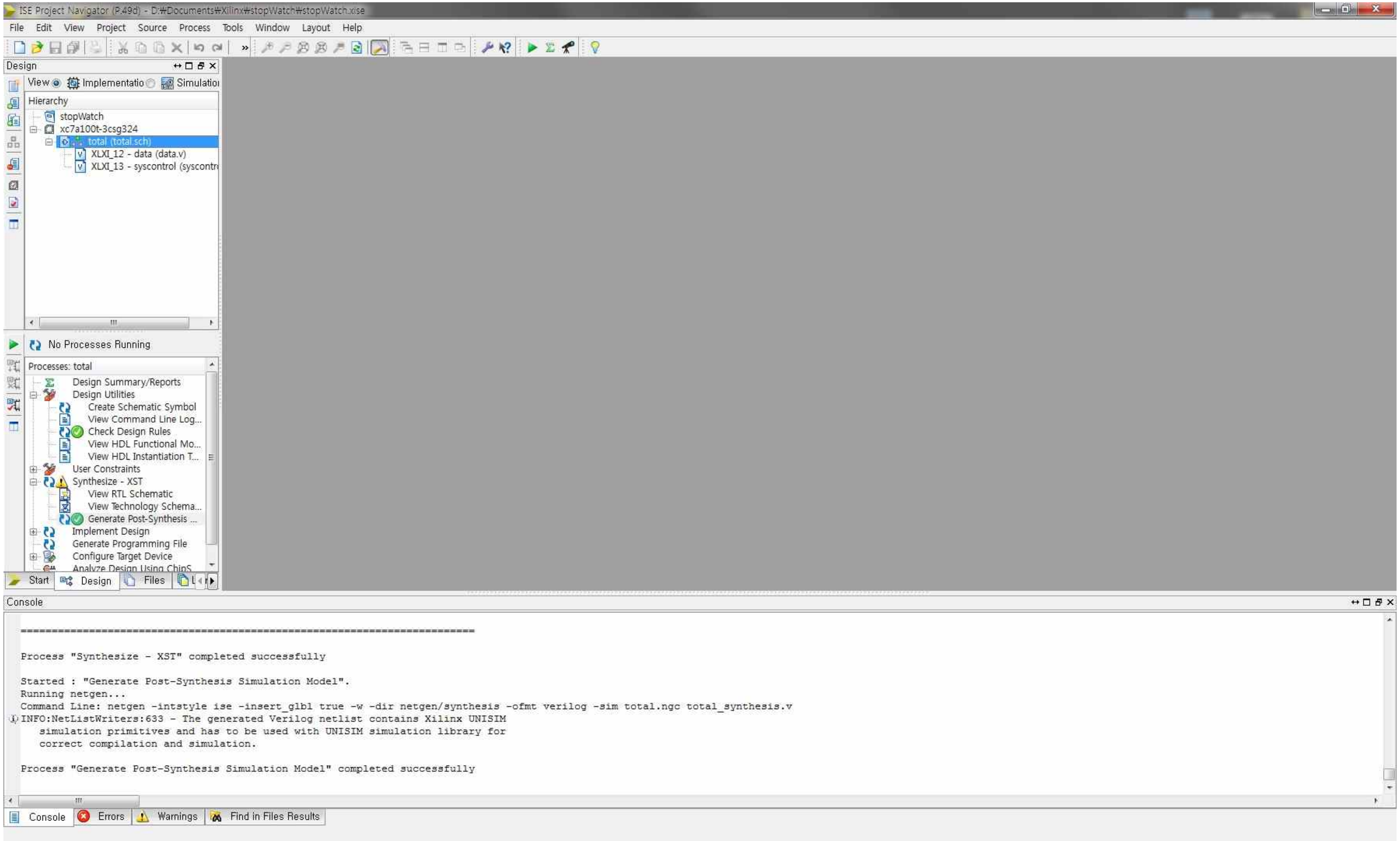


```
`timescale 1ns / 1ps

module syscontrol(CLK, RESET_L, X, ENCNT, S1, S0);
   input CLK;
   input RESET_L;
   input X;
   output reg ENCNT;
   output S1, S0;
   parameter   Stop = 2'b00,
               Wait1st = 2'b01,
               Wait2nd = 2'b10;
   reg [1:0] Sreg;
   wire RESET;

   assign RESET = ~RESET_L;

   /* state machine */
   always @(posedge CLK) begin
      if(RESET) Sreg <= S0;
      else begin
         case (Sreg)
            Stop     :  if(X==0) Sreg <= Stop;
                        else Sreg <= Wait1st;
            Wait1st  :  if(X==0) Sreg <= Wait1st;
                        else Sreg <= Wait2nd;
            Wait2nd  :  if(X==0) Sreg <= Wait2nd;
                        else Sreg <= Stop;
            default  :  Sreg <= Stop;
         endcase
```

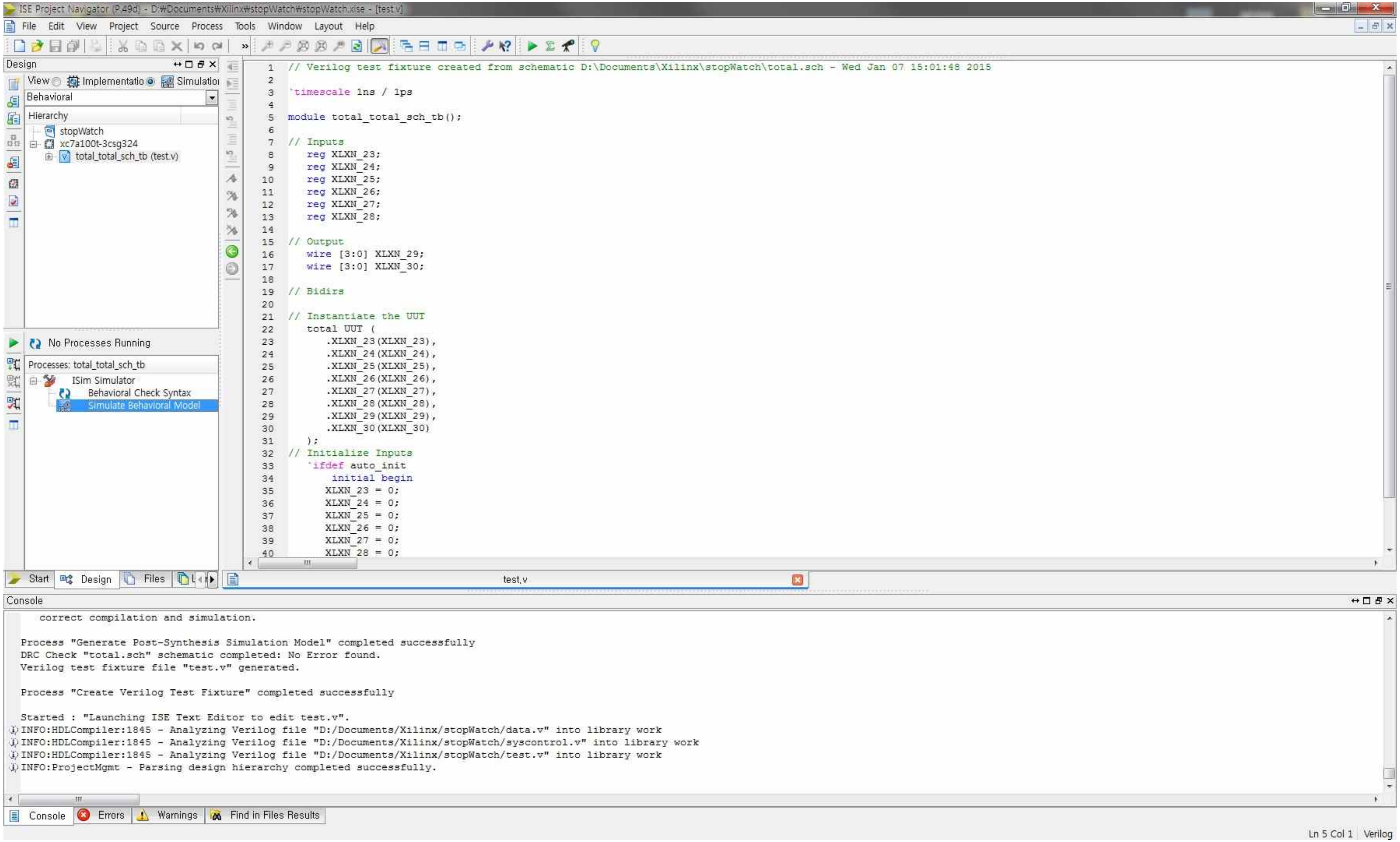
```
Started : "Create Schematic Symbol".
Running vhdtdtfi...
Command Line: vhdtdtfi -lib work D:/Documents/Xilinx/stopWatch/syscontrol.v -lang verilog -prj stopWatch -o syscontrol.spl -module syscontrol -template D:/Xilinx/14.4/ISE_DS/ISE//data/splveri.tft -deleteonerror
Compiling verilog file "D:/Documents/Xilinx/stopWatch/syscontrol.v"
tdtfi(verilog) completed successfully.
Running spl2sym...
Command Line: spl2sym -intstyle ise -family spartan3a syscontrol.spl D:/Documents/Xilinx/stopWatch/syscontrol.sym
Creating symbol file D:\Documents\Xilinx\stopWatch\syscontrol.sym

Process "Create Schematic Symbol" completed successfully
```



```
Started : "Create Schematic Symbol".
Running vhdtdtfi...
Command Line: vhdtdtfi -lib work D:/Documents/Xilinx/stopWatch/data.v -lang verilog -prj stopWatch -o data.spl -module data -template D:/Xilinx/14.4/ISE_DS/ISE//data/splveri.tft -deleteonerror
Compiling verilog file "D:/Documents/Xilinx/stopWatch/data.v"
tdtfi(verilog) completed successfully.
Running spl2sym...
Command Line: spl2sym -intstyle ise -family artix7 data.spl D:/Documents/Xilinx/stopWatch/data.sym
Creating symbol file D:\Documents\Xilinx\stopWatch\data.sym

Process "Create Schematic Symbol" completed successfully

Started : "Launching Schematic Editor to edit total.sch".
```



```
========================================================================

Process "Synthesize - XST" completed successfully

Started : "Generate Post-Synthesis Simulation Model".
Running netgen...
Command Line: netgen -intstyle ise -insert_glbl true -w -dir netgen/synthesis -ofmt verilog -sim total.ngc total_synthesis.v
INFO:NetListWriters:633 - The generated Verilog netlist contains Xilinx UNISIM
   simulation primitives and has to be used with UNISIM simulation library for
   correct compilation and simulation.

Process "Generate Post-Synthesis Simulation Model" completed successfully
```


ISE Project Navigator (P.49d) - D:₩Documents₩Xilinx₩stopWatch₩stopWatch.xise - [test.v]
File Edit View Project Source Process Tools Window Layout Help
Design
View ○ Implementatio ● Simulatio
Behavioral
Hierarchy
stopWatch
xc7a100t-3csg324
total_total_sch_tb (test.v)
No Processes Running
Processes: total_total_sch_tb
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
Start
Design
Files
test.v
Console
Errors
Warnings
Find in Files Results
Ln 5 Col 1
Verilog

```
// Verilog test fixture created from schematic D:\Documents\Xilinx\stopWatch\total.sch - Wed Jan 07 15:01:48 2015

`timescale 1ns / 1ps

module total_total_sch_tb();

// Inputs
   reg XLXN_23;
   reg XLXN_24;
   reg XLXN_25;
   reg XLXN_26;
   reg XLXN_27;
   reg XLXN_28;

// Output
   wire [3:0] XLXN_29;
   wire [3:0] XLXN_30;

// Bidirs

// Instantiate the UUT
   total UUT (
      .XLXN_23(XLXN_23),
      .XLXN_24(XLXN_24),
      .XLXN_25(XLXN_25),
      .XLXN_26(XLXN_26),
      .XLXN_27(XLXN_27),
      .XLXN_28(XLXN_28),
      .XLXN_29(XLXN_29),
      .XLXN_30(XLXN_30)
   );
// Initialize Inputs
   `ifdef auto_init
       initial begin
      XLXN_23 = 0;
      XLXN_24 = 0;
      XLXN_25 = 0;
      XLXN_26 = 0;
      XLXN_27 = 0;
      XLXN_28 = 0;
```

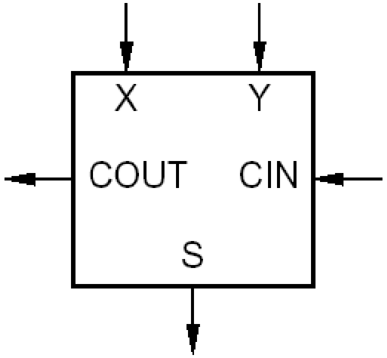
```
   correct compilation and simulation.

Process "Generate Post-Synthesis Simulation Model" completed successfully
DRC Check "total.sch" schematic completed: No Error found.
Verilog test fixture file "test.v" generated.

Process "Create Verilog Test Fixture" completed successfully

Started : "Launching ISE Text Editor to edit test.v".
INFO:HDLCompiler:1845 - Analyzing Verilog file "D:/Documents/Xilinx/stopWatch/data.v" into library work
INFO:HDLCompiler:1845 - Analyzing Verilog file "D:/Documents/Xilinx/stopWatch/syscontrol.v" into library work
INFO:HDLCompiler:1845 - Analyzing Verilog file "D:/Documents/Xilinx/stopWatch/test.v" into library work
INFO:ProjectMgmt - Parsing design hierarchy completed successfully.
```

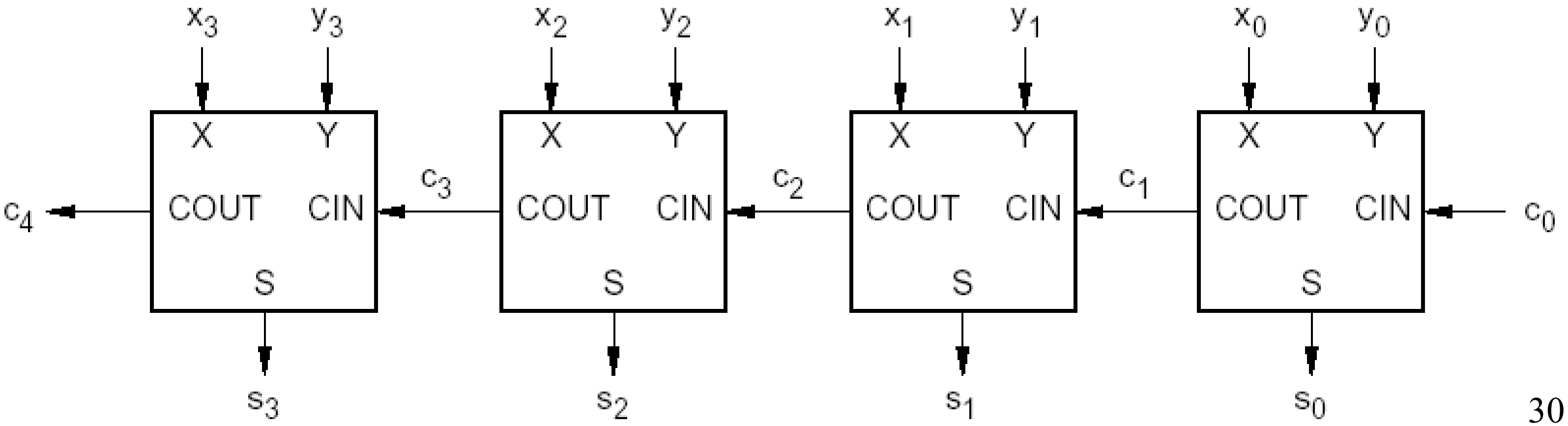
# Example 3: 4-bit Serial Adder

(Decomposing state machines, synchronous design methodology, system controller design)

# Review of combinational adders

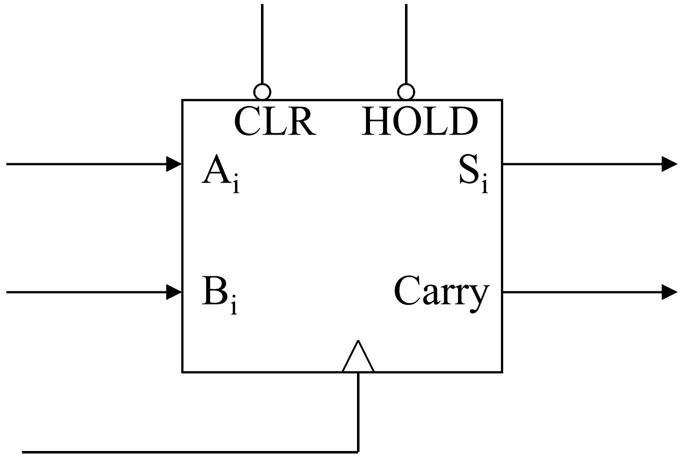
- **1-bit full adder and 4-bit ripple adder**





# Binary serial adder

- **Serial input feed (two data streams A and B)**
- **Generate serial output**

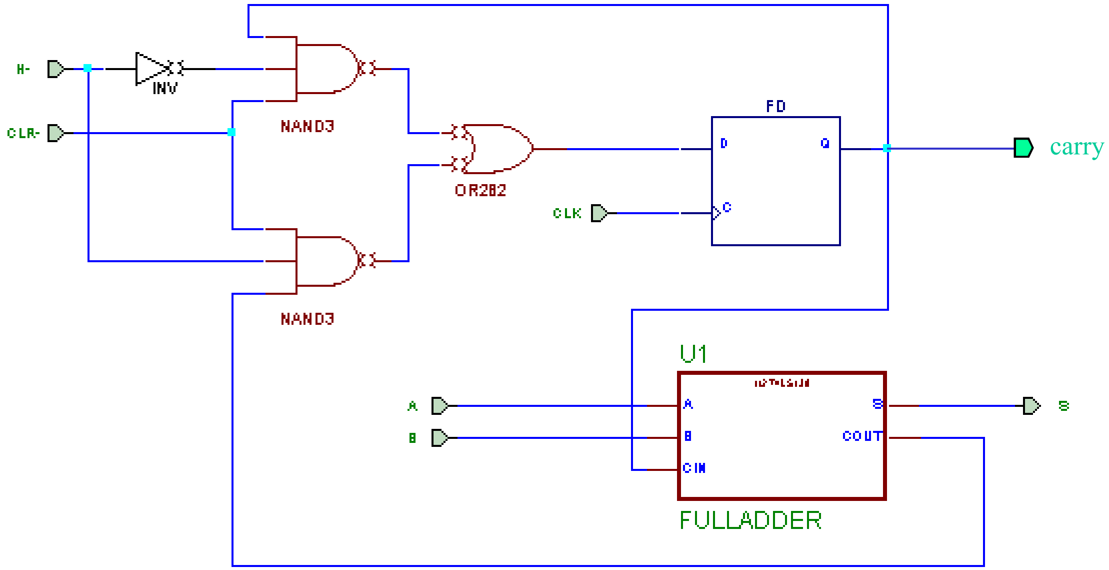


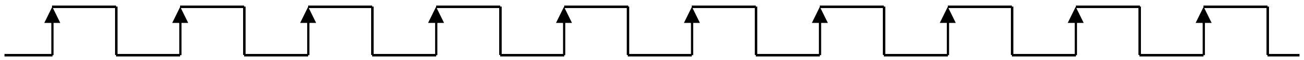
CLR – clear the previous carry synchronously

HOLD – hold the previous value of the carry

# Binary serial adder implementation

- **LS183: dual 1 bit full adders**

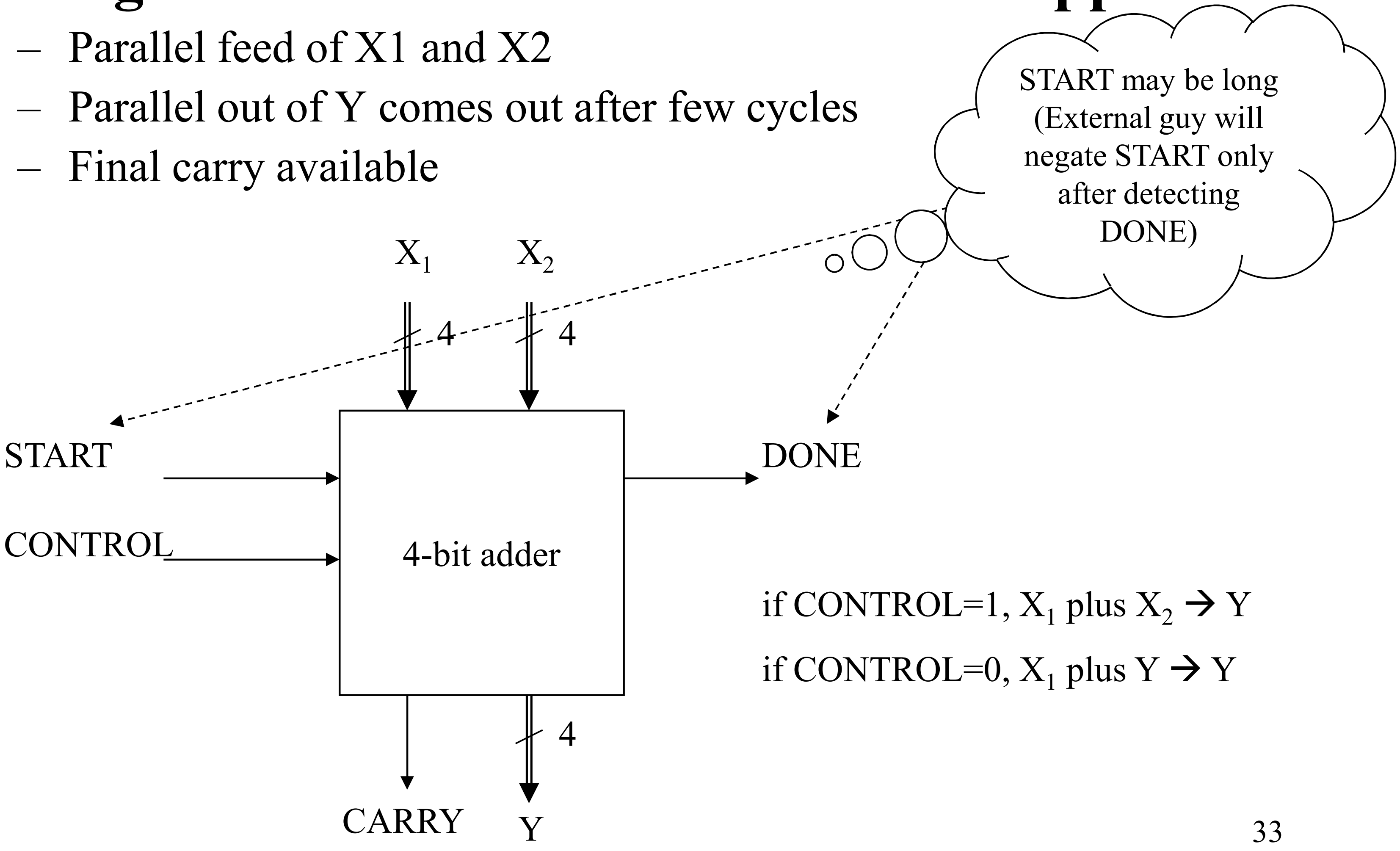




| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| B | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| Q(Carry) | 0 | | | | | | | | |
| S | 0 | | | | | | | | |

# Problem Statement

- **Design a 4-bit adder based on a serial approach**
  - Parallel feed of X1 and X2
  - Parallel out of Y comes out after few cycles
  - Final carry available


START may be long (External guy will negate START only after detecting DONE)
$X_1$
$X_2$
4
4
START
CONTROL
4-bit adder
DONE
4
CARRY
Y

if CONTROL=1, $X_1$ plus $X_2$ → Y

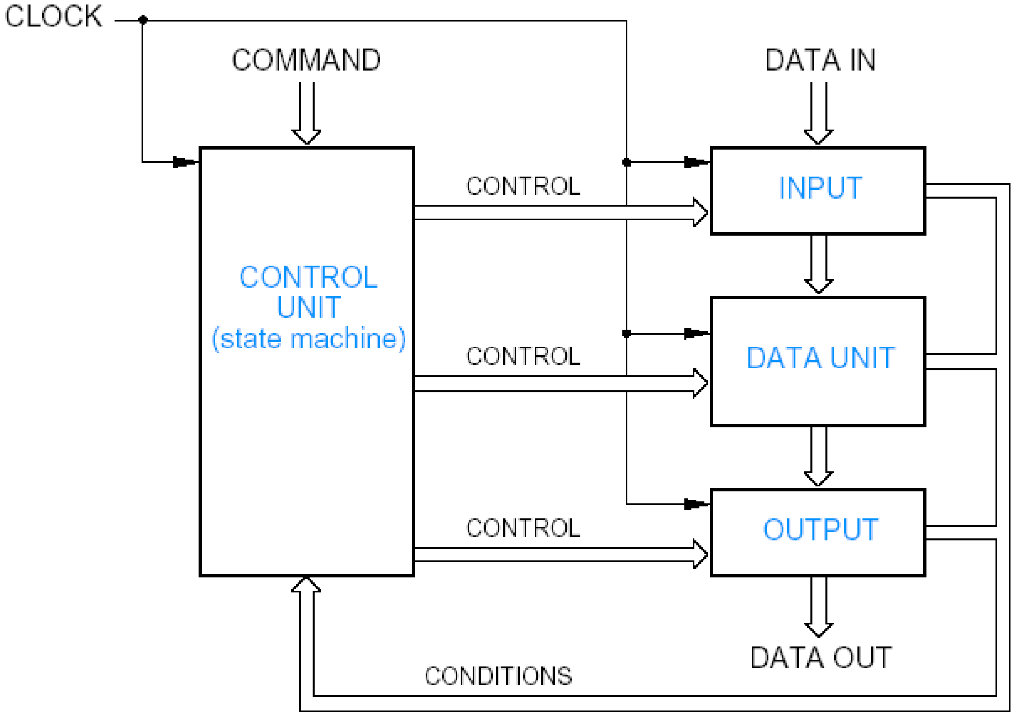
if CONTROL=0, $X_1$ plus Y → Y

# Start from LBB

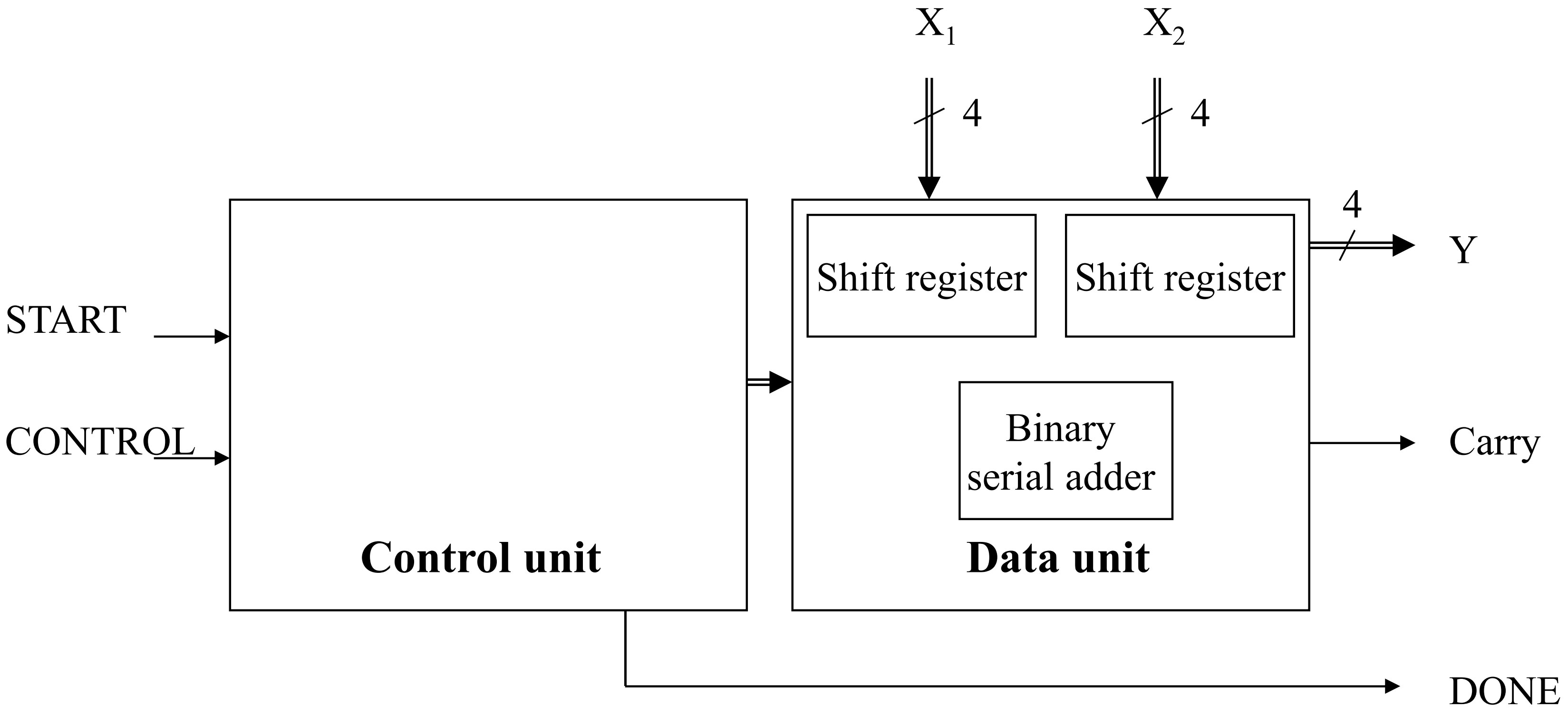
- **Here, we have some basic building blocks in mind**
  - 4-bit shift registers, binary serial adder, etc
- **Tie these elements together and make them controllable from the outside world**
- **We want to take maximum advantage of common building blocks (MSI chips that are available)**

# Design System Architecture

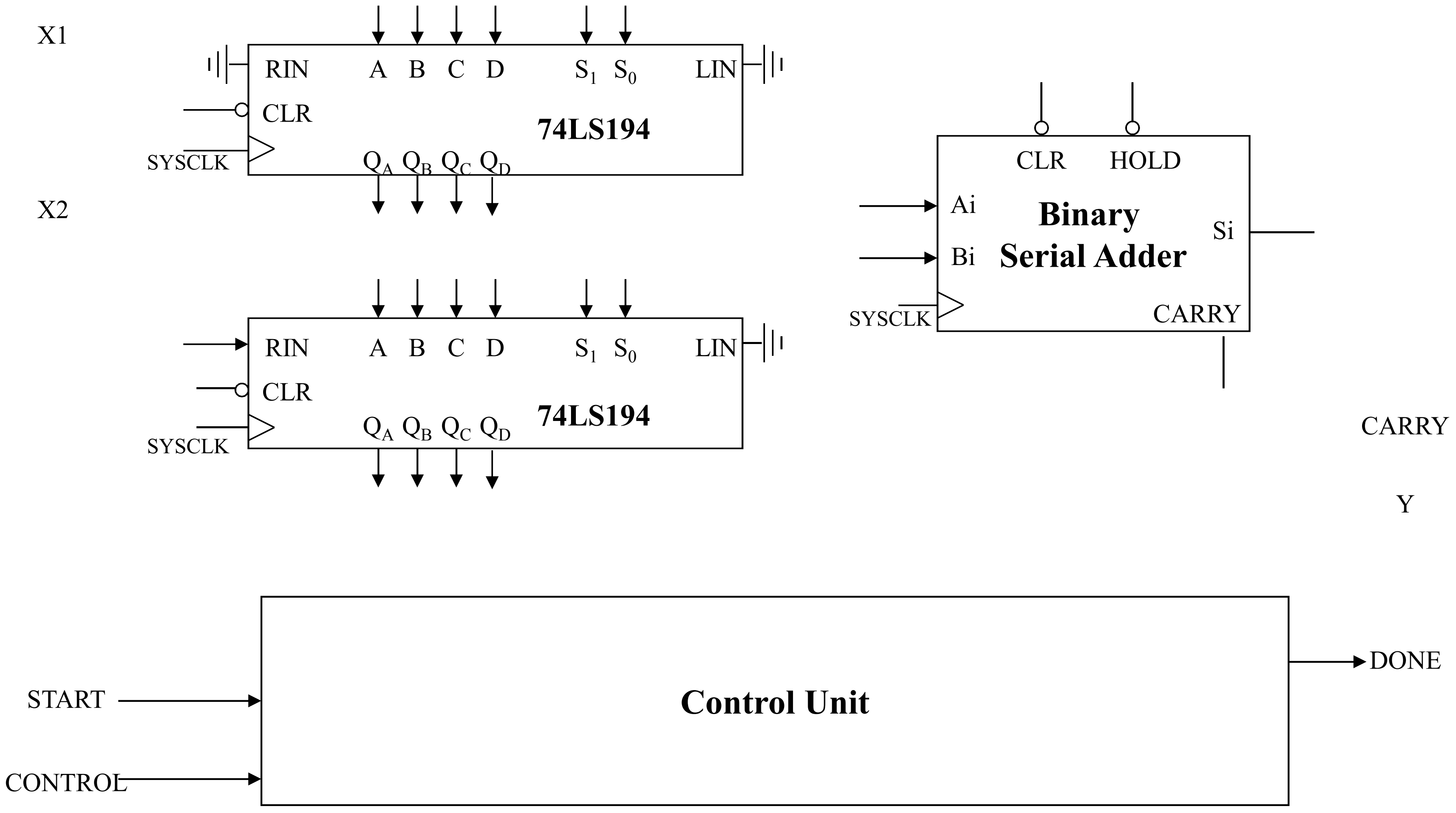
- **First step: divide the system into a control unit and data unit**
  - Data unit – stores, routes, combines, and generally process data
  - Control unit – starting & stopping the process, testing conditions, and deciding what to do next



# Data unit & Control unit



# Architecture

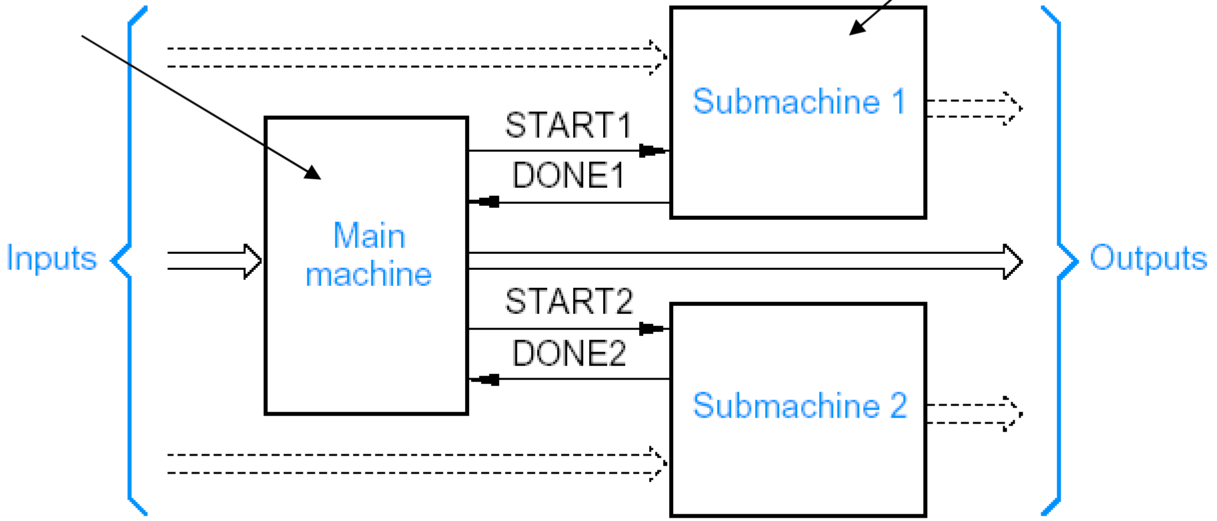


# Decomposing state machine

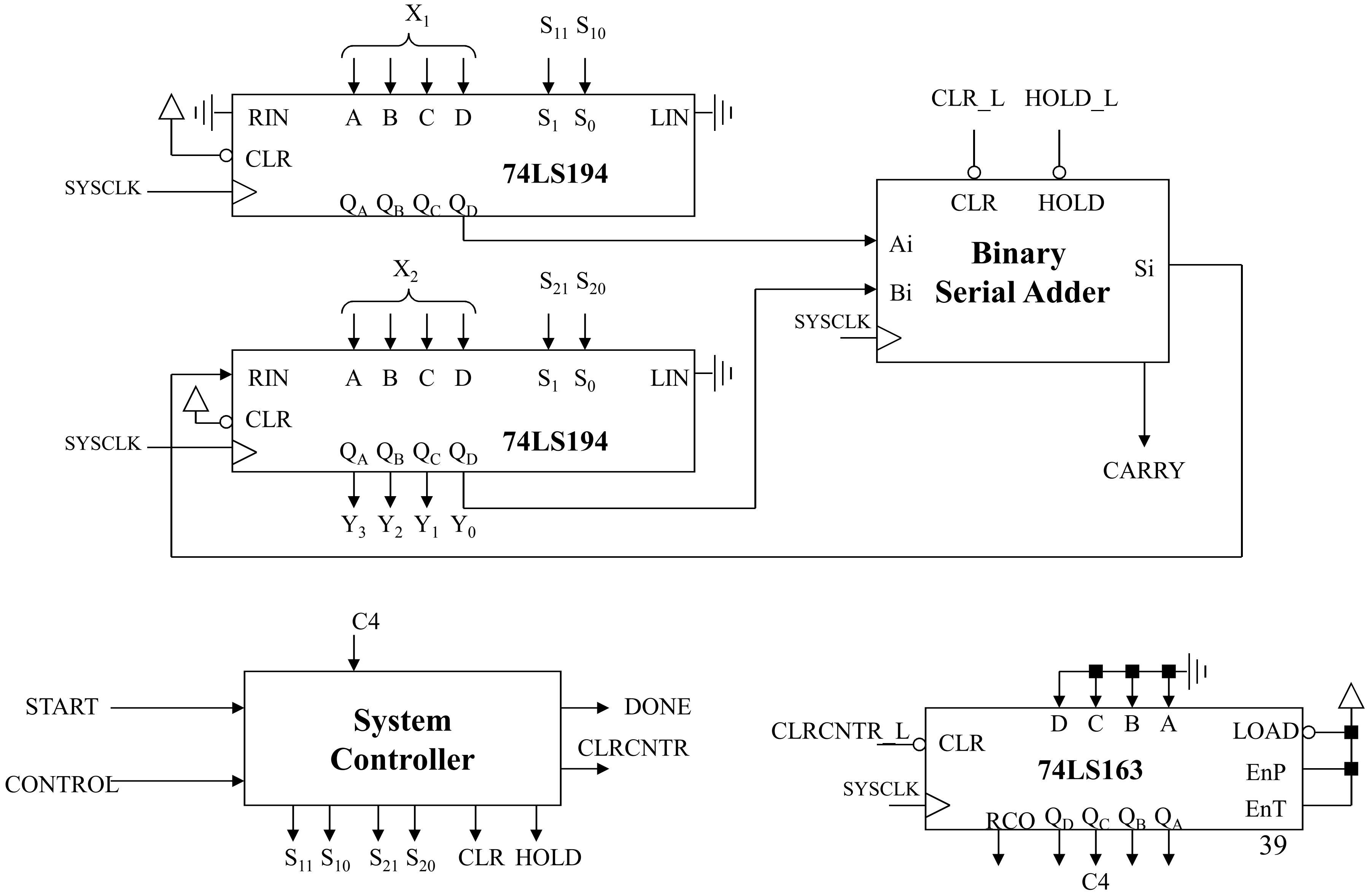
- **Second step: the state machine part (control unit) can be decomposed into several parts**
  - Main machine (system controller) – provides the primary inputs and outputs and does top level control
  - Submachines – perform lower-level steps under control of main machine
    - Typical submachine – counter
      - Saves $2^n$ states in main machine
      - Easier to follow the control

2 bit counter to count # of bits added

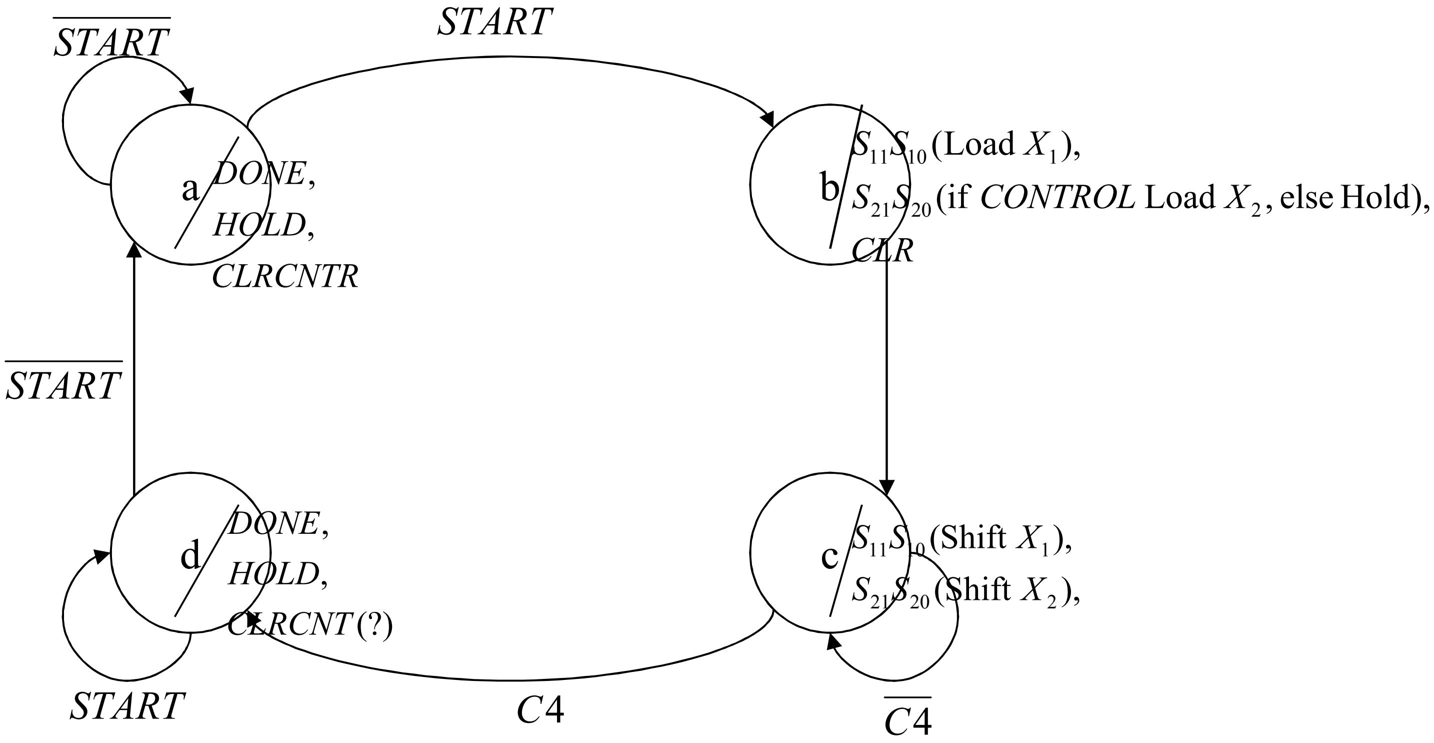
Our system controller



# Architecture



# System controller design


$\overline{START}$
$START$
a
$DONE$,
$HOLD$,
$CLRCNTR$
b
$S_{11}S_{10}$ (Load $X_1$),
$S_{21}S_{20}$ (if $CONTROL$ Load $X_2$, else Hold),
$CLR$
$\overline{START}$
d
$DONE$,
$HOLD$,
$CLRCNT$ (?)
c
$S_{11}S_{10}$ (Shift $X_1$),
$S_{21}S_{20}$ (Shift $X_2$),
$START$
$C4$
$\overline{C4}$

# Example results

```
  0101 (5)
+ 0001 (1)
-------------
  0110 (6)
```

| Sate | Carry | Reg 1 | Reg 2 | Counter |
|---|---|---|---|---|
| a | - | - | - | 0 |
| b | - | - | - | 0 |
| c | 0 | 0101 | 0001 | 1 |
| c | 1 | 0010 | 0000 | 2 |
| c | 0 | 0001 | 1000 | 3 |
| c | 0 | 0000 | 1100 | 4 |
| d | 0 | 0000 | 0110 | 5 |

# State assignment and Transition Table

| | Q1Q0 | Q1Q0 (D1D0) | DONE | CLRCNTR | CLR | HOLD | S11 | S10 | S21 | S20 |
|---|---|---|---|---|---|---|---|---|---|---|
| a | 0 0 | 0 ST | 1 | 1 | 0 | 1 | x | x | 0 | 0 |
| b | 0 1 | 1 1 | 0 | 0 | 1 | x | 1 | 1 | CONT | CONT |
| c | 1 1 | 1 $\overline{C4}$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| d | 1 0 | ST 0 | 1 | x | 0 | 1 | x | x | 0 | 0 |

# Excitation and Output Eqs.

| $Q_1$ \ $Q_0$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | ST | 1 |

$D_1 = Q_0 + ST \cdot Q_1$

| | 0 | 1 |
|---|---|---|
| 0 | ST | 1 |
| 1 | 0 | $\overline{C4}$ |

$D_0 = \overline{Q_1}Q_0 + \overline{Q_1} \cdot ST + Q_0 \cdot \overline{C4}$

| $Q_1$ \ $Q_0$ | 0 | 1 |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 1 | 0 |

$DONE = \overline{Q_0}$

| $Q_1$ \ $Q_0$ | 0 | 1 |
|---|---|---|
| 0 | 1 | 0 |
| 1 | x | 0 |

$CLRCNTR = \overline{Q_0}$

| $Q_1$ \ $Q_0$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 0 |

$CLR = \overline{Q_1}Q_0$

| $Q_1$ \ $Q_0$ | 0 | 1 |
|---|---|---|
| 0 | 1 | x |
| 1 | 1 | 0 |

$HOLD = \overline{Q_0}$

| $Q_1$ \ $Q_0$ | 0 | 1 |
|---|---|---|
| 0 | x | 1 |
| 1 | x | 0 |

$S_{11} = \overline{Q_1}$

| $Q_1$ \ $Q_0$ | 0 | 1 |
|---|---|---|
| 0 | x | 1 |
| 1 | x | 1 |

$S_{10} = 1$

| $Q_1$ \ $Q_0$ | 0 | 1 |
|---|---|---|
| 0 | 0 | c |
| 1 | 0 | 0 |

$S_{21} = \overline{Q_1}Q_0 \cdot CONTROL$

| $Q_1$ \ $Q_0$ | 0 | 1 |
|---|---|---|
| 0 | 0 | c |
| 1 | 0 | 1 |

$S_{20} = Q_0 \cdot CONTROL + Q_1Q_0$

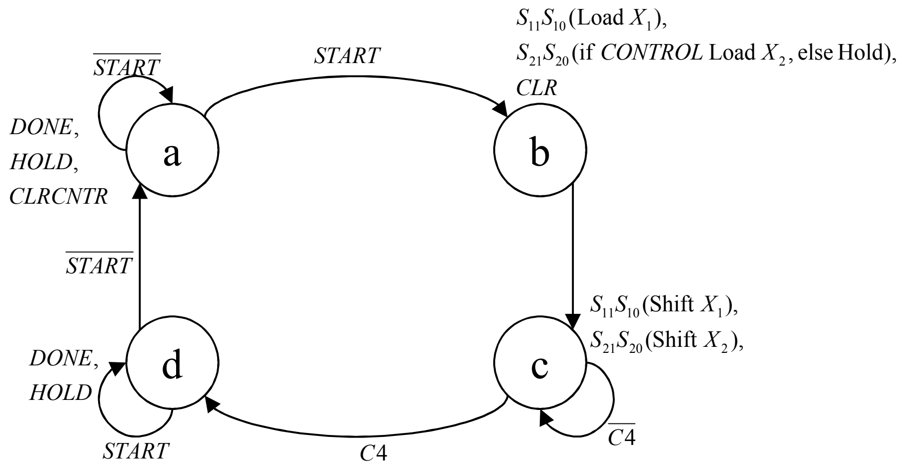
# Logic Diagram

# Asynchronous Inputs and Output Glitches

- **Things to watch out for in synchronous design**
  - Clock skew
  - Gating the clock
  - Asynchronous inputs
  - Output glitches

# Example

- **Binary counting order for our previous state assignment**



$S_{11}S_{10}$(Load $X_1$),
$S_{21}S_{20}$(if *CONTROL* Load $X_2$, else Hold),
*CLR*
$\overline{START}$
*START*
*DONE*,
*HOLD*,
*CLRCNTR*
a
b
$\overline{START}$
$S_{11}S_{10}$(Shift $X_1$),
$S_{21}S_{20}$(Shift $X_2$),
*DONE*,
*HOLD*
d
c
*START*
*C*4
$\overline{C4}$

| | Q1Q0 | Q1Q0 (D1D0) | DONE | CLRCNTR | CLR | HOLD | S11 | S10 | S21 | S20 |
|---|---|---|---|---|---|---|---|---|---|---|
| a | 0 0 | 0 ST | 1 | 1 | 0 | 1 | x | x | 0 | 0 |
| b | 0 1 | 1 0 | 0 | 0 | 1 | x | 1 | 1 | CONT’ | CONT’ |
| c | 1 0 | 1 C4 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| d | 1 1 | ST ST | 1 | x | 0 | 1 | x | x | 0 | 0 |

# Focus on part of solution

| $Q_1 \backslash Q_0$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | ST |

$D_1 = Q_1\overline{Q_0} + \overline{Q_1}Q_0 + ST \cdot Q_1$

| | 0 | 1 |
|---|---|---|
| 0 | ST | 0 |
| 1 | C4 | ST |

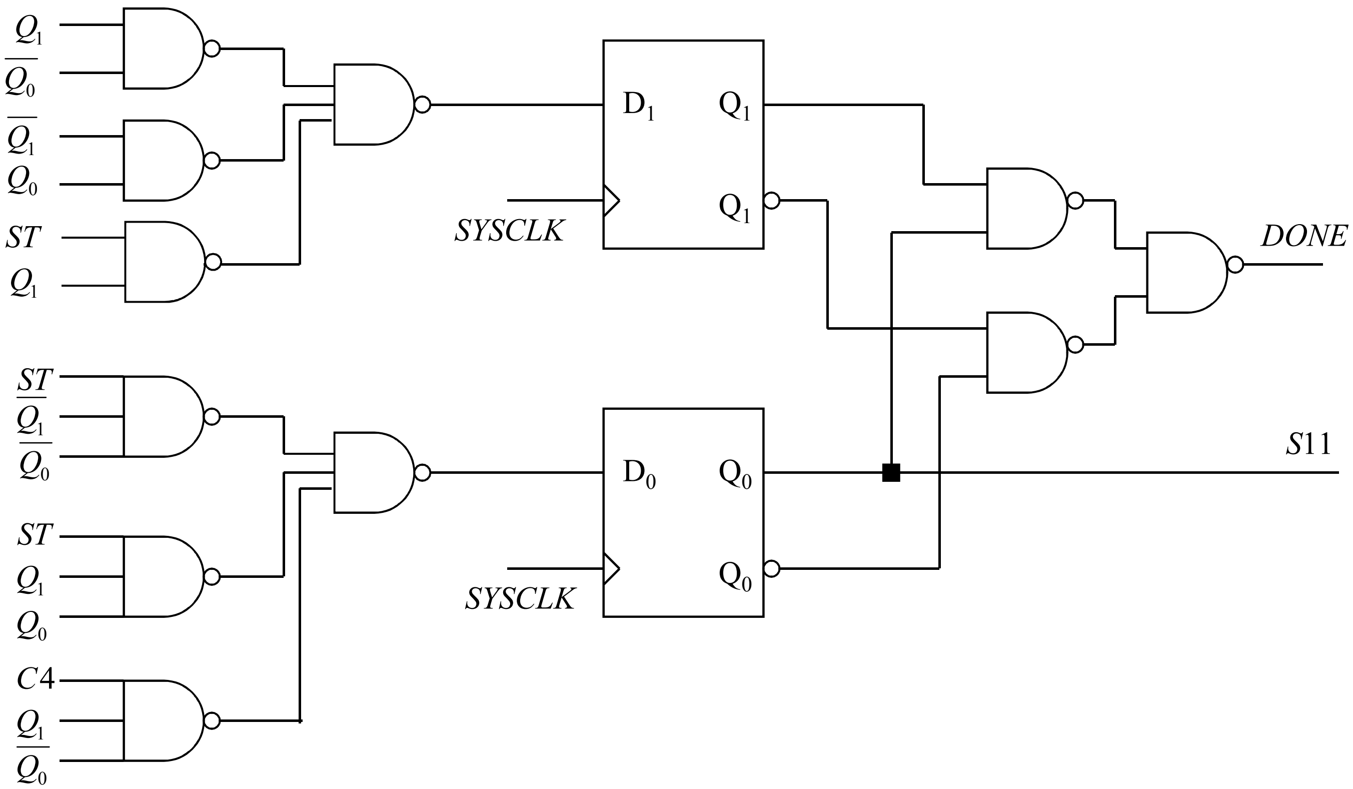
$D_0 = ST \cdot \overline{Q_1}\,\overline{Q_0} + ST \cdot Q_1Q_0 + C4 \cdot Q_1\overline{Q_0}$

| $Q_1 \backslash Q_0$ | 0 | 1 |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | 1 |

$DONE = \overline{Q_1}\,\overline{Q_0} + Q_1Q_0$

| $Q_1 \backslash Q_0$ | 0 | 1 |
|---|---|---|
| 0 | - | 1 |
| 1 | 0 | - |

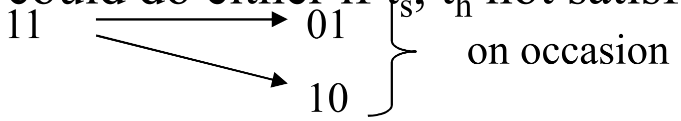
$S11 = Q_0$



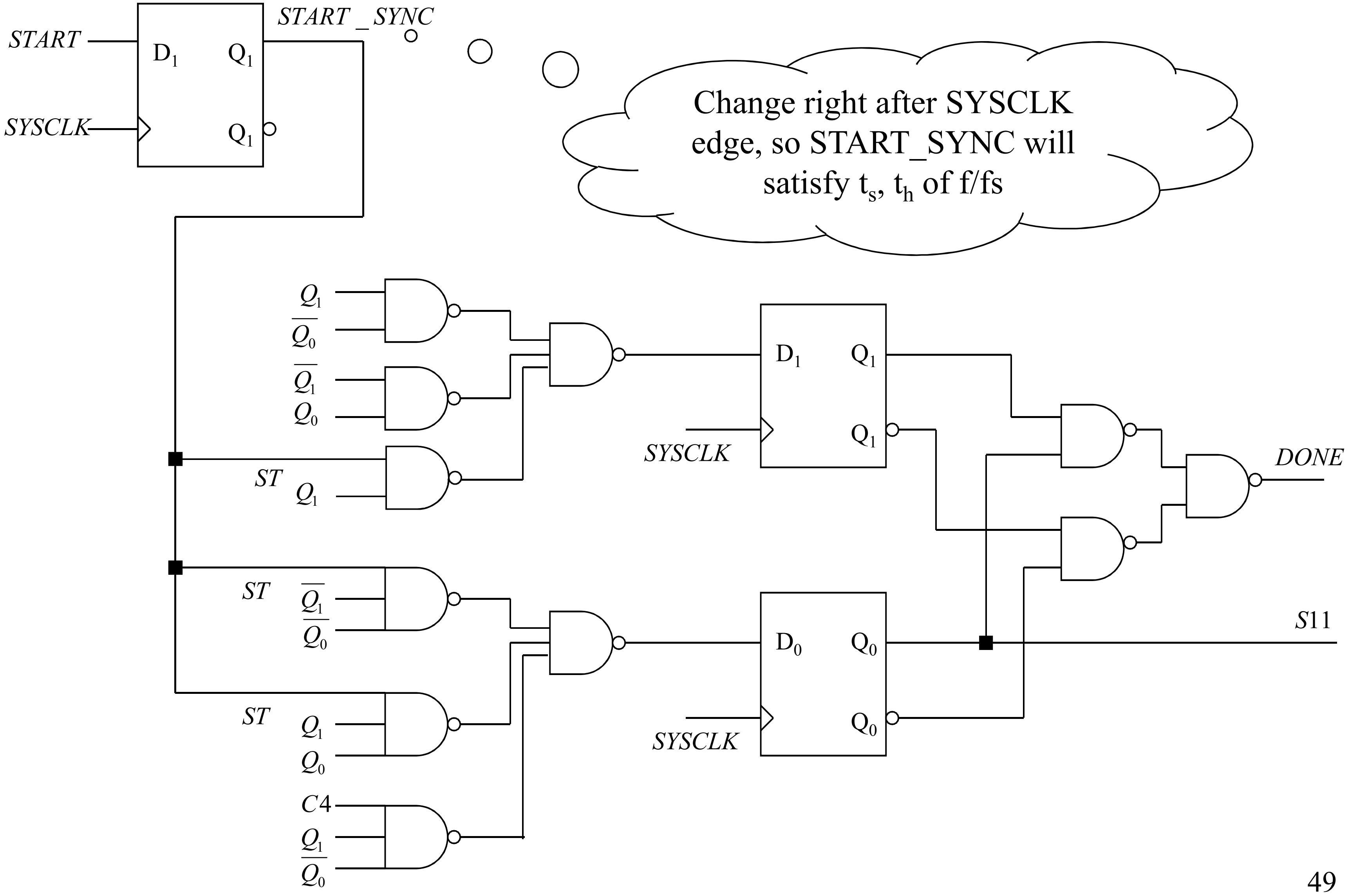
# Asynchronous START

- **Is START synchronous or asynchronous?**
  - Could be either
  - Assume asynchronous (comes from another system not using same SYSCLK)
- **Look at transition from “11” to “00” by negating START**
- **What happens if $t_s$, $t_h$ of D f/fs are not satisfied due to asynchronous START?**
  - Usually stay at 1 or go to 0
  - Problem? – early change to START=0 => “00”

    late change to START=0 => “11”
- **Problem: unexpected results can happen**
  - Delays not equal, f/fs different
  - Generally – could do either if $t_s$, $t_h$ not satisfied



# Synchronize START (Synchronizer)



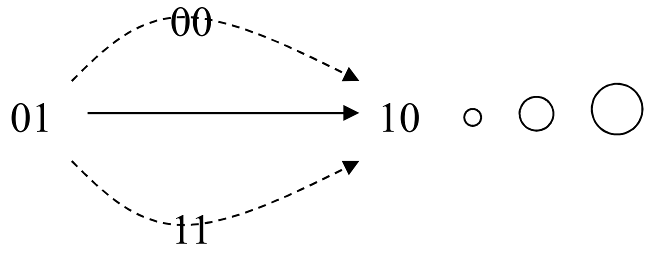
Change right after SYSCLK edge, so START_SYNC will satisfy $t_s$, $t_h$ of f/fs

# Additional Problem?

- **What else besides START_SYNC=1 or 0?**
- **Metastable – stuck in middle for a while**
  - What happen if START does not satisfy $t_s$, $t_h$ of "Synchronizer" D f/f
  - START_SYNC not 1 or not 0 for a while
- **Metastability – real problem** (early versions of several microprocessor chips had this problem!)
- **Synchronizer Failure and Metastability**
  - Solutions?

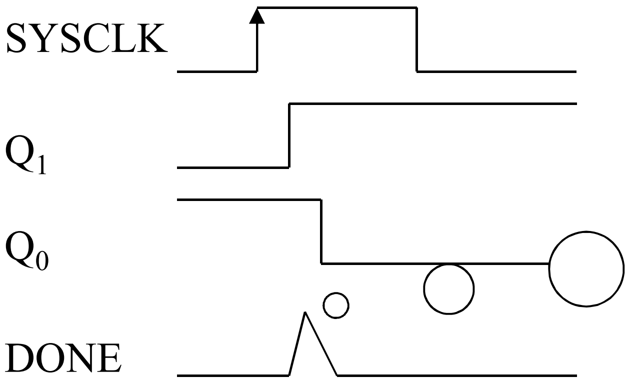
# Output Glitch on DONE

- **Look at transition between b="01" and c="10"**



One f/f change at a time because slightly different delay s

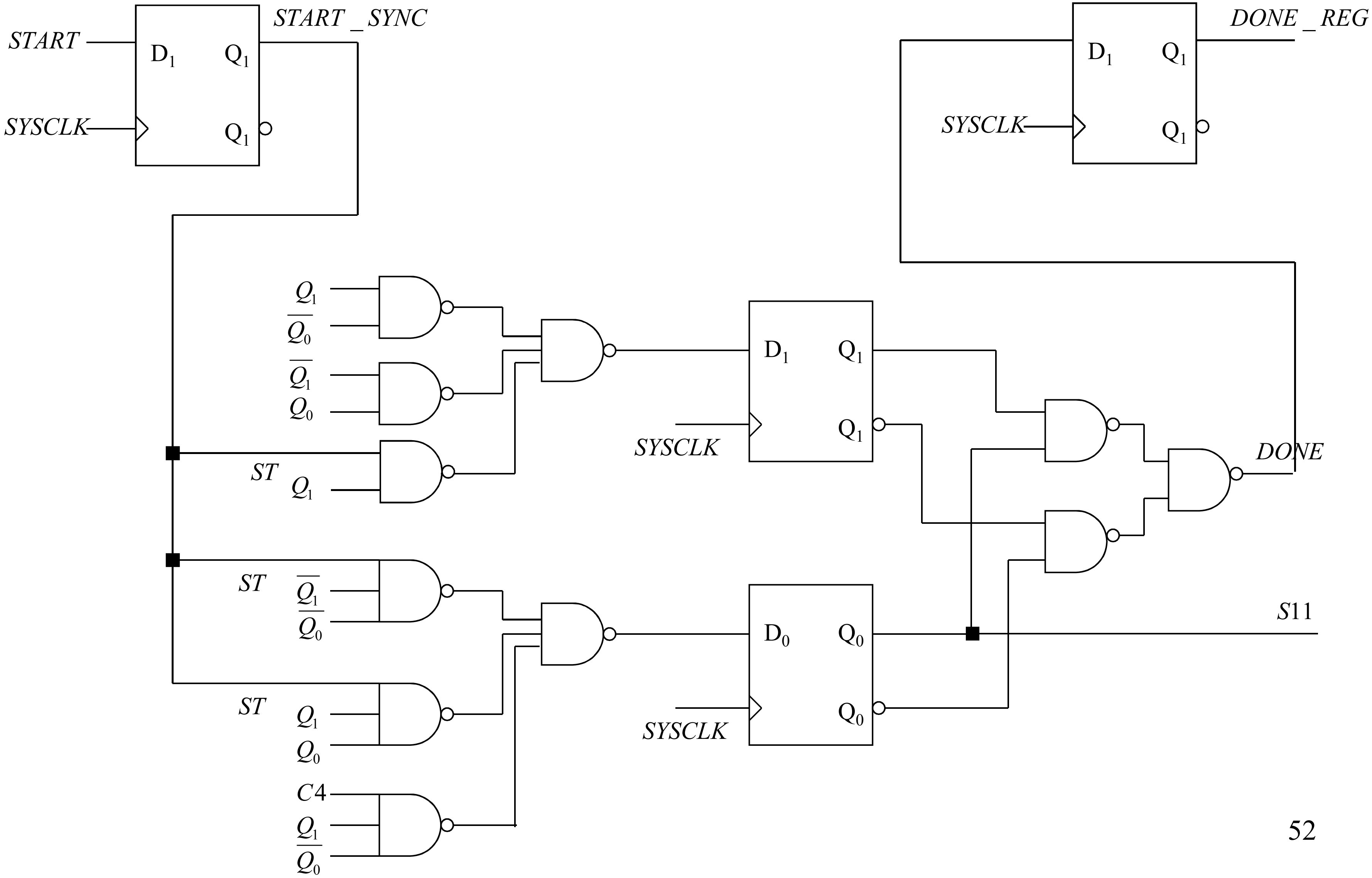
- **For a moment in the transition from "01" to "10"**
  - $Q_1Q_0$ = "00" or "11"
  - DONE=1 between states b ("01") and c ("10")



Problem? May or may not. DONE is used for synchronous inputs of other parts – not likely the problem What if DONE is used for asynchronous CLR of other parts?

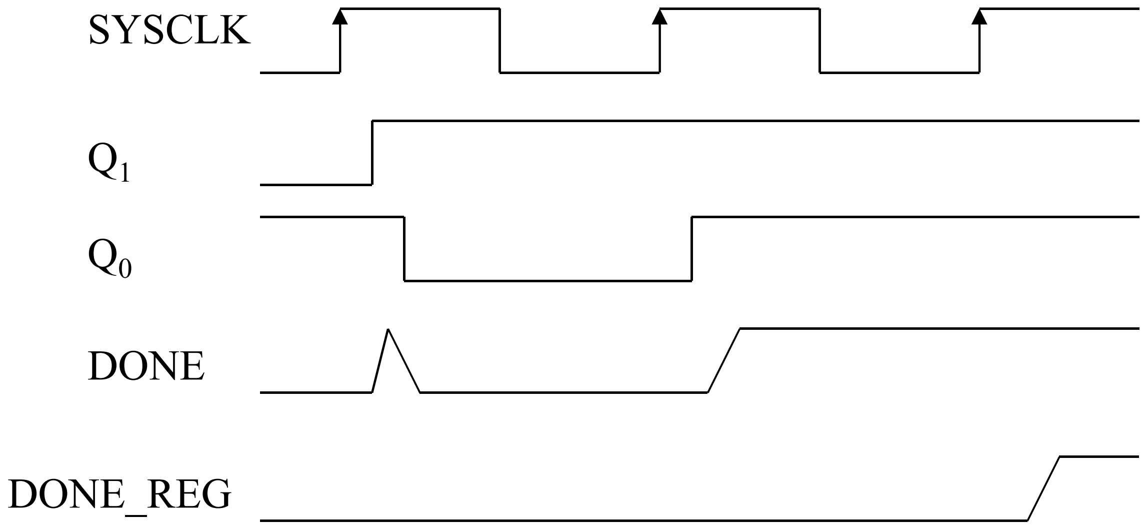
# Put a register (Stabilizer) on output

- **One D f/f on DONE**



# Work?

- **DONE_REG delayed – usually no problem**



- **Register output not always needed**
  - Good state assignment (compare this with our first state assignment)
  - Some good output logics (e.g., S11)

# Synchronous Design Methodology (Summary)

- **All LBBs and f/fs are clocked by the same common clock signal**
  - We use guaranteed LBBs and f/fs by the manufacturer (critical race free!!)
  - Glitches on combinational circuits connecting LBBs and f/fs have no effect, since the control inputs are sampled only after the glitches have had a chance to settle out
- **Three tasks to ensure reliable system operation**
  - Minimize and determine the amount of clock skew
  - Ensure that f/fs have positive setup- and hold-time margins
  - Identifying asynchronous inputs, synchronize them with the clock
  - Filter any problematic output glitches with output stabilizers