Design of Digital Systems II

Sequential Logic Design Practices (2)

Moslem Amiri, Václav Přenosil

Embedded Systems Laboratory
Faculty of Informatics, Masaryk University
Brno, Czech Republic

amiri@mail.muni.cz
prenosil@fi.muni.cz

Fall, 2014

- A **shift register** is an *n*-bit register with a provision for shifting its stored data by one bit position at each tick of clock

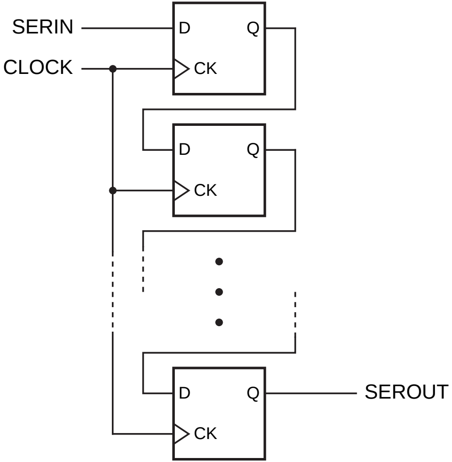

Figure 1: Structure of a serial-in, serial-out shift register.

Shift Registers: Shift-Register Structure

- Fig. 1 shows structure of a **serial-in, serial-out shift register**
  - Serial input, SERIN, specifies a new bit to be shifted into one end at each clock tick
  - This bit appears at serial output, SEROUT, after *n* clock ticks, and is lost one tick after
  - Thus, an *n*-bit serial-in, serial-out shift register can be used to delay a signal by *n* clock ticks

- A **serial-in, parallel-out shift register** has outputs for all of its stored bits, making them available to other circuits
  - Can be used to perform **serial-to-parallel conversion**

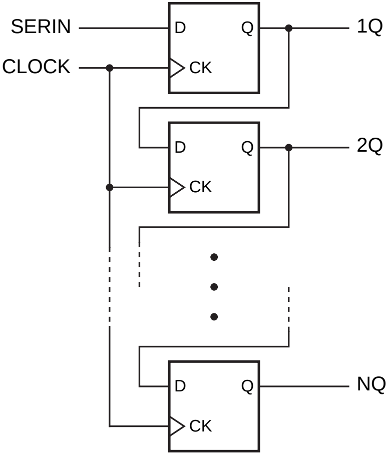

Figure 2: Structure of a serial-in, parallel-out shift register.

- It is possible to build a **parallel-in, serial-out shift register**
  - Can be used to perform **parallel-to-serial conversion**

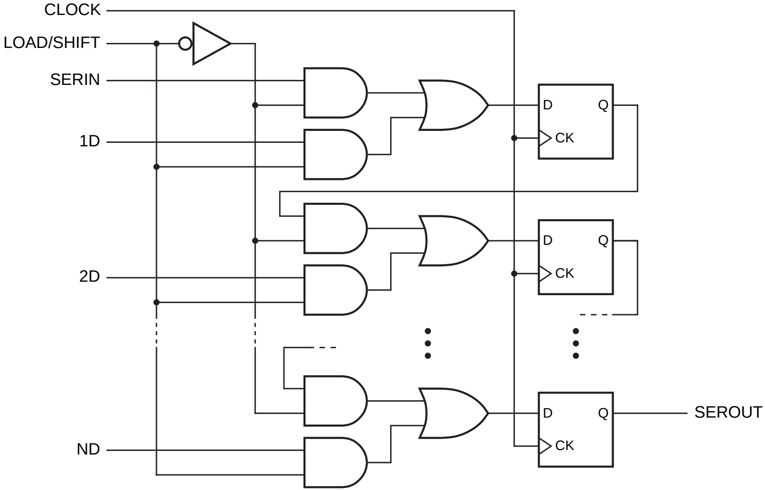

Figure 3: Structure of a parallel-in, serial-out shift register.

- By providing outputs for all of stored bits in a parallel-in shift register, **parallel-in, parallel-out shift register** is obtained

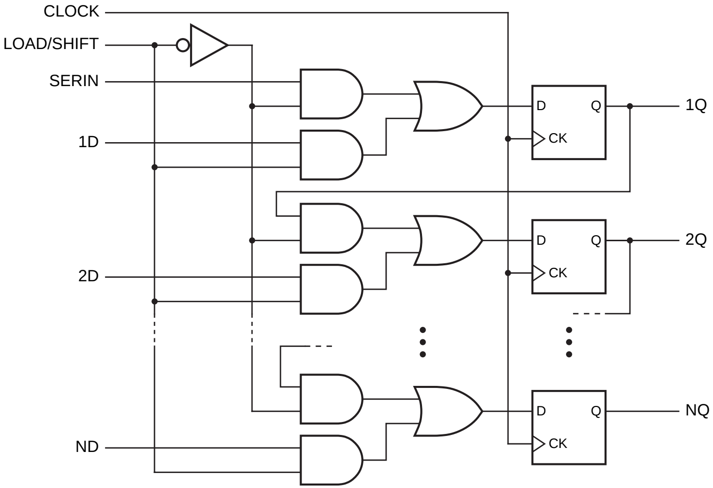

Figure 4: Structure of a parallel-in, parallel-out shift register.

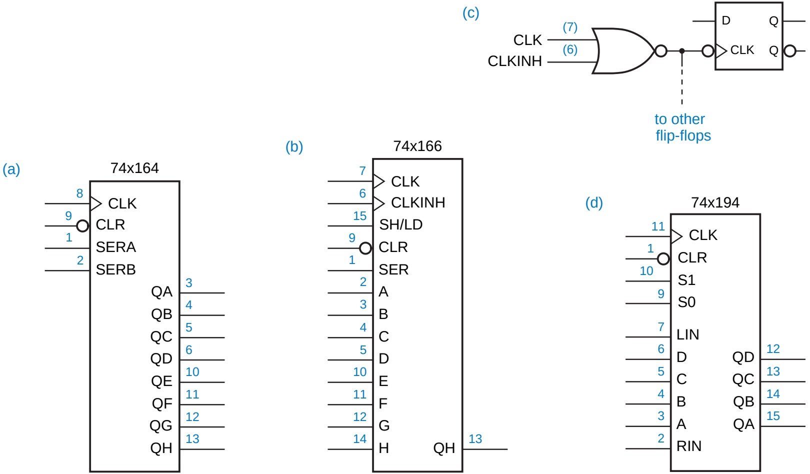

Figure 5: Traditional logic symbols for MSI shift registers: (a) 74x164 8-bit serial-in, parallel-out shift register; (b) 74x166 8-bit parallel-in, serial-out shift register; (c) equivalent circuit for 74x166 clock inputs; (d) 74x194 universal shift register.

- Fig. 5
  - 74x164
    - Is a serial-in, parallel-out device
    - Has an asynchronous clear input (CLR_L)
    - Has two serial inputs that are ANDed internally; both SERA and SERB must be 1 for a 1 to be shifted into first bit of register
  - 74x166
    - Is a parallel-in, serial-out shift register
    - Has an asynchronous clear input
    - Shifts when SH/LD is 1, and loads new data otherwise
    - Has an unusual clocking arrangement called a **gated clock**; it has two clock inputs that are connected to internal flip-flops
    - CLK is connected to a free-running system clock, and CLKINH is asserted to inhibit CLK, so that neither shifting nor loading occurs on next clock tick, and current register contents are held
    - CLKINH must be changed only when CLK is 1; otherwise, undesired clock edges occur on internal flip-flops
  - 74x194
    - Is an MSI 4-bit bidirectional, parallel-in, parallel-out shift register
    - Its logic diagram is shown in Fig. 6

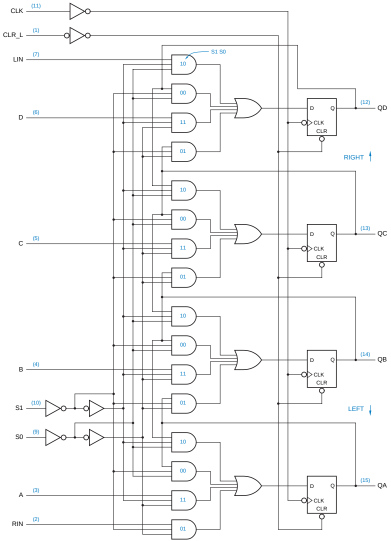

Figure 6: Logic diagram for the 74x194 4-bit universal shift register, including pin numbers for a standard 16-pin dual in-line package.

- 74x194
  - Is a **bidirectional shift register** because its contents may be shifted in either of two directions, depending on a control input
    - Shift registers studied previously are **unidirectional shift registers** because they shift in only one direction
  - The two directions
    - *left*: from QD to QA
    - *right*: from QA to QD
  - Is called a **universal** shift register because it can be made to function like any of less general shift register types

Table 1: Function table for the 74x194 4-bit universal shift register.

	Inputs		Next state			
Function	S1	S0	QA∗	QB∗	QC∗	QD∗
Hold	0	0	QA	QB	QC	QD
Shift right	0	1	RIN	QA	QB	QC
Shift left	1	0	QB	QC	QD	LIN
Load	1	1	A	B	C	D

Shift Registers: Serial/Parallel Conversion

- Most common application of shift registers
  - To convert parallel data into serial format for transmission or storage
  - To convert serial data back to parallel format for processing or display
- Example: **digital telephony**
  - Digital switching equipment is installed in central offices (COs)
  - Home phones have a two-wire analog connection to CO
  - An analog-to-digital converter samples analog voice signal 8000 times per second (once every 125 μs) when it enters CO
  - A/D converter produces a corresponding sequence of 8000 8-bit bytes representing sign and magnitude of analog signal at each sampling point
  - Voice is transmitted digitally on 64-Kbps **serial channels** throughout phone network
  - It is converted back to an analog signal by a digital-to-analog converter at far-end CO
  - 64 Kbps bandwidth required by one digital voice signal is far less than can be obtained on one digital signal line or switched by digital ICs
    - Digital telephone equipment **multiplexes** many 64-Kbps channels onto a single wire, saving both wires and digital ICs for switching
    - This is the main reason that telephone network has gone digital

Shift Registers: Serial/Parallel Conversion

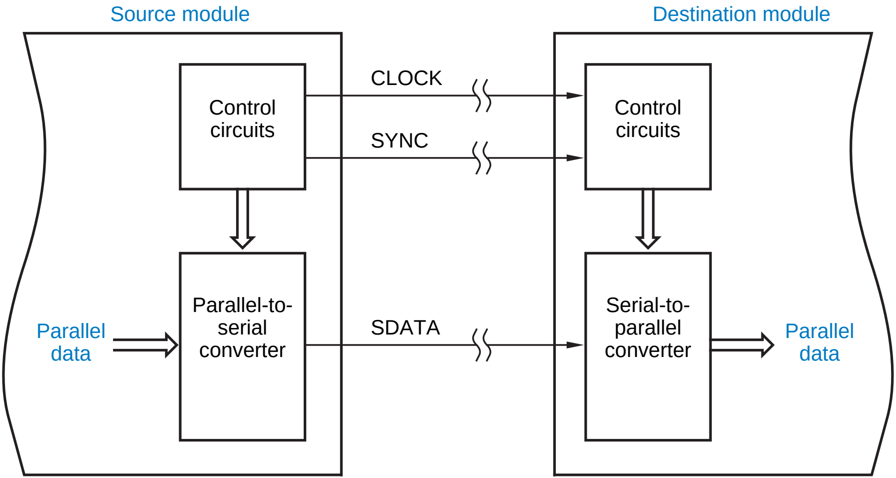

Figure 7: A system that transmits data serially between modules.

Shift Registers: Serial/Parallel Conversion

- Fig. 7
  - Clock signal
    - Provides timing reference for transfers, defining the time to transfer one bit
    - In systems with just two modules, clock may be part of control circuits located on source module
    - In larger systems, clock may be generated at a common point and distributed to all of modules
  - Serial data
    - Data itself is transmitted on a single line
  - Synchronization pulse (or sync pulse)
    - Provides a reference point for defining data format, such as beginning of a byte or word in serial data stream
    - Some systems omit this signal and instead use a unique pattern on serial data line for synchronization

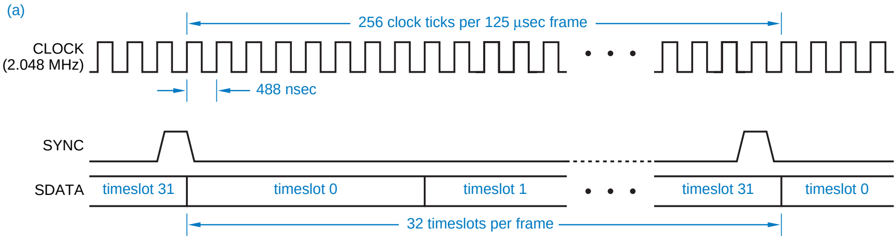

Figure 8: Timing diagram for parallel-to-serial conversion: a complete frame.

- General timing characteristics in a typical digital telephony application are shown in Fig. 8
  - CLOCK signal has a frequency of 2.048 MHz to allow transmission of 32×8000 8-bit bytes per second
  - 1-bit-wide pulse on SYNC signal identifies beginning of a 125-μs interval called a *frame*
    - A total of 256 bits are transmitted on SDATA during this interval, which is divided into 32 *timeslots* containing eight bits each
  - Each timeslot carries one digitally encoded voice signal
  - Both timeslot numbers and bit positions within a timeslot are located relative to SYNC pulse
- Fig. 9 shows a circuit that converts parallel data to serial format of Fig. 8, with detailed timing shown in Fig. 10

Shift Registers: Serial/Parallel Conversion

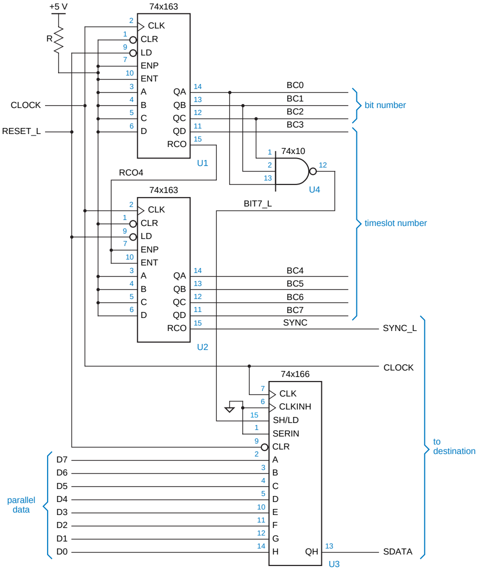

Figure 9: Parallel-to-serial conversion using a parallel-in shift register.

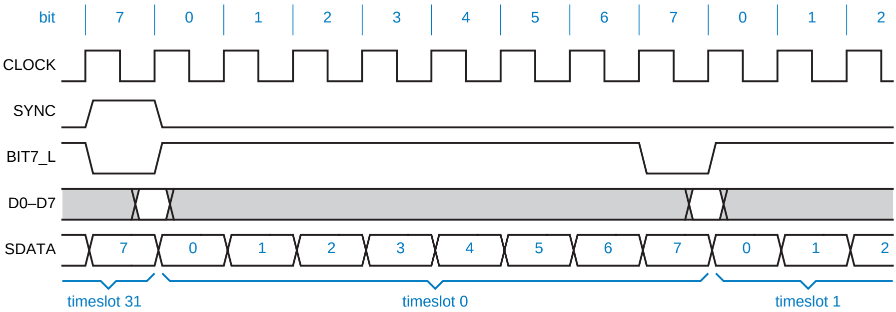

Figure 10: Timing diagram for parallel-to-serial conversion: one byte at the beginning of the frame.

Shift Registers: Serial/Parallel Conversion

- Figs. 9 and 10
  - Two 74x163 counters are wired as a free-running modulo-256 counter to define frame
    - Five high-order counter bits: timeslot number
    - Three low-order counter bits: bit number
  - A 74x166 parallel-in shift register performs parallel-to-serial conversion
    - Bit 0 of parallel data (D0-D7) is connected to '166 input closest to SDATA output, so bits are transmitted serially in the order 0 to 7
    - During bit 7 of each timeslot, BIT7_L signal is asserted, which causes '166 to be loaded with D0-D7
    - Value of D0-D7 is irrelevant except during setup- and hold-time window around clock edge on which '166 is loaded
- A destination module can convert serial data back into parallel format using circuit of Fig. 11

Shift Registers: Serial/Parallel Conversion

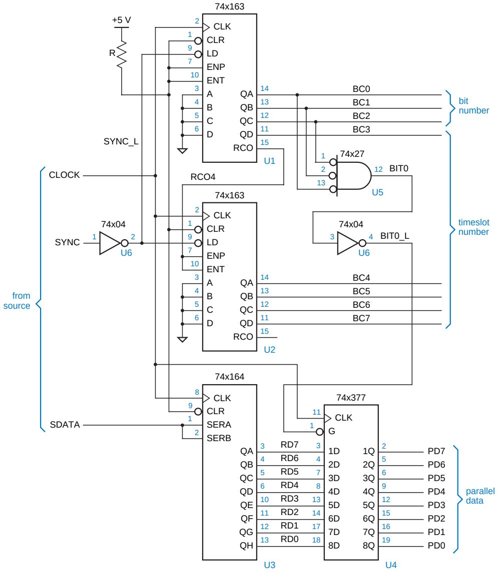

Figure 11: Serial-to-parallel conversion using a parallel-out shift register.

Shift Registers: Serial/Parallel Conversion

- Fig. 11
  - A modulo-256 counter built from a pair of '163s is used to reconstruct timeslot and bit numbers
  - Although SYNC is asserted during state 255 of counter on source module, SYNC loads destination module's counter with 0 so that both counters go to 0 on same clock edge
  - Counter's high-order bits (timeslot number) are not used here, but they may be used by other circuits in destination module to identify the byte from a particular timeslot on parallel data bus (PD0-PD7)

Shift Registers: Serial/Parallel Conversion

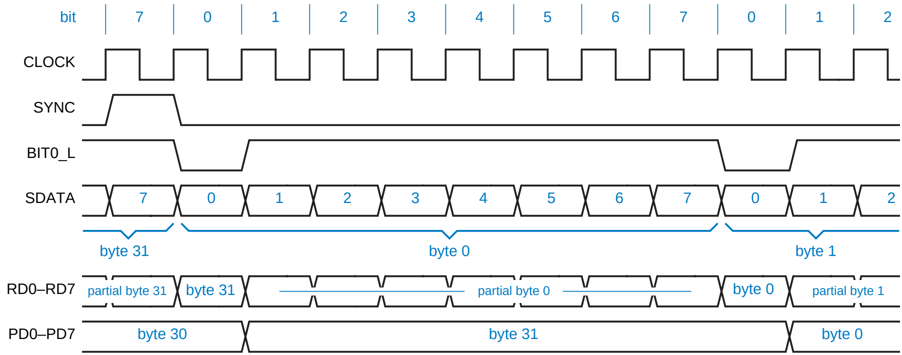

Figure 12: Timing diagram for serial-to-parallel conversion.

Shift Registers: Serial/Parallel Conversion

- Fig. 12
  - A complete received byte is available at parallel output of 74x164 shift register during clock period following reception of last bit (7) of byte
  - Parallel data in this example is **double-buffered**
    - Once it is fully received, it is transferred into a 74x377 register, where it is available on PD0-PD7 for eight full clock periods until next byte is fully recieved
  - BIT0_L signal enables '377 to be loaded at proper time

Shift Registers: Shift-Register Counters

- A shift register can be combined with combinational logic to form a state machine whose state diagram is cyclic
  - Such a circuit is called a **shift-register counter**
- A shift-register counter does not count in an ascending or descending binary sequence
  - But it is useful in many "control" applications

- **Ring counter**
  - Simplest shift-register counter which uses an n-bit shift register to obtain a counter with n states

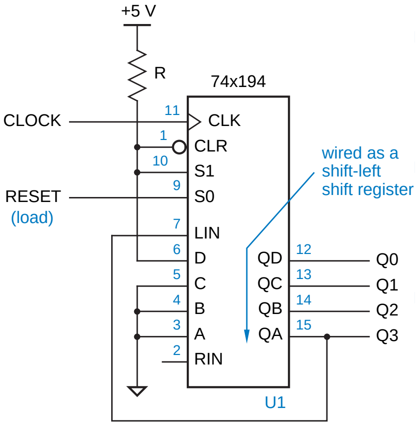

Figure 13: Simplest design for a four-bit, four-state ring counters with a single circulating 1.

- Fig. 13
  - 74x194 universal shift register is wired so that it normally performs a left shift
    - When RESET is asserted, it loads 0001
    - Once RESET is negated, '194 shifts left on each clock tick
  - LIN serial input is connected to "leftmost" output, so next states are $0010, 0100, 1000, 0001, 0010, \ldots$
    - Counter visits four unique states before repeating

Table 2: Function table for the 74x194 4-bit universal shift register.

	Inputs		Next state			
Function	S1	S0	QA*	QB*	QC*	QD*
Hold	0	0	QA	QB	QC	QD
Shift right	0	1	RIN	QA	QB	QC
Shift left	1	0	QB	QC	QD	LIN
Load	1	1	A	B	C	D

Shift Registers: Ring Counters

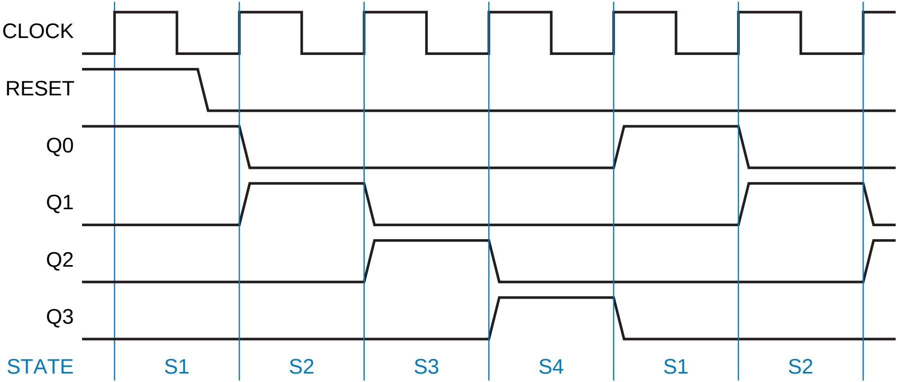

Figure 14: Timing diagram for a 4-bit ring counter.

- Ring counter in Fig. 13 is not robust
  - If its single 1 output is lost due to a temporary hardware problem, counter goes to state 0000 and stays there forever
  - Likewise, if an extra 1 output is set, counter will go through an incorrect cycle of states and stay in that cycle forever
  - Complete state diagram for counter circuit has 16 states, of which 12 states are not part of normal counting cycle

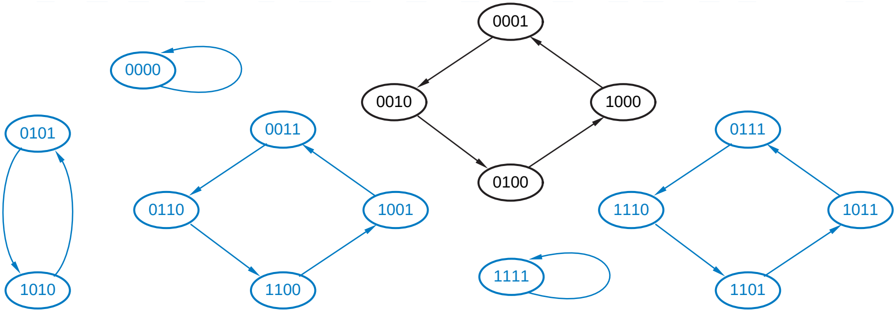

Figure 15: State diagram for a simple ring counter.

- A **self-correcting counter** is designed so that all abnormal states have transitions leading to normal states

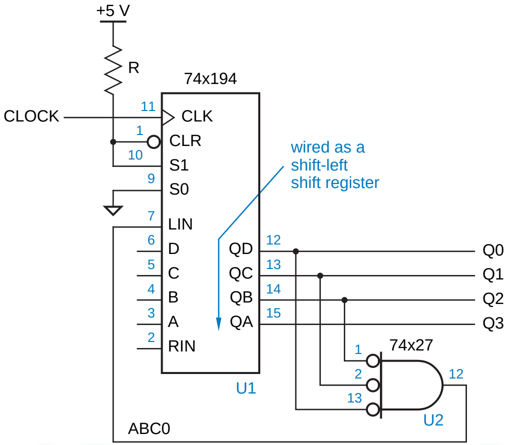

Figure 16: Self-correcting four-bit, four-state ring counter with a single circulating 1.

- Fig. 16
  - Circuit uses a NOR gate to shift a 1 into LIN only when three least significant bits are 0
  - An explicit RESET signal is not necessarily required
    - Regardless of initial state of shift register on power-up, it reaches state 0001 within four clock ticks
    - An explicit reset signal is required only if it is necessary to ensure that counter starts up synchronously with other devices in system or to provide a known starting point in simulation

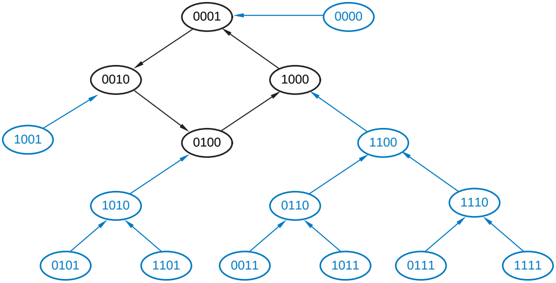

Figure 17: State diagram for a self-correcting ring counter.

- In general, an n-bit self-correcting ring counter uses an $n-1$-input NOR gate, and corrects an abnormal state within $n-1$ clock ticks
- In CMOS and TTL logic families, wide NAND gates are generally easier to come by than NORs

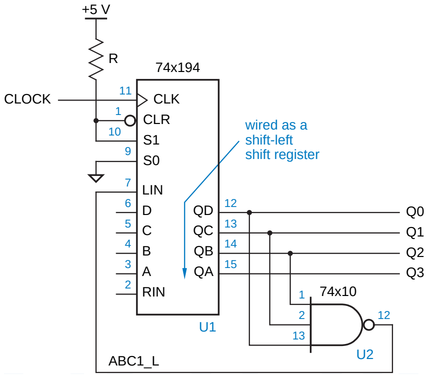

Figure 18: Self-correcting four-bit, four-state ring counter with a single circulating 0.

- Major appeals of a ring counter for control applications
  - Its states appear in 1-out-of-n decoded form directly on flip-flop outputs
    - Exactly one flip-flop output is asserted in each state
  - These outputs are "glitch free"
    - Compare with binary counter and decoder approach

- **Johnson counter**
  - An *n*-bit shift register with complement of serial output fed back into serial input with $2n$ states

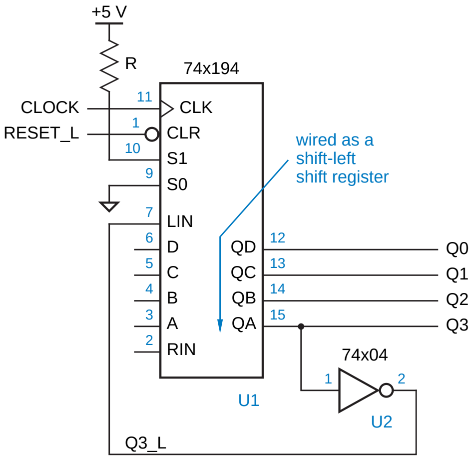

Figure 19: Basic four-bit, eight-state Johnson counter.

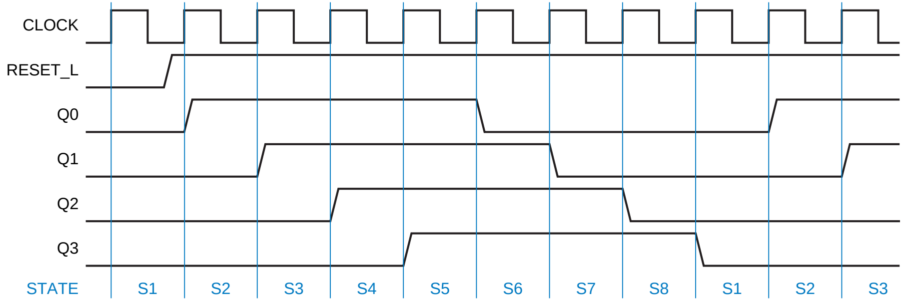

Figure 20: Timing diagram for a 4-bit Johnson counter.

Table 3: States of a 4-bit Johnson counter.

State Name	Q3	Q2	Q1	Q0	Decoding
S1	0	0	0	0	$Q3' \cdot Q0'$
S2	0	0	0	1	$Q1' \cdot Q0$
S3	0	0	1	1	$Q2' \cdot Q1$
S4	0	1	1	1	$Q3' \cdot Q2$
S5	1	1	1	1	$Q3 \cdot Q0$
S6	1	1	1	0	$Q1 \cdot Q0'$
S7	1	1	0	0	$Q2 \cdot Q1'$
S8	1	0	0	0	$Q3 \cdot Q2'$

- If both true and complemented outputs of each flip-flop are available, each normal state of counter can be decoded with a 2-input AND or NOR gate
  - Decoded outputs are glitch free

Shift Registers: Johnson Counters

- An n-bit Johnson counter has $2^n - 2n$ abnormal states, and is therefore subject to robustness problems

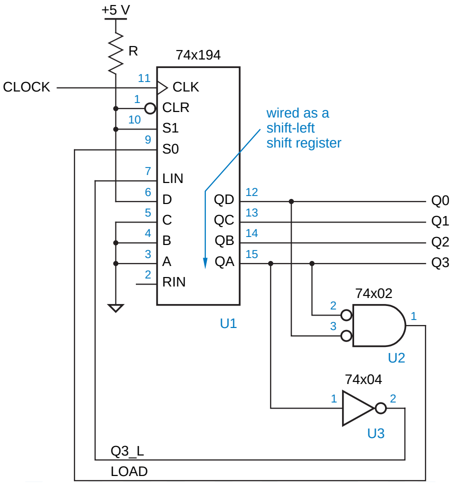

Figure 21: Self-correcting four-bit, eight-state Johnson counter.

Shift Registers: Johnson Counters

- Fig. 21
  - This circuit loads 0001 as next state whenever current state is $0xx0$
  - A similar circuit using a single 2-input NOR gate can perform correction for a Johnson counter with any number of bits
    - Correction circuit must load $00\ldots01$ as next state whenever current state is $0x\ldots x0$
  - Proof that this circuit corrects any abnormal state
    - An abnormal state can always be written in the form $x\ldots x10x\ldots x$, since the only states that can't be written in this form are normal states $00\ldots00, 11\ldots11, 01\ldots1, 0\ldots01\ldots1,$ and $0\ldots01$
    - Therefore, within $n-2$ clock ticks, shift register will contain $10x\ldots x$
    - One tick later it will contain $0x\ldots x0$, and one tick after that the normal state $00\ldots01$ will be loaded

Table 4: Verilog module for an extended-function 8-bit shift register.

```
module Vrshftreg ( CLK, CLR, RIN, LIN, S, D, Q );
  input CLK, CLR, RIN, LIN;
  input [2:0] S;
  input [7:0] D;
  output [7:0] Q;
  reg [7:0] Q;

  always @ (posedge CLK or posedge CLR)
    if (CLR == 1) Q <= 0;
    else case (S)
      0: Q <= Q;               // Hold
      1: Q <= D;               // Load
      2: Q <= {RIN, Q[7:1]};   // Shift right
      3: Q <= {Q[6:0], LIN};   // Shift left
      4: Q <= {Q[0], Q[7:1]};  // Shift circular right
      5: Q <= {Q[6:0], Q[7]};  // Shift circular left
      6: Q <= {Q[7], Q[7:1]};  // Shift arithmetic right
      7: Q <= {Q[6:0], 1'b0};  // Shift arithmetic left
      default Q <= 8'bx;       // should not occur
      endcase
endmodule
```

Table 5: Function table for an extended-function 8-bit shift register.

	Inputs			Next state							
Function	***S2***	***S1***	***S0***	***Q7****	***Q6****	***Q5****	***Q4****	***Q3****	***Q2****	***Q1****	***Q0****
Hold	0	0	0	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
Load	0	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Shift right	0	1	0	RIN	Q7	Q6	Q5	Q4	Q3	Q2	Q1
Shift left	0	1	1	Q6	Q5	Q4	Q3	Q2	Q1	Q0	LIN
Shift circular right	1	0	0	Q0	Q7	Q6	Q5	Q4	Q3	Q2	Q1
Shift circular left	1	0	1	Q6	Q5	Q4	Q3	Q2	Q1	Q0	Q7
Shift arithmetic right	1	1	0	Q7	Q7	Q6	Q5	Q4	Q3	Q2	Q1
Shift arithmetic left	1	1	1	Q6	Q5	Q4	Q3	Q2	Q1	Q0	0

- Ring counters are often used to generate multiphase clocks or enable signals in digital systems
- Fig. 22 shows a set of clock or enable signals that might be required in a digital system with six distinct phases of operation
  - Each phase lasts for two ticks of a master clock signal, MCLK, during which the corresponding active-low phase-enable signal Pi_L is asserted

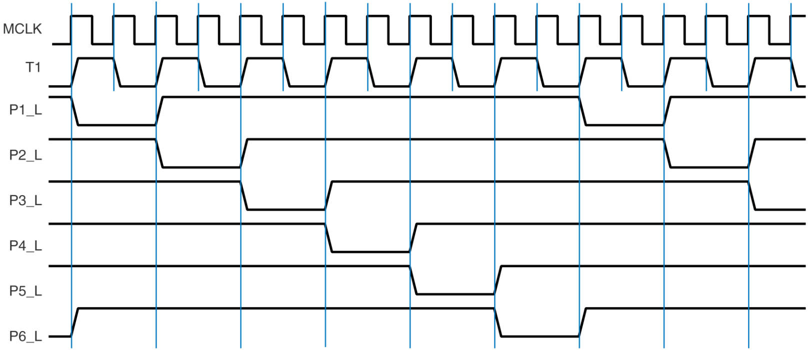

Figure 22: Six-phase timing waveforms required in a certain digital system.

- To obtain timing of Fig. 22 from a ring counter
  - We need an extra flip-flop to count the two ticks of each phase, so that a shift occurs on second tick of each phase
  - Three control inputs are provided
    - `RESET`: When this input is asserted, no outputs are asserted. Counter always goes to first tick of phase 1 after `RESET` is negated
    - `RUN`: When asserted, this input allows counter to advance to second tick of current phase, or to first tick of next phase; otherwise, current tick of current phase is extended
    - `RESTART`: Asserting this input causes counter to go back to first tick of phase 1, even if `RUN` is not asserted

Table 6: Verilog module for a six-phase waveform generator.

```
module Vrtimegen6 ( MCLK, RESET, RUN, RESTART, P_L );
  input MCLK, RESET, RUN, RESTART;
  output [1:6] P_L;
  reg [1:6] IP;  // internal active-high phase signals
  reg T1;        // first tick within phase

  always @ (posedge MCLK)
    if (RESET == 1) begin T1 <= 1; IP <= 6'b0; end
    else if ( (IP == 6'b0) || (RESTART == 1) )
      begin T1 <= 1; IP <= 6'b100000; end
    else if (RUN == 1)
      begin T1 <= ~T1; if (T1==0) IP <= {IP[6],IP[1:5]}; end
    else IP <= IP;

  assign P_L = ~IP;  // active-low phase outputs
endmodule
```

- A modification to previous application is to produce output waveforms that are asserted only during second tick of each two-tick phase
  - Shown in Fig. 23
  - One way to do this is to create a 12-bit ring counter and use only alternate outputs
    - Fig. 24

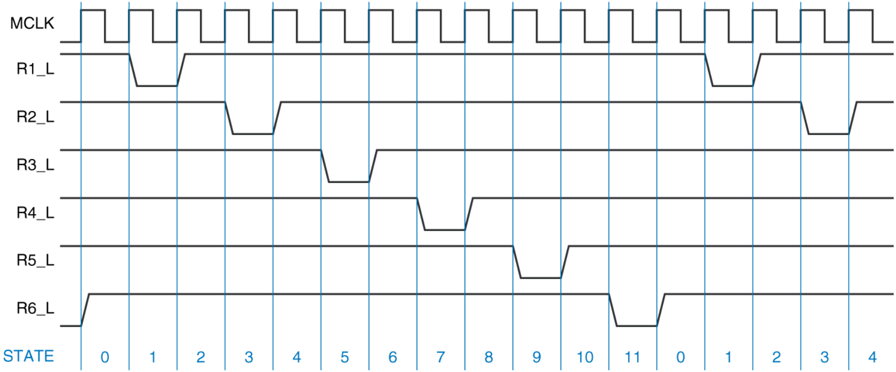

Figure 23: Modified timing waveforms for a digital system.

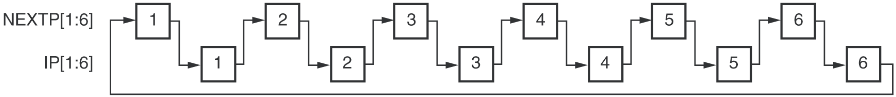

Figure 24: Shifting sequence for waveform generator 12-bit ring counter.

- Fig. 24
  - A 6-bit active-high variable, `IP`, is used for circuit's output, `P_L`
  - The extra six shift-register bits, `NEXTP[1:6]`, are interleaved with `P_L[1:6]`
  - Verilog module for this scheme is shown in Tab. 7

Table 7: Verilog module for a modified six-phase waveform generator.

```
module Vrtimegen12 ( MCLK, RESET, RUN, RESTART, P_L );
  input MCLK, RESET, RUN, RESTART;
  output [1:6] P_L;
  reg [1:6] IP;     // internal active-high phase signals
  reg [1:6] NEXTP;  // internal between-phase signals

  parameter IDLE = 6'b0, FIRSTP = 6'b100000;

  always @ (posedge MCLK)
    if (RESET == 1)
      begin IP <= IDLE; NEXTP <= IDLE; end                          // reset case
    else if ( ((IP == IDLE) && (NEXTP == IDLE)) || (RESTART == 1) )
      begin IP <= IDLE; NEXTP <= FIRSTP; end                        // restart case
    else if (RUN == 1)
      begin
        if ({IP,NEXTP} == 12'b0) begin IP <= IDLE; NEXTP <= FIRSTP; end // error case
        else begin IP <= NEXTP; NEXTP <= {IP[6],IP[1:5]}; end       // normal case
      end
    else begin IP <= IP; NEXTP <= NEXTP; end

  assign P_L = ~IP;  // active-low phase outputs
endmodule
```

Shift Registers in Verilog

- Tab. 7
  - In continuous-assignment statement, if we mistakenly used logical negation (`!`) in place of bitwise negation (`~`), resulting circuit would have all of its output bits except `P_L[6]` set to a constant 0
  - Statement `IP <= NEXTP` and then `NEXTP <= {IP[6],IP[1:5]}` near the end of sequential `always` block
    - Nonblocking assignments ensure that all of righthand sides are evaluated before any lefthand sides are changed
    - Thus, new value of `NEXTP` after clock tick is the shifted value of *previous* value of `IP` as desired
    - Previous `NEXTP` is assigned to `IP` at an infinitesimally short delta time *after* clock tick

References

John F. Wakerly, *Digital Design: Principles and Practices (4th Edition)*, Prentice Hall, 2005.