Experimenter's Corner

By Forrest M. Mims

Experimenting with Shift Registers

SHIFT registers are among the most versatile of digital logic circuits. This month, we'll cover the basics of shift register operation and design. We'll also look at some of the most important applications for shift registers. Next month, we'll look at some of the more important CMOS and TTL integrated shift registers. We will also present some application circuits you'll enjoy building.

The Basic Shift Register. Figure 1 is a block diagram of a very simple 4-bit shift register made from four D flip-flops connected in series. To understand the operation of this circuit, assume that the Q output of each flip-flop is at logic 0. When a clock pulse is applied to the SHIFT line, the logic level at the D input of each flip-flop is loaded into the corresponding flip-flop. Thus, if all of the Q outputs are initially at logic 0, the status of the four outputs (0000) will not be changed after the arrival of the clock pulse.

If a logic 1 is applied to the SERIAL INPUT, a logic 1 will be loaded into the first flip-flop when the next clock pulse arrives. The four-bit output nibble appearing at the PARALLEL OUTPUTS will then be 1000. If the logic level applied to the SERIAL INPUT is then changed to logic 0, the logic 1 will move one position to the right when the next clock pulse arrives. The four-bit nibble stored in the register will then be 0100. The rightward movement of the logic 1 will continue as additional clock pulses are received. The nibble changes to 0010 and then to 0001. Upon receipt of the fifth clock pulse, the logic 1 is pushed entirely out of the register and replaced by a logic 0. The register will then again contain the nibble 0000.

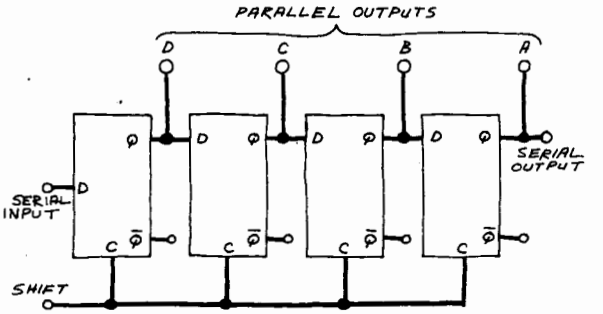


Fig. 1. Basic block diagram of a D flip-flop shift register.

Several significant things have occurred during the course of applying five clock pulses to the basic shift register. First, the logic 1 applied to the SERIAL INPUT appeared at the SERIAL OUTPUT only after the arrival of four clock pulses. Therefore, the shift register has functioned as a *digital delay line.* Secondly, the logic 1 migrated through the register, appearing at one of the four Q outputs at any given time in a sequence controlled by the clock rate. Taken together, the PARALLEL OUTPUTS can be used to actuate sequentially or *strobe* a series of external circuits in accordance with any pattern of bits presented to the SERIAL INPUT. In general, the ENABLE inputs of many logic ICs are active when a logic 0 is applied to them, so a logic 0 would usually be used as an activating strobe bit.

Thirdly, the bit pattern appearing at the four PARALLEL OUTPUTS can be considered a binary word. As the logic 1 moved from left to right, the magnitude of the word was halved at each clock pulse (1000 = 8; 0100 = 4; 0010 = 2 and 0001 = 1). Thus, the shift register performed a numerical divide-by-two operation. Finally, between clock pulses, the shift register has acted as a conventional data storage register. The register stored data without changing or modifying them, and data were always available at the PARALLEL OUTPUTS

Experimental Shift Register. Many different kinds of integrated shift registers are available, and we'll examine several of them next month. However, if you would like to build and experiment with your own flip-flop shift register, you can try the circuit shown in Fig.2. It is made from a pair of CMOS dual D flip-flops, and does everything the basic register of Fig. 1 does.

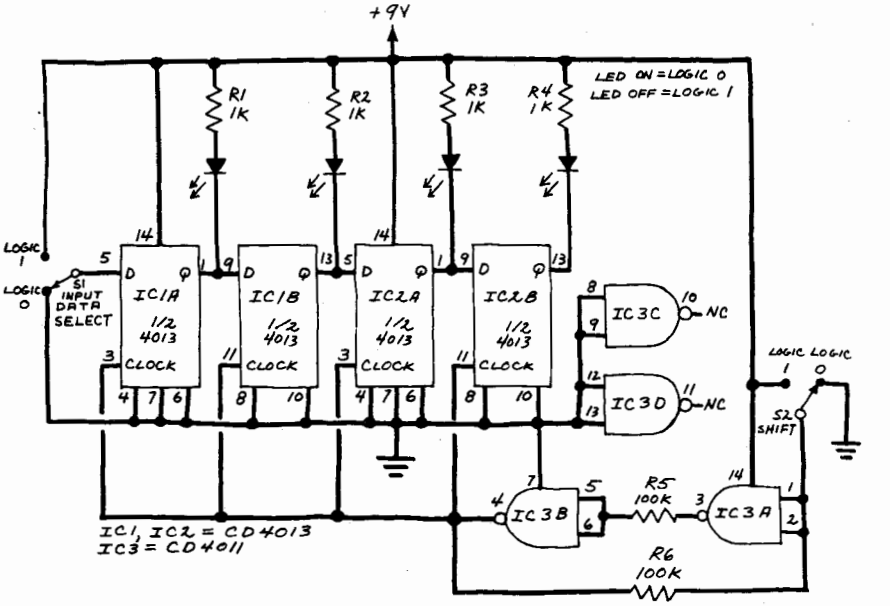


Fig. 2. Schematic diagram for an experimental CMOS shift register made from D flip-flops.

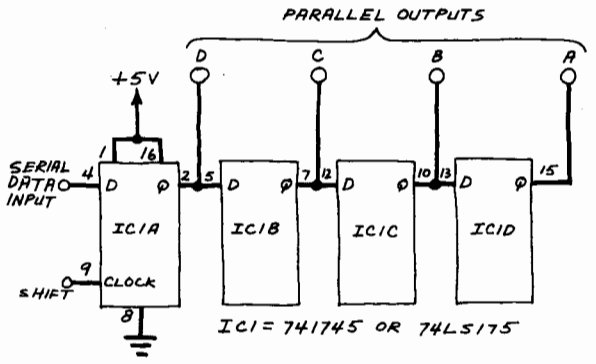


Fig. 3. A four-bit shift register made from a single quad D flip-flop.

Two NAND gates (*IC3A* and *IC3B*) connected as a bistable latch provide a bounce-free pulse to the clock inputs of each flip-flop when *S2* is placed in its LOGIC 1 position. This switch and the INPUT DATA SELECT switch allow you to cycle the shift register and change the input data in any fashion you choose. The logic level of each Q output is indicated by a LED.

If you prefer, you can use flip-flops other than those contained in the 4013 to make a shift register. For example, the 7474 is a TTL dual D flip-flop. The 74175 contains four D flip-flops in a single DIP and, as you can see in Fig. 3, readily lends itself to use as 4-bit TTL shift register.

Incidentally, if you don't have any D flip-flops on hand, but do have some JK flip-flops (such as the 4027, 7473, 7476, etc.), you can convert the JK units into D flip-flops. Simply connect the input and output of an inverter to the J and K inputs, respectively. The node comprising the flip-flop's J input ' the inverter input behaves as a data (D) input.

Shift Register Types. Now that we've seen what a basic shift register can do and how it does it, let's examine some of the technical jargon used to characterize various types of shift registers. First, that shown in Fig. 1 is called a *serial-in/parallel-out and serial-out* shift register. It is a *serial-in* register because data can be entered bit by bit (serially) into the input of only the first flip-flop. It is a *parallel-out* register since all four outputs are simultaneously available. Because the final output is always available, the circuit also provides a *serial-out* capability. A *parallel-in* capability is not available with the circuit in Fig. 1, but can be added with the help of a suitable logic network.

These descriptive terms allow us to specify the most important kinds of shift registers:

Serial-In/Serial-Out. This is the basic shift register. It can be as simple as a 2-bit register or as complex as a million-bit bubble memory.

Serial-In/Parallel-Out. This register is more flexible than a simple *serial-out* register because all of the contents of the register are always available.

Parallel-In and Serial-In/Serial-Out. Such a register allows all of the bits in a complete digital word to be loaded simultaneously and then clocked out one bit at a time.

Parallel-In and Serial-In/Parallel-Out and Serial-Out. This is the "complete" shift register. It can be used as a conventional data register or as a universal shift register.

Although the basic register shifts bits only to the right, some registers can shift bits in *both* directions. These are the most versatile of all shift registers. Figure 4 shows all the input, output and control lines of a 4-bit universal shift register.

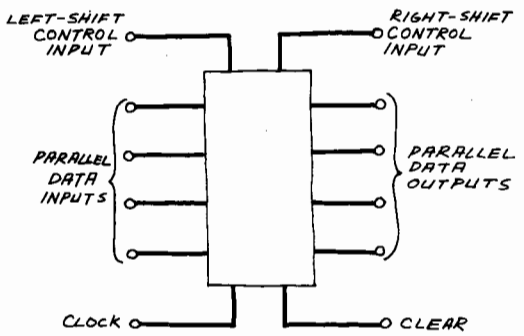


Fig. 4. Inputs and outputs for a hypothtical universal four-bit shift register.

Shift Register Applications. Shift registers have literally dozens of applications. In the remainder of this column we'll examine several important applications conceptually. We'll experiment with some specific circuits next month.

Multiplication. Shift registers are vital components in many digital computing circuits. Consider, for example, this problem in binary multiplication: Multiply 110_2 by 101_2.

```
  110 multiplicand
  101 multiplier
  ---
  110
 000    partial products
 110
 ----
 11110      final product.
```

The rules for binary multiplication are: (0) (0) = 0; (0) (1) = 0; (1) (0) = 0; and (1) (1) = 1. The rules for binary addition are: 0 + 0 = 0; 0 + 1 = 1; 1 + 0 = 1; and 1 + 1 = 0, carry 1, or 10.

Refer again to the multiplication problem above and you'll discover a binary-multiplication shortcut: When one bit in the multiplier is 0, its partial product is 000; when the bit is 1 the partial product equals the multiplicand. Therefore, to multiply two binary numbers, inspect the least significant bit in the multiplier. If it is 0, write down a string of 0s equal in length to the number of bits in the multiplicand. If it is 1, write down the multiplicand. This entry becomes the first partial product.

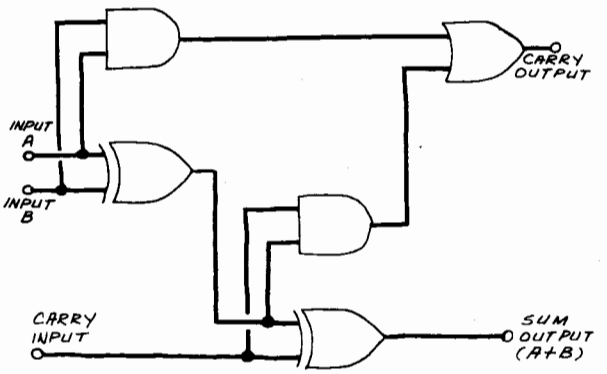


Fig. 5. Logic diagram of a binary full adder.

Next, move to the second-most significant bit in the multiplier. Repeat the foregoing procedure to arrive at the second partial product. Then shift the result one bit position to the left and add the two partial products.

Continue inspecting, shifting and adding until all the bits in the multiplier have been accounted for. The sum of the last two partial products becomes the final product.

This exercise illustrates a very important characteristic of digital arithmetic—binary multiplication can be accomplished by shifting left and adding. The arithmetic-logic unit (ALU) in virtually every microprocessor includes a logical comparator, an adder and a shift register. Multiplication can be performed by a relatively straightforward program that makes alternate comparisons, shifts and additions. If you would like to know more, Lou Frenzel has written a very clear explanation of this procedure in an excellent book, *Getting Acquainted with Microcomputers* (Howard W. Sams & Co., 1978, pp. 197-203).

Multiplication and Division by Two. Another neat binary-arithmetic trick that shift registers can perform is multiplication or division by a factor of two. As we have already

observed, shifting any binary word one bit to the right divides the word integrally by two. For example, 1110 (14) shifted right one bit is 0111 (7). Similarly, shifting any binary word one bit position to the left multiplies the word by two. For example, 1001 (9) shifted left one bit is 10010 (18).

Serial Addition. A binary *full adder* is a straightforward combinational circuit made from two exclusive -OR gates and several additional gates connected as shown in Fig. 5. The circuit is called a full adder since it can both accept and generate carry bits.

A single full adder can add only two data bits plus one carry bit. Therefore a number of adders arranged in parallel are required to simultaneously add all of the bits in two data words. For example, the simultaneous addition of all of the bits in two bytes requires a parallel array of eight full adders.

It's possible to add two data words using just one adder if the addition is performed one bit position at a time. Two shift registers are required to store the words being added, and a third is required to store the sum. A single D flip-flop is needed to store the carry bit which will result when the two bits to be added are both 1 or if the sum of the two bits and carry bit which might be present is 10 or 11. Figure 6 is a block diagram of a serial-shift-register adder.

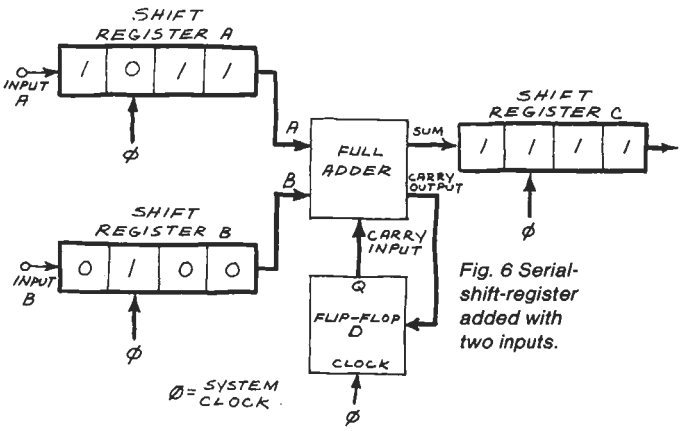


Fig. 6 Serial-shift-register added with two inputs.

The operation of a shift-register serial adder is a very good example of a sequential logic circuit. Referring to Fig. 6, the two words to be added are loaded into shift registers A and B. They are then clocked through the adder a pair of bits at a time and the resulting partial sums are loaded into shift register C. The complete addition requires only four clock cycles.

Can you think of a way to simplify the serial adder in Fig. 6? Shift register C can be eliminated entirely by feeding the output of the adder back to the input of Shift Register A, which then becomes an *accumulator*.

Although the operation of the serial adder seems simple enough, a control circuit is required to prevent the application of any more clock pulses once the addition has been completed. Otherwise, any new data that happens to be at the inputs of Shift Registers A and B will be cycled through the adder, and the sum stored in Register C will be pushed out and lost.

You can learn about an important aspect of the operation of the control section of a microprocessor or digital computer by designing a simple circuit. The circuit should monitor the operation of a serial adder and save the final sum by either disabling the clock pulses or moving the sum into still another register. Hint—the use of a 2-bit counter offers one solution.

Data Transmission. Computer data is usually transmitted in serial fashion one bit at a time. A shift register at the transmitting end reduces each word to be transmitted into its component bits, and one bit is transmitted each time a clock pulse arrives. A second shift register at the receiving end reconstructs the transmitted words bit by bit. It passes them to a storage register each time a complete word has been received and reconstructed. Figure 7 summarizes a shift-register data transmission system.

A shift register that transmits a word one bit at a time is called a *parallel-to-serial converter*. A shift register that assembles data words from a stream of incoming bits is called a *serial-to-parallel converter*. Both applications find use in many operations other than data transmission. Closely related to data transmission are applications in which a shift register acts as a temporary storage register or delays the arrival of a data word by a preselected number of clock pulses.

Memory Stack. A *memory stack* consists of two or more data registers used to hold temporary data. In a microcomputer, a stack is implemented within the main memory (RAM) or by a special set of data registers. A *pointer register* keeps track of where data is stored in the stack.

Shift registers can be used to make a memory stack. In the version shown in Fig. 8, four shift registers are arranged in

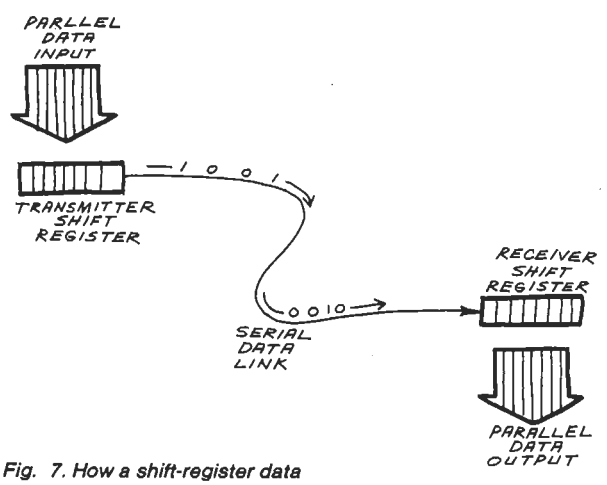


Fig. 7. How a shift-register data transmission system operates.

parallel so that up to four 4-bit data words or nibbles (half of an 8-bit byte) can be stored. The clock (shift) lines of all four registers are tied together so that a 4-bit nibble can be loaded into the stack in one operation. The nibble can then be pushed down into the stack as more are loaded.

If the nibble moves in only one direction through the stack, the first nibble to enter is the first to exit. This is a FIFO (first in/first out) stack. Several variations are possible. For example, the capability of shifting in both directions means that a nibble can be pushed into and popped out of the stack. In a LIFO (last in/first out) stack, the last nibble pushed into the stack is the first to be popped out of the stack.

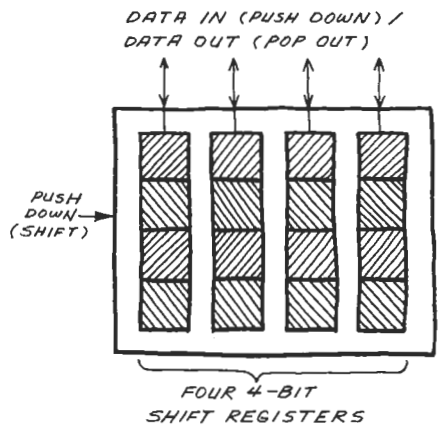


Fig. 8. Four 4-bit words can be stored in this memory stack.

eader's Letters. Several readers have sent comments that many followers of this column might find helpful. R.C. Amendola, for example, detected an error in the circuit of the six-digit event counter described in the February 1980 "Experimenter's Corner." In Fig. 3 (p. 100), the strobe signals to *Q1-Q3* must be inverted for the display to work properly. Mr. Amendola suggests inserting inverters in the strobe lines. A better way is to do what I did in the prototype version and use pnp transistors for *Q1-Q3*. To do this, you'll need to reverse the collector and emitter connections of each transistor, as they appear in the schematic.

Incidentally, this same error appears on page 35 in the first printing of *Engineer's Notebook*, a circuit sourcebook I recently wrote for Radio Shack. Subsequent printings include the corrected circuit.

Noah T.W. Givens, a former research technician at Bell Laboratories in Norcross, GA, wrote to reaffirm my warning about the hazards of working with glass fibers (May 1980, p. 86). Mr. Givens states," . . . if you place (a) fiber on your index finer to score it, you risk getting an extremely nasty piece of optical fiber in your finger. A worse splinter you'll never find. Besides being so very small in diameter, the thing is virtually invisible."

He also suggests using a fresh razor blade to prepare fiber for cleaving. He doesn't like the word *score* as that implies "dragging the blade across the fiber." He points out all that's necessary is simply to *touch* the blade to the fiber at the desired point of separation and then apply tension to separate the fiber.

Finally, some readers continue to make requests for custom circuit designs or detailed information about specialized technical topics. Because of the great volume of mail I receive, it is not possible to repond personally to such requests. Nevertheless, I very carefully read all letters and I will consider escribing in a future column those topics or circuits which ppear to have wide reader interest. Letters pointing out errors receive prompt attention.

In short, although individual replies are impracticable, your suggestions, criticisms and comments about this column are always welcome. ◇

One NOR gate starts shift-register loop

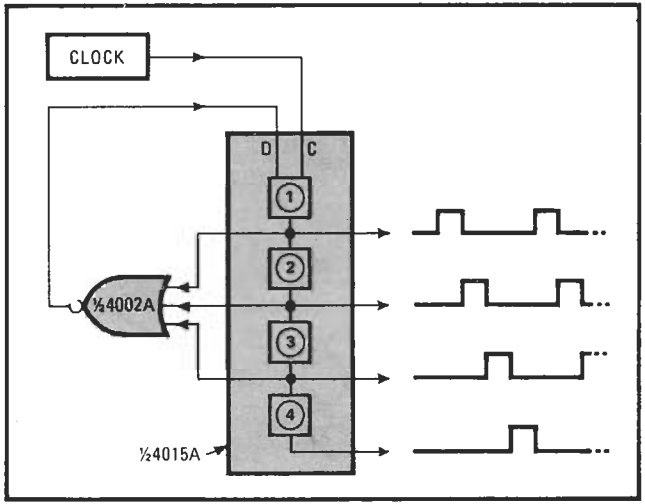
by Jean-Pierre Dujardin
Ohio State University, Columbus, Ohio

A circulating shift register with a single logic 1 in the loop is required in cyclic-triggering operations such as sampling transducers in time-sharing telemetry. Systems for starting this type of circuit are often complex, but the arrangement shown here simply uses a NOR gate with the four-stage shift register.

As the waveforms show, the output terminals of the 4015A shift register go high in a continuing sequence from stage one through stage four and then back to stage one again. The 4002A three-input NOR gate starts this operation and keeps it going.

The input terminals of the NOR gate are connected to the first three output terminals of the shift register. When these terminals are at logic 0, the output terminal of the gate is at logic 1, which is brought to the data input terminal (D) of the register. The next clock pulse transfers the logic 1 at D into the first stage of the register. When at least one of the inputs to the gate is a logic 1, the output from the gate is a 0, which is presented to the register input. Thus, after a maximum of three clock pulses, a single 1 is circulating.

This circuit requires no external timing to introduce the single 1 into the loop and no resetting. If external noise introduces errors, they are automatically corrected. Extension of the system to more than four shift-register stages is straightforward: outputs from all but the last stage are fed into a NOR gate that, in turn, feeds the D input of the first stage in the register. □



C-MOS ring circuit. Arrangement of NOR gate and four-stage shift register provides a pulse output that circulates to each of the output terminals in sequence, moving from one stage to the next as the clock cycles. The two C-MOS ICs determine performance level.

Multiplexer does double duty as dual-edge shift register

by James A. Mears
Dallas, Texas

Useful and sometimes unusual logic functions may be found disguised as parts with other functional names, and they often offer circuit advantages not found in the part designed for a given task. A good example of this principle is the 74157 quad two-input multiplexer, which can serve in a pinch as a clocked shift register.

Unlike other registers that advance on one edge of the clock, it will shift on each successive transition. This attribute is particularly useful when it is necessary to process data at twice the clock frequency without using a frequency doubler.

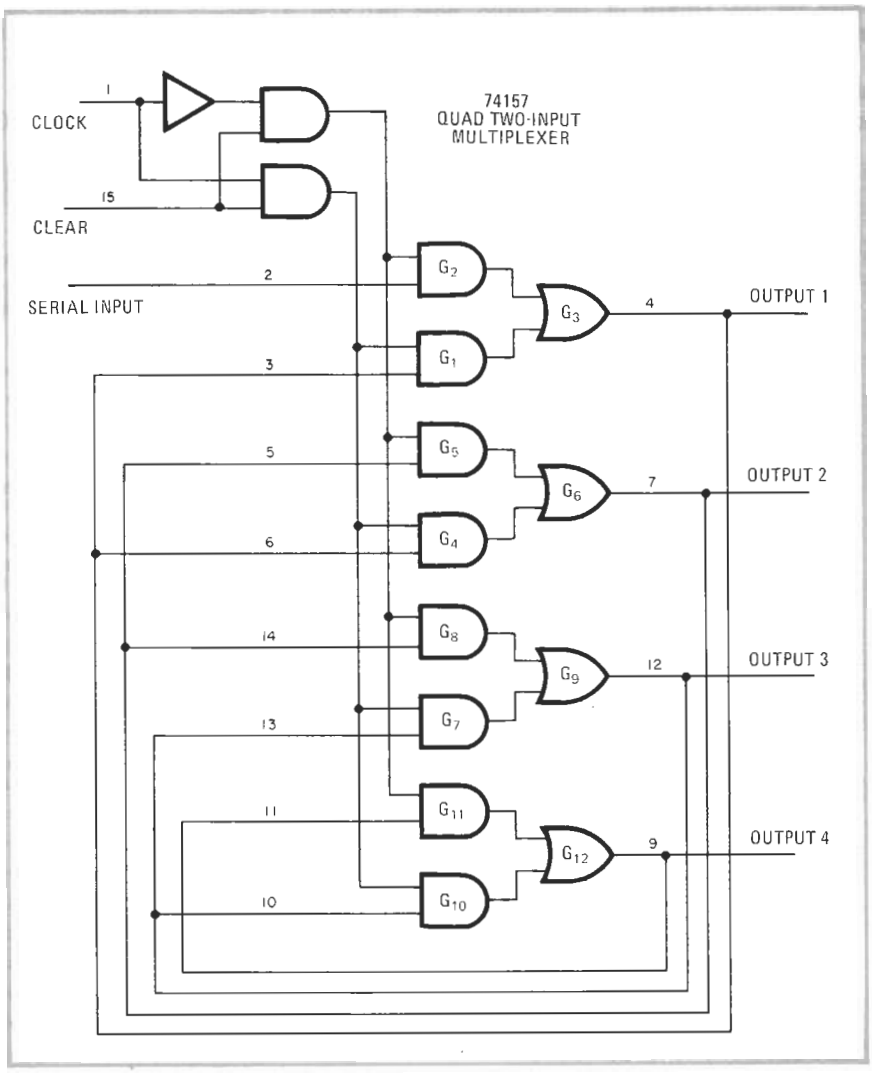
A good application for this type of register would be in the timing chain of a dynamic memory controller, where it could give a fine timing resolution without the need for a high-frequency clock. The pin functions in the figure have been renamed to give a better understanding of the circuit operation of the modified multiplexer.

Signals applied to the serial input are routed to output 1 through the AND-OR gate combination, G_1 through G_3, with output 1 connected to the alternate input of G_1 and to the corresponding input of the next stage, G_4 through G_6. The last three stages are connected similarly to form a chain of D latches (another good use for the 74157).

Thus, when the clock is high, the input signal will pass to the output, back to the alternate input and to the next stage. On the low-going clock edge, the output of the first stage is latched, and the signal is passed to the next stage's output. This process is repeated for each clock transition, thereby shifting the signal through the register. At any time, the register can be disabled (all outputs low) by bringing the clear input high.

Variations of this circuit can be built with the 74257, which is equivalent to the 74157 but has three-state outputs. The circuit can also be built with the 74158 (74157 with inverted outputs) by adding inverters to the outputs and connecting them up as before. Registers may be cascaded by connecting the serial input to the preceding stage's output. □

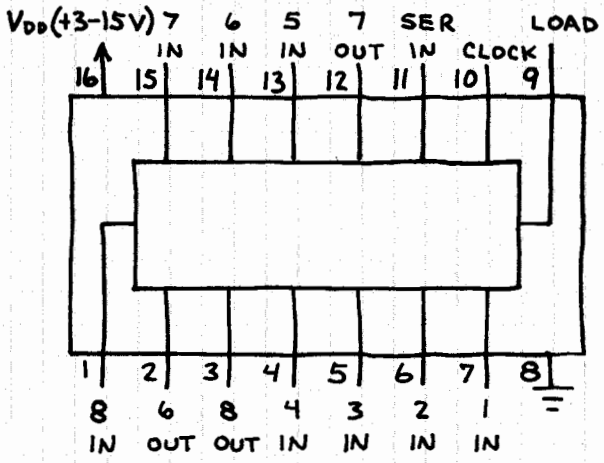
Designer's casebook is a regular feature in *Electronics*. We invite readers to submit original and unpublished circuit ideas and solutions to design problems. Explain briefly but thoroughly the circuit's operating principle and purpose. We'll pay $75 for each item published.



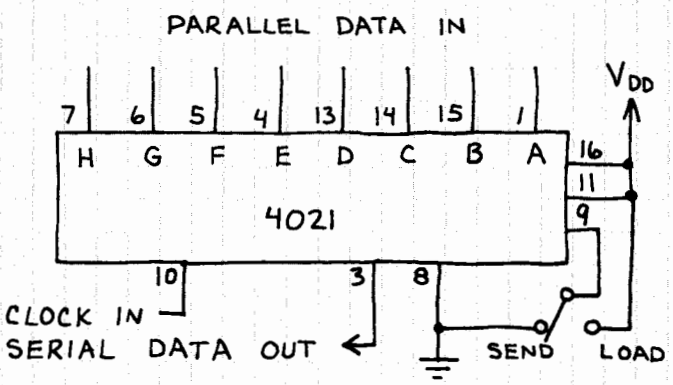
Disguise. Many logic chips can simulate other functions, such as this 74157 quad multiplexer, which performs well as a shift register. This part also gives the added advantage of dual-edge clocking, thus eliminating the need for frequency doublers in such circuits as the timing chain of high-resolution dynamic memory controllers.

8-STAGE SHIFT REGISTER 4021

PARALLEL INPUT / SERIAL OUTPUT SHIFT REGISTER. ALSO SERIAL INPUT. DATA AT PARALLEL INPUTS IS FORCED INTO THE REGISTER IRRESPECTIVE OF THE CLOCK STATUS WHEN PIN 9 IS MADE HIGH. KEEP PIN 9 LOW FOR NORMAL OPERATION.

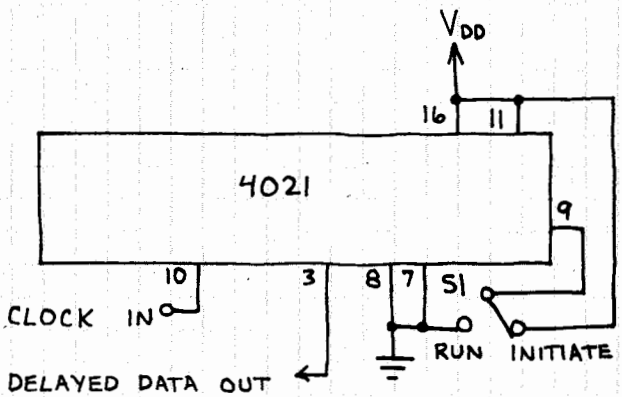


PARALLEL-TO-SERIAL DATA CONVERTER



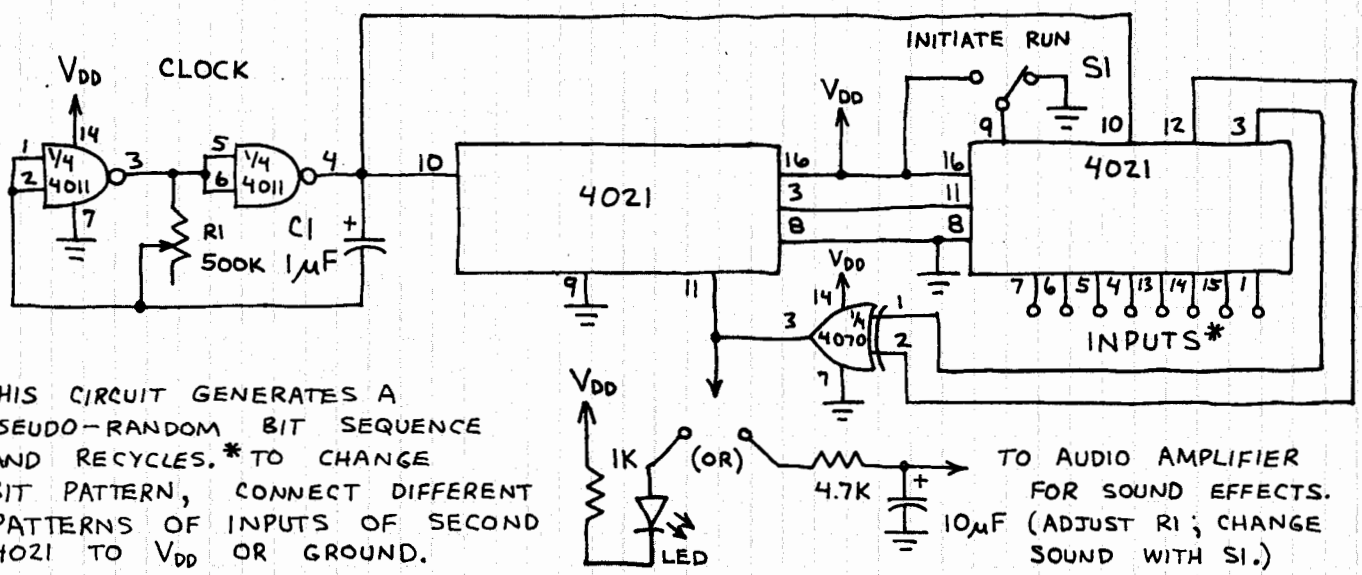
ALL 1's (H's) ARE SENT AFTER THE 8-BIT WORD IS TRANSMITTED.

8-STAGE DELAY LINE



THE FIRST PARALLEL INPUT (PIN 7) IS GROUNDED. THIS LOADS A SINGLE L WHEN SI IS SWITCHED TO INITIATE. THE SINGLE L BIT REACHES THE OUTPUT AFTER 8 CLOCK PULSES.

PSEUDO-RANDOM SEQUENCER

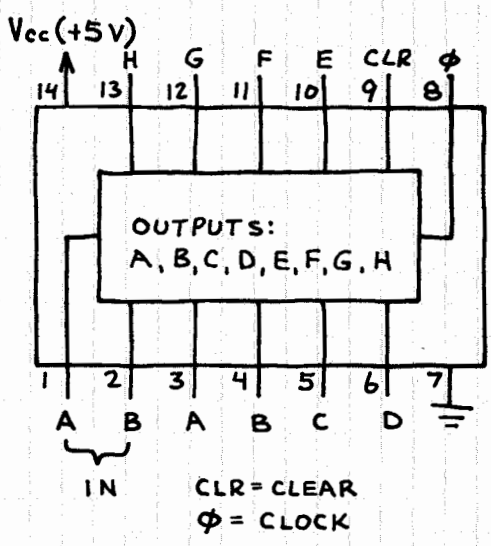


THIS CIRCUIT GENERATES A PSEUDO-RANDOM BIT SEQUENCE AND RECYCLES. * TO CHANGE BIT PATTERN, CONNECT DIFFERENT PATTERNS OF INPUTS OF SECOND 4021 TO V_{DD} OR GROUND.

TO AUDIO AMPLIFIER FOR SOUND EFFECTS. (ADJUST R1; CHANGE SOUND WITH SI.)

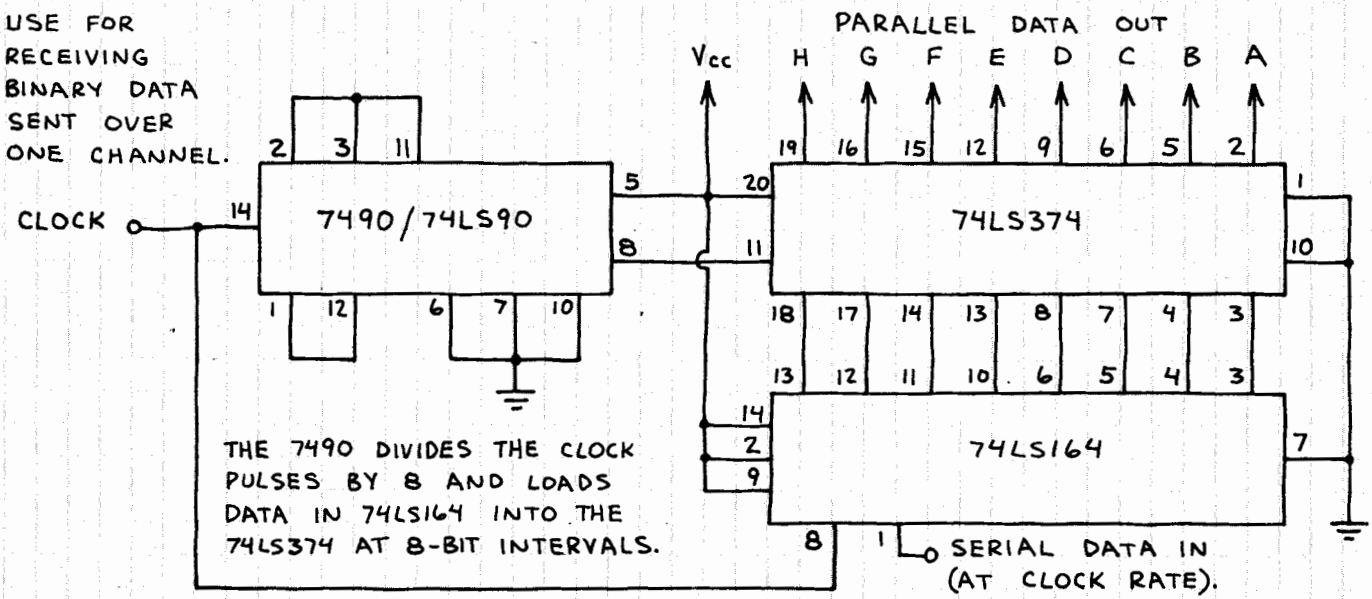
8-BIT SHIFT REGISTER 74LS164

DATA AT ONE OF THE TWO SERIAL INPUTS IS ADVANCED ONE BIT FOR EACH CLOCK PULSE. DATA CAN BE EXTRACTED FROM THE 8 PARALLEL OUTPUTS OR IN SERIAL FORM AT ANY SINGLE OUTPUT. ENTER DATA AT EITHER INPUT. THE UNUSED INPUT MUST BE HELD HIGH OR CLOCKING WILL BE INHIBITED. MAKING PIN 9 LOW CLEARS THE REGISTER TO LLLL.

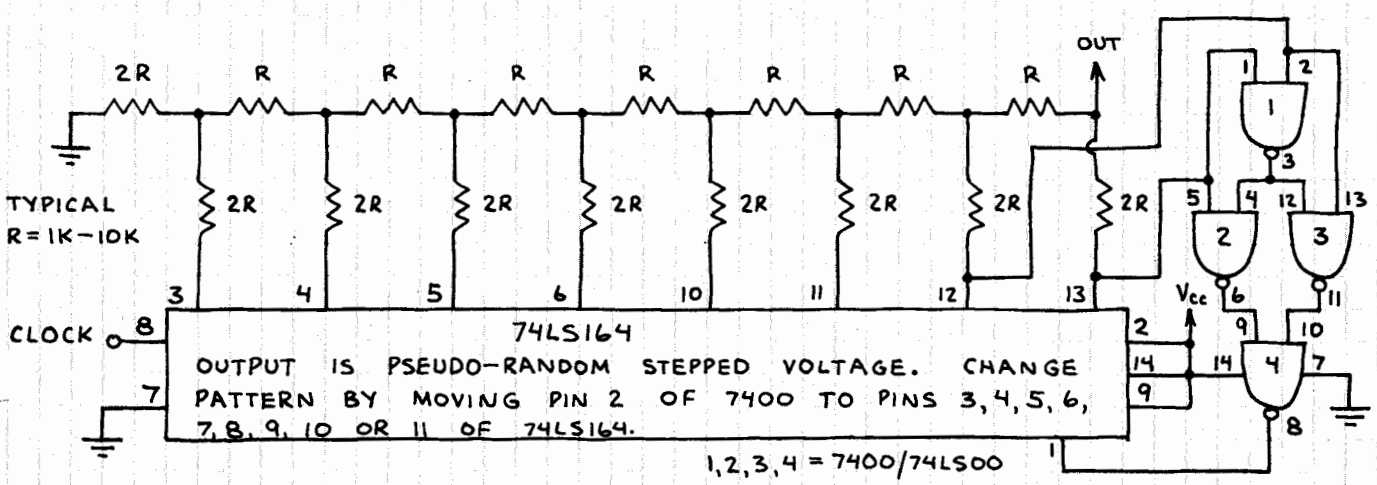


8-BIT SERIAL-TO-PARALLEL DATA CONVERTER

USE FOR RECEIVING BINARY DATA SENT OVER ONE CHANNEL.

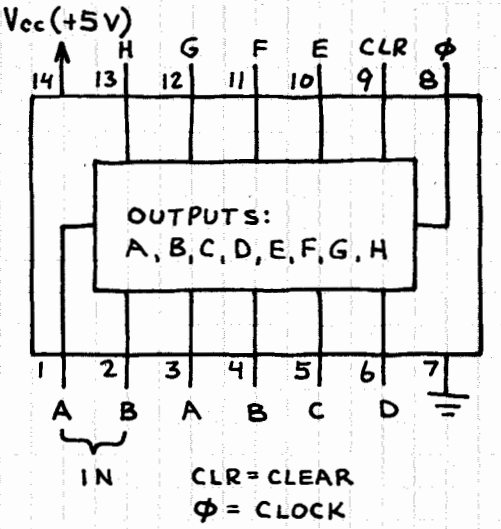


PSEUDO-RANDOM VOLTAGE GENERATOR



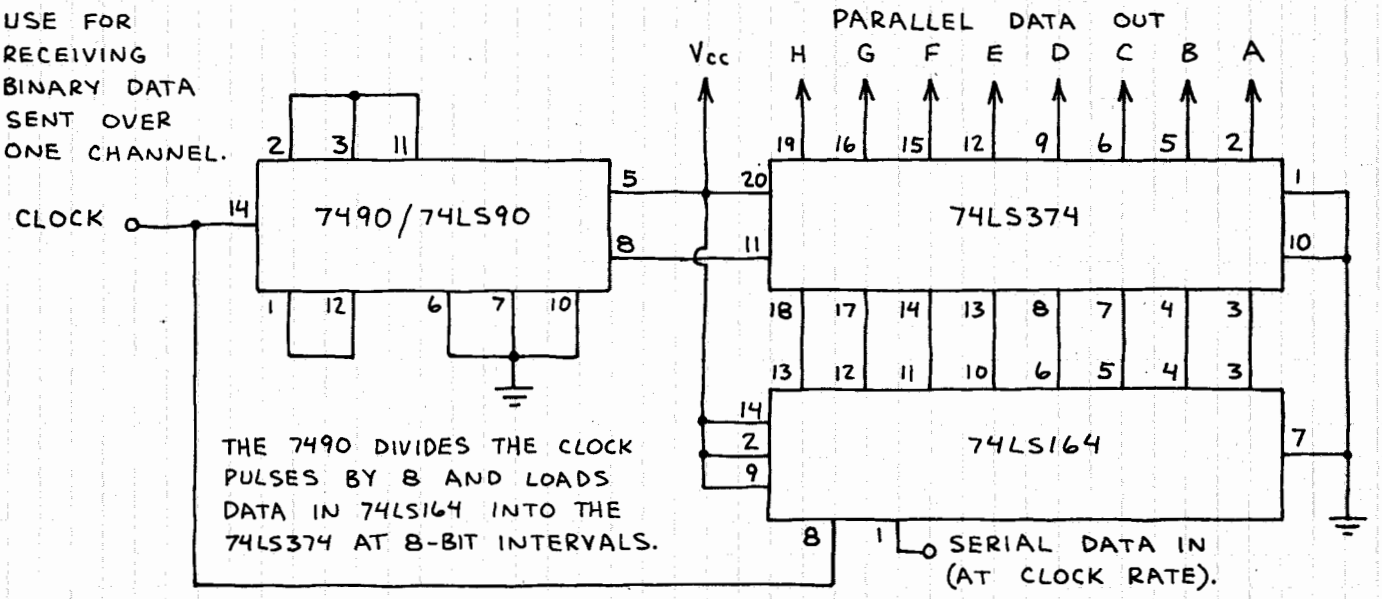
8-BIT SHIFT REGISTER
74LS164

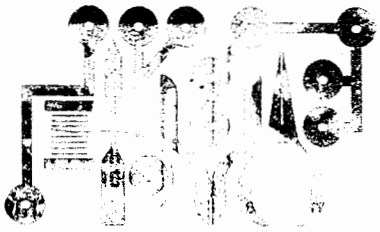
DATA AT ONE OF THE TWO SERIAL INPUTS IS ADVANCED ONE BIT FOR EACH CLOCK PULSE. DATA CAN BE EXTRACTED FROM THE 8 PARALLEL OUTPUTS OR IN SERIAL FORM AT ANY SINGLE OUTPUT. ENTER DATA AT EITHER INPUT. THE UNUSED INPUT MUST BE HELD HIGH OR CLOCKING WILL BE INHIBITED. MAKING PIN 9 LOW CLEARS THE REGISTER TO LLLL.



8-BIT SERIAL-TO-PARALLEL DATA CONVERTER

USE FOR RECEIVING BINARY DATA SENT OVER ONE CHANNEL.





Experimenter's Corner

By Forrest M. Mims

More On Shift Registers

IN LAST month's column, we introduced the shift register as one of the most versatile of all digital-logic circuits. Shift-register operation and several shift-register applications were then covered. This month, we'll complete our discussion of shift registers by examining some specific chips and experimenting with some application circuits we think you'll enjoy.

Integrated Shift Registers. As we mentioned last month, you can custom-design a shift register by using any number of cascaded flip-flops. Fortunately, this procedure is rarely necessary. Semiconductor companies have designed and manufactured in integrated form most of the popular shift-register configurations.

Many different TTL and MOS shift registers are available, often for very reasonable prices. Even if you don't have immediate plans for incorporating one or more shift registers in a project of your own design, you'll be far better prepared to tackle a future project if you're aware of some of the chips that are now available.

TABLE I—TTL SHIFT REGISTERS

Type	Function	Bits	Freq. (MHz)	Shift Right	Shift Left	Load	Hold
7491	SISO	8	10	Yes	No	No	No
7494	SISO	4	10	Yes	No	Yes	No
7495	PIPO	4	36	Yes	No	Yes	No
7496	PIPO	5	10	Yes	No	Yes	No
74L99	PIPO	4	3	Yes	No	Yes	No
74164	SIPO	8	25	Yes	No	No	No
74165	PISO	8	25	Yes	No	Yes	Yes
74166	PISO	8	20	Yes	No	Yes	Yes
74178	PIPO	4	25	Yes	No	Yes	Yes
74179	PIPO	4	25	Yes	No	Yes	Yes
74194	B/PIPO	4	25	Yes	Yes	Yes	Yes
74195	PIPO	4	30	Yes	No	Yes	No
74198	B/PIPO	8	25	Yes	Yes	Yes	Yes
74199	PIPO	8	25	Yes	No	Yes	Yes
74LS295	PIPO	4	25	Yes	No	Yes	No
74LS299	B/PIPO	8	35	Yes	Yes	Yes	Yes
74LS323	B/PIPO	8	35	Yes	Yes	Yes	Yes
74LS395	PIPO	4	25	Yes	No	Yes	No

TABLE II—CMOS SHIFT REGISTERS

Type	Function	Bits	Freq. (MHz)	Shift Right	Shift Left	Load	Hold
4006	SISO	18	10	Yes	No	No	No
4014	PISO	8	5	Yes	No	Yes	No
4015	SIPO	8	9	Yes	No	No	No
4021	PISO	8	5	Yes	No	Yes	No
4031	SISO	64	8	Yes	No	No	No
4034	B/PIPO	8	10	Yes	Yes	Yes	Yes
4035	PIPO	4	5	Yes	No	Yes	No
4	SIPO	8	5	Yes	No	No	Yes
40100	B/SISO	32	3	Yes	Yes	No	Yes
40104	B/PIPO	4	9	Yes	Yes	Yes	No
40194	B/PIPO	4	9	Yes	Yes	Yes	Yes

Tables I and II list most of the TTL and CMOS shift registers available to experimenters. These tables, which were compiled from Motorola, National, RCA and Texas Instruments data books, list only some of the more important shift register parameters. Also, some specifications (such as maximum clock frequency) can vary somewhat within the same chip type if different manufacturers are involved. Therefore, you should check the specifications provided by the manufacturer of the chip you are thinking of using in a project for more specific information.

Here is a brief explanation of the table headings:

Function. The function of each shift register is identified by a four- or five-letter code. S means serial, P parallel, I in, O out, and B bidirectional. Therefore, a shift register listed as B/PIPO can provide bidirectional, parallel-in/parallel-out operation. Incidentally, many of the shift registers listed in the tables will provide more than one operating function. Most PIPO registers, for instance, also provide SISO operation.

Bits. This parameter specifies the number of register elements in the listed device.

Frequency. The maximum shift (clock) frequency is given. Frequency specifications for CMOS shift registers vary with both the V_{DD} supply voltage and the manufacturer. Consult the manufacturer's literature for the specific operating frequencies for a given chip.

Modes. Four operating modes are given. Those whose functions are obvious are *shift left* and *shift right*. *Load* is normally found on parallel-input shift registers. Depending upon the status of the load input, upon receipt of the next clock pulse, the register will either ignore or accept the data present at its input(s). *Hold* means that the clock input can be inhibited or disabled so that the shift register will store its contents like a memory register.

Several other operating modes not listed in the tables may also be available. *Preset* or *clear* enables all the register elements to be cleared to logic 0. *Recirculate* causes the data at the output of a shift register to be cycled back to the input.

Exotic Shift Registers. Several kinds of elaborate integrated shift registers are available for such serial-memory applications as refreshing cathode-ray tube traces and storing characters to be printed by high-speed printers. A typical example is Synertek's SY1404A 1024-bit MOS dynamic shift register. This family also includes a dual 512-bit register (SY1403A) and quad 256-bit register (SY1402A). The maximum clock rate for these chips is a relatively slow 2.5 MHz, but by means of a multiplexing technique, data can be accepted at a 5-MHz rate. Even more capability is provided by 2048-bit shift registers such as the SY2401 and SY2827.

Bubble memories and charge-coupled devices (CCDs) are among the most exotic shift registers available. Bubble-memory capacity can exceed a *million bits per chip*, making this exotic register a strong contender in the search for a solid-state replacement for the disk memory.

Application Circuits. Now let's try experimenting with some readily available shift registers to see how they work and what they can do. The circuits that follow use both CMOS and TTL shift registers. Once you see how easy it is to use

these various chips, you'll want to consider experimenting with other shift registers as well.

eudorandom Sequencer. Figure 1 is the schematic of a pseudorandom generator whose operation is patterned after the S2688/MM5837 shift-register noise generator that was described in the March 1980 installment of this column. The circuit consists of two 8-bit 4021 CMOS shift registers (*IC2* and *IC4*) cascaded to form a single 16-bit register.

The 4021 is a PISO register with the bonus feature that the outputs of the 6th, 7th and 8th stages are available. This means it can be used as a 6-, 7- or 8-bit shift register and makes possible the pseudorandom sequencer circuit. Such a circuit requires that the outputs of two stages in a shift register be coupled back to the input via an exclusive-OR gate.

By connecting the exclusive-OR gate's inputs to the final two outputs of the shift register, the sequencer will generate a pseudorandom sequence that recycles after 255 clock pulses. The bit pattern within a single 255-bit cycle is essentially random, but the periodic recycling of the pattern compromises its randomness over the long term.

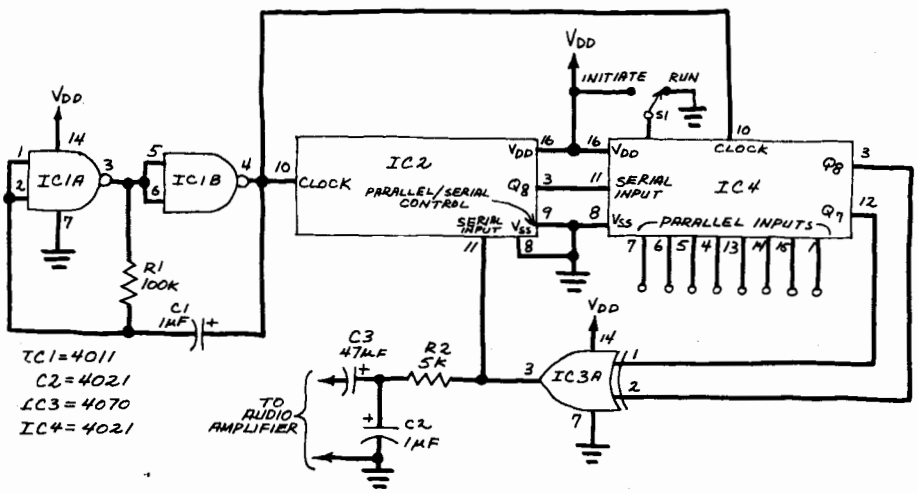


Fig. 1. Schematic diagram of a pseudo-random sequencer made from CMOS shift registers.

Two of the gates in *IC1* form a simple clock for this circuit. This is a CMOS chip, so be sure to connect the unused inputs (pins 8, 9, 12 and 13) to V_{DD} or V_{SS} to keep the chip from drawing excess current and overheating. You should also connect the unused inputs of *IC3* (pins 5, 6, 8, 9, 12 and 13) to either V_{DD} or V_{SS}.

This circuit has several useful applications. Connect its output to an audio amplifier through the low-pass filter composed of *R2* and *C2* to convert the pseudorandom bit pattern into audible noise. For best results, speed up the clock by reducing the value of *C1* to 0.01 μF or so. At very fast clock speeds, the output will sound like a pure tone. At slower frequencies, the repetitious pattern present in the output signal can be heard.

An interesting LED flasher can be made by connecting the output of the circuit to the cathode of a LED whose anode is connected to V_{DD} through a 1000-ohm series resistor. Slow the clock rate to a few hertz by increasing the value of *C1* to 10 μF or so. The LED will then flash on and off in an irregular, seemingly random pattern. This LED-flasher application illustrates how the circuit can be used to strobe (actuate) another circuit at a pseudorandom rate.

As you alter the clock rate or change the connections to the shift registers, the pseudorandom sequence generator might shut down. Make sure the clock is working by monitoring its output with an audio amplifier and loudspeaker (a tone should be heard) or with an oscilloscope (a square wave should be displayed). If the clock is running, try resetting h shift register by switching its pin 9 (the PARALLEL/SE-..AL CONTROL input) from V_{SS} to V_{DD} and then back to V_{SS}. The circuit should then resume normal operation.

You can control the pattern of bits moving through the shift registers. Connect the PARALLEL DATA inputs of *IC4* to V_{DD} or V_{SS} to set up any desired sequence of logic 0's and 1's. Then toggle *S1* from RUN to INITIATE and then back to RUN to load the PARALLEL DATA inputs. Repeat this procedure as desired to create many different bit sequences.

Incidentally, if you intend to use this circuit for commercial purposes, you should first write the U.S. Patent and Trademark Office (2021 Jefferson Davis Highway, Arlington, VA 22202) and request a copy of U.S. Patent 4,191.175. The fee for a single copy of a patent is 50¢. I've not yet seen this patent myself. However, after the March 1980 "Experimenter's Corner" appeared, William L. Nagle, president of Paratronic Systems, Inc. (Honeybrook, PA 91344) wrote to this magazine that, ". . . your readers should be cautioned that use of this for any other than private purposes would be an infringement of the patent our company holds for such devices and their applications." The complete letter was reproduced in the *Letters* column of the July 1980 issue.

I am interested in seeing this patent because its number indicates an issuance date late in 1979. Publications describing pseudorandom sequence generators date back to at least 1973 when Fairchild Semiconductor's *The TTL Applications Handbook* (a truly outstanding book) described two such sequencers on pages 8 through 21.

Incidentally, one of the Fairchild circuits employed one 4-bit and seven 8-bit shift registers to generate a truly *long*, nonrepetitive output bit sequence. According to the descriptive text, ". . . even at a frequency of 20-million states per second the counter would not repeat until more than 18 centuries had elapsed." (!) For more information about pseudorandom sequencers, see Don Lancaster's indispensable *CMOS Cookbook* (Howard Sams & Co., 1977). Don discusses the topic and presents two circuits on pages 318 through 323.

Pseudorandom Voltage Generator. Connect a suitable resistive ladder network to the outputs of a SIPO or PIPO shift register set up as a pseudorandom sequencer and you get a pseudorandom voltage generator. The resistor network serves as a digital-to-analog converter.

Figure 2 shows such a circuit designed around a 74164 or 74LS164 SIPO 8-bit shift register. The four NAND gates (*IC2A* through *IC2D*) form an exclusive-OR gate. You can substitute one-fourth of a 7486 quad exclusive-OR gate in the circuit if you prefer.

Use a 555 timer IC connected as an astable oscillator like the one shown in Fig. 3 to provide clock pulses for the circuit. The amplitude of the pseudorandom, stepped output voltage can be changed by connecting pin 2 of *IC2A* to any of the six other output pins of *IC1*.

An interesting application for this circuit is a pseudorandom tone sequencer that can be made by connecting its output to the control input of a voltage-controlled oscillator or voltage-to-frequency converter. One suitable vco is the 4046, a chip that was highlighted in "Experimenter's Corner" for July and August 1980. Suitable V/F converters include the

9400 and LM331—see "Experimenter's Corner," for October 1979 for details.

-Bit Serial-to-Parallel Data Converter. Here's a cir-uit that you can use to load serial data onto an 8-bit data bus. Referring to Fig. 4, clock pulses are applied simultaneously to

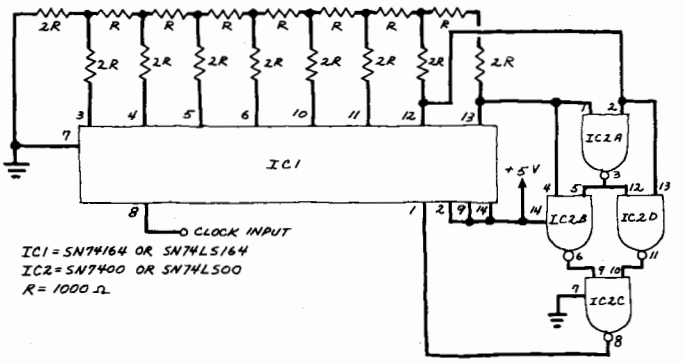


Fig. 2. Schematic of a pseudo-random voltage generator.

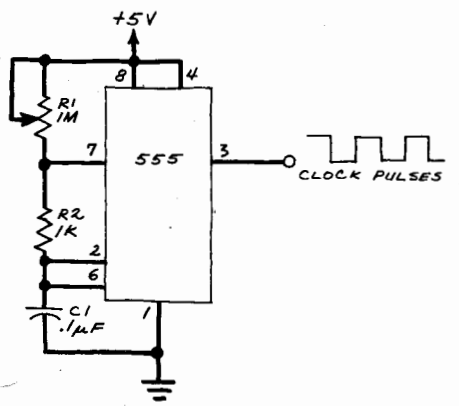


Fig. 3. A 555 clock pulse generator.

experiment with the circuit as I did when working with the prototype, connect the cathode of a LED to each output pin of *IC2*. Connect the anodes of the LEDs to 1000-ohm resistors which are in turn connected to +5 volts. You can use smaller resistance values (as small as 270 ohms) for the current-limiting resistors if you want the LEDs to glow brighter. I prefer 1000 ohms to keep current consumption down to 2 or 3 milliamperes per LED.

An additional LED and series resistor connected between pin 8 of *IC1* and ground will allow you to monitor the status of the shift register. You can then manually test the circuit by slowing down the clock to about one pulse per second and applying input data bits to pin 1 of *IC3*.

This circuit can be used as the receiving portion of a digital data-transmission system. A good design exercise would be to devise a suitable transmitter for converting 8-bit bytes into a serial bit stream. Hint—use a PISO shift register and a divide-by-8 counter.

Programmable Sequence Generator. Figure 5 schematically shows an 8-bit programmable sequence generator made from two series-connected 74194 shift registers (*IC1* and *IC2*). In operation, any desired bit pattern is first selected by means of switches *S1* through *S8*. A normally closed push-button switch, *S9*, is momentarily opened to load the selected bit pattern into the two shift registers. When clock signals are applied simultaneously to pin 11 of both shift registers, the bit pattern will be shifted one position for each clock pulse. Because the serial output of the second shift register (pin 12 of *IC2*) is connected to the serial input of the first shift register (pin 2 of *IC1*), so long as the clock pulses are received the bit pattern will recirculate through the registers. It will remain unchanged until it is modified by means of the data-select and load switches.

Use the oscillator shown in Fig. 3 to generate clock pulses for this circuit. The output LEDs shown in Fig. 5 are optional, but including them allows the circuit to double as an attention-getting programmable light flasher. The LEDs can be arranged in various patterns to enhance the effect.

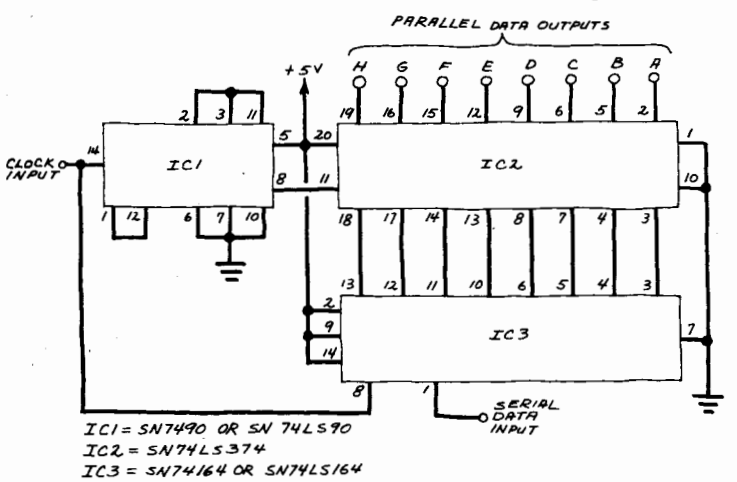


Fig. 4. Schematic of an 8-bit serial-to-parallel data converter.

counter *IC1* and shift register *IC3*. The data to be loaded on the bus should be applied at the clock rate to the serial input of the shift register.

The 7490 (*IC1*) is configured as a divide-by-eight counter. It generates a load pulse that causes *IC2*, a 74LS374 octal D flip-flop, to accept the data appearing at the parallel outputs of shift register *IC3*.

Summarizing, after 8 bits have been loaded into the shift ister, a pulse from the counter causes the data to be trans-red to the octal flip-flop. The flip-flop is updated with new data from the register after eight additional clock pulses.

You can use the 555 oscillator shown schematically in Fig. 3 as a clock circuit for the data converter. If you want to

A more practical application can be accomplished by connecting a resistor ladder network (see Fig. 2) to the outputs of the shift registers. The circuit then will function as a programmable waveform generator. Without the ladder network, the circuit can be used to strobe various circuits in any programmed sequence. More shift registers can be added for even longer sequences.

If you build this circuit, be sure to include the 0.1-μF power-supply decoupling capacitors. Without these capacitors, the shift registers will be affected by power-supply transients that can arise during the switching sequence. A typical effect of such a transient is an unwanted change in the sequence.

stage goes low, the bit resulting from the gating of the first and last bits is a logic 0.

Because this bit is fed back to the input of the first register (pin 2 of *IC1*), it might at first glance appear that all of the register outputs would remain at logic 0 after the first cycle of clock pulses. Note, however, that the output of gate *IC3C* is connected to pin 10 of both registers. When the output of this gate is logic 0, the shift registers ignore the data presented to their parallel inputs. When its output switches to logic 1, the shift registers load the data present at their inputs. This, of course, is what occurs when the ENABLE INPUT is brought to logic 0 to start the circuit.

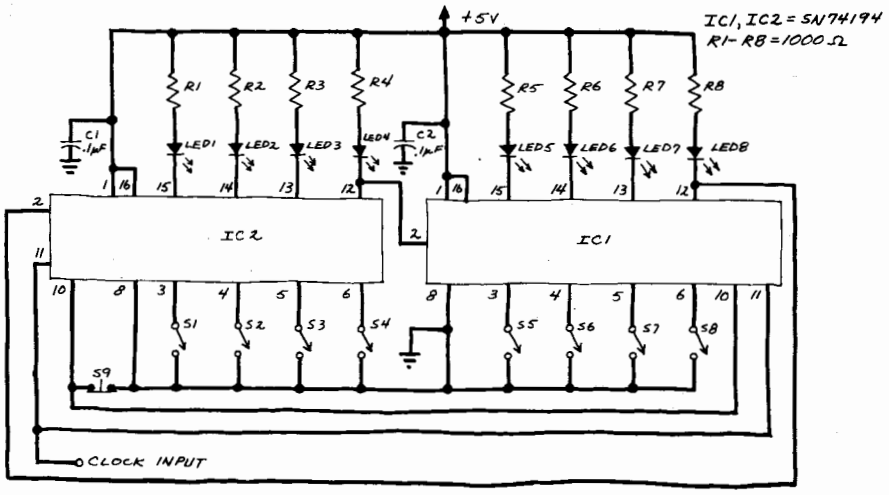


Fig. 5. A shift-register 8-bit programmable sequence generator.

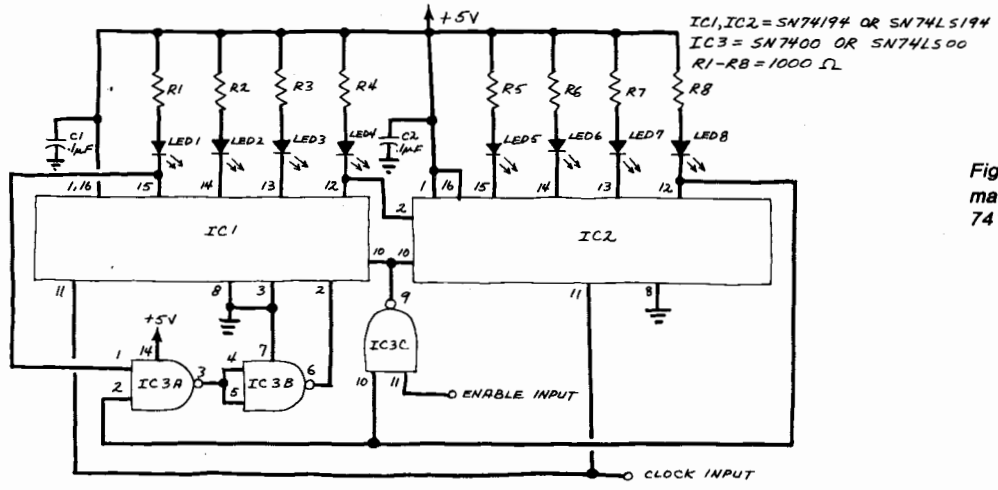


Fig. 6. A bargraph generator made from a pair of 74194 shift registers

Bargraph Generator. Figure 6 is the schematic diagram of an unusual bargraph generator made from a pair of 74194 shift registers. In operation, the circuit's ENABLE INPUT is brought momentarily to logic 0 to start the circuit. This causes seven of the eight bit positions to be loaded with logic 1's, because their inputs are left unconnected and therefore assume a high state. The output of the first stage of *IC1* goes to logic 0 because its input (pin 3) is grounded.

The bits in the first and last stages, logic 0 and logic 1, respectively, are combined in AND fashion by *IC3A* and *IC3B*. The result is presented to the serial input of the first shift register. On the first clock pulse, therefore, the outputs of the first two positions go to logic 0 while all of the other outputs stay at logic 1. This pattern continues as subsequent clock pulses are received until each of the shift-register outputs switches in turn from logic 1 to logic 0. When the final

This circuit has such practical applications as strobing various external circuits in a sequential fashion. It also makes a very interesting visual display.

Going Further. Shift registers are ideally suited for experimentation. The bargraph generator shown in Fig. 6 is a good example of this. I began with the remnants of the circuit in Fig. 5 (the input switches were removed) and tinkered with the basic circuit by adding a single 7400 quad NAND gate IC. Within a few minutes, the bargraph effect was achieved. You can do the same kind of experimentation on your own by selecting different register outputs to be connected to NAND gate *IC3A* in Fig. 6. You can also try adding additional stages for more complex effects. The cost of integrated shift registers is very reasonable, and you'll learn a good deal about these very versatile logic circuits by experimenting with them. ◇

Do you know what a MOS shift register is? Do you know how it works? Here are the answers plus how to interface them with other logic families and different applications

by DON LANCASTER

A SHIFT REGISTER IS A DIGITAL DATA STORage device. The data can be the letters to be displayed on a TV screen, numbers in a computer or calculator, intermediate values in a digital filter, or part of an elaborate code or sequence. Shift registers are made up of individual *stages*. Each stage can store one *bit* of information, called a binary **1** or a **0**, and usually corresponding to a "yes" or "no" or else perhaps a "present" or "absent" command. Four bits together can represent a decimal number, while six bits together can handle one ASCII character, and so on. In a shift register, the contents can be moved or *shifted* so that the contained information is marched one and only one stage at a time through the device. The shifting process is called *clocking* and one or more clocks are involved in completing the shifting operation.

Figure 1 shows how we might make a shift register out of either a JK or type-D flip-flop. While TTL (Transistor-Transistor logic) devices are shown, we could use any logic family we like. Input data corresponding to a "1" or "0" is presented to the first stage. When the system is clocked, the first bit of data is *entered* and then stored in the *first* stage. On the second clocking, the contents of the first stage get passed on to the second, and the first stage then accepts a new bit of information from the input. The next clocking passes the output of stage 2 on to stage 3, and the output of stage 1 on to stage 2. Finally, stage 1 accepts a new bit of input information.

One more clocking *fills* the register in Fig. 1 as it is only four bits long, and all four stages now have information in them. If we do no more clocking, the register will *keep* the information we sent it. Four more clocking pulses and we can march the data out and use it somewhere else.

So what good is a shift register? We can use it to *store* information. It is a digital *memory*. We can use it to *delay* information. We can use it to *format* information, either in a *buffer* mode where the enter and readout clock rates may be different, or in a *variable-access* mode where we can enter and leave individual stages with data. With certain types of shift registers, we can convert *serial* data to *parallel* form or parallel data (all at once) to serial (one at a time in sequence) form. We can also build counters and sequencers with shift registers. Two popular types are called the *walking ring* computer and the *pseudo random sequence* generator.

Organization

The *organization* of a shift register is decided by how many stages it has and how you can get at the individual stages.

A serial-in-serial-out register gives you the input only to the first stage and the final output of the last stage. it is sometimes called a *serial* register or a SISO (Serial-in-Serial-Out) register. There is no intermediate access.

A SIPO register gives you the outputs of all stages including the last one. The eight-bit 74164 is a typical TTL example. A parallel-in-serial-out or PISO register lets you simultaneously load all the stages but then marches the contents out as a serial-bit string. The TTL 74165 is an eight-bit example of this type.

The most versatile type of shift register would be a PIPO (Parallel-In-Parallel-out) version. Here, you could load data either serially one bit at a time or "broadside" parallel. You could also get all the data out either in broadside parallel all-at-once form, or one bit at a time in serial form. The 74195 is a four-bit TTL package that does this.

You might think that since you could use the PIPO register for everything else anyway that it would be the only way to go. The problem is that you can easily put 2048 shift register stages on a single small chip of silicon. For a 2048-bit PIPO register, you'd need a minimum of 4099 leads for inputs, outputs, clocks, and power supplies. This is a most unwieldy package to say the least, even if we don't worry about the extra circuitry needed for each parallel input. Now the same register can be done SISO in as little as 5 leads.

So, for *short* shift register applications, we have a choice of the four formats. For *long* shift register uses, the only economical way to go is the SISO route. We'll consider everything longer than 24 bits a *long* shift register here. This is often a changeover point. 24 bits or less and you usually use the more flexible and faster TTL registers, often at four or eight stages per package. Above 25 bits, you go to the long serial MOS registers and pick up as many as 2048 bits of storage in a single package.

The majority of registers shift only towards the output and are called *shift right* registers. A very few can also shift back towards the input and are called *bidirectional* or *shift-right-shift-left* devices. These are expensive and not normally available in long lengths. One trick you can do with a *recirculating* register (more on this in a bit) is clock it rapidly ahead one stage less than its length, making it *appear* to back up one, rather than go forward *all but one* of its stages.

Two more things may enter into our register organization. We may have more than one shift register in a single package. One, two, and six registers per package are common. Usually, they have common clocking, but not always. For instance, the Signetics 2518 is a hex 32-bit shift register; the 2519 is a hex 40-bit version. Both have common clocking and a common enter/recirculate control.

You often use several shift registers in parallel. For instance, you might use four shift registers to individually handle each bit of a four-bit BCD or binary-coded-decimal digit. Thus each clocking of the register array gets you a whole new decimal number, rather than only ¼ of it The four bits is sometimes called a *word* and sometimes a *byte*. Likewise, an alphanumeric character can be represented by a six bit ASCII character code. Here, we use six registers at once to give us one whole new character on each clocking. Of course, we have to make sure all the registers get clocked exactly alike, for if they didn't, all the data bits would be hopelessly scrambled. This is usually very easy to prevent.

A final feature of a shift register's organization is its *recirculatibility*. Sometimes we might like to look at the contents of a shift register a bit at a time, and then *return* the information back into the same relative slots in the shift register for later use. This is called *recirculation*. Some sort of switching or selection must be provided if you are sometimes going to *enter* new data as opposed to recirculating old data. Some of the long MOS shift registers have an *internal* recirculate logic and are normally used if you need recirculation. We'll see in a minute that recirculation is essential for the *dynamic* registers if you are going to keep the data more than a fraction of a second. Figure 2 shows the logic needed to add an external recirculate to a shift register.

Long MOS shift registers

There's an incredible variety of long shift registers available using several different MOS (Metal-Oxide-Semiconductor)

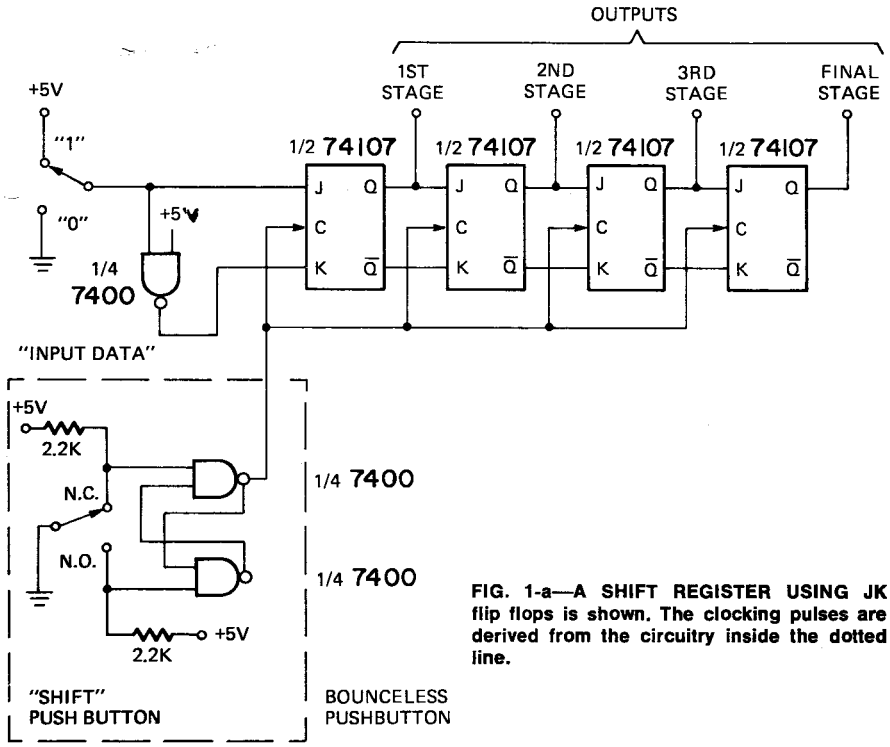


FIG. 1-a—A SHIFT REGISTER USING JK flip flops is shown. The clocking pulses are derived from the circuitry inside the dotted line.

(*a*) USING "JK" FLIP FLOPS

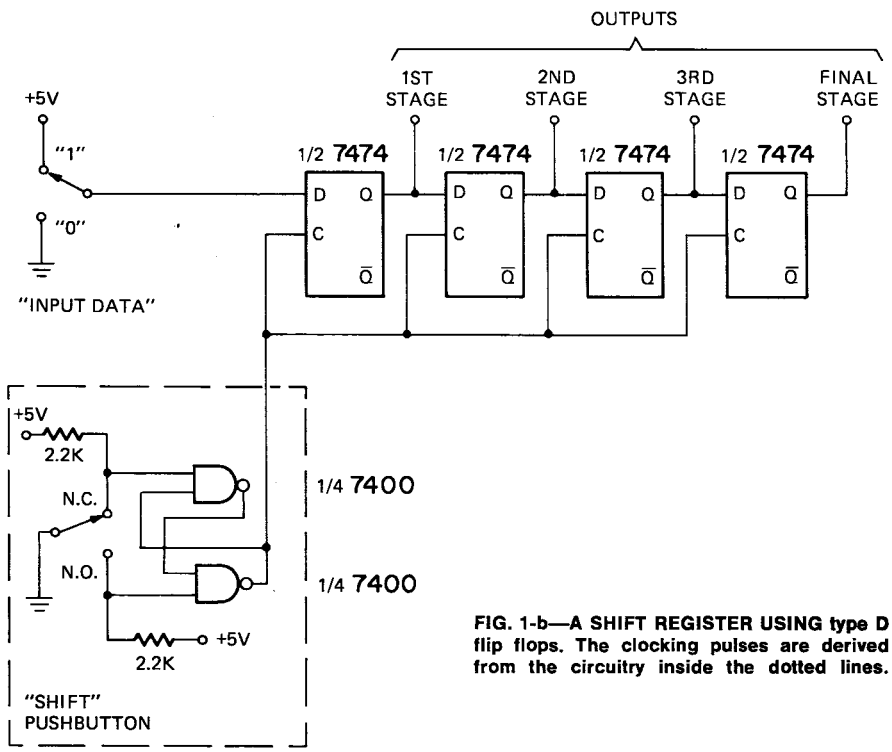


FIG. 1-b—A SHIFT REGISTER USING type D flip flops. The clocking pulses are derived from the circuitry inside the dotted lines.

(*b*) USING "D" FLIP FLOPS

technologies. These range from small 16- and 21-bit versions up to 2048-bit ones in a single package. A brief and more or less random listing is given in Table I, while some of the more prominent manufacturers are listed in Table II. The typical single-unit price varies from around $3 to around $15 per unit and typically runs well under a penny per bit for the longer versions. Some of these have shown up surplus (see back ads of **Radio-Electronics**) for as little as a quarter each for manufacturers seconds. Some of the seconds we tested from the back ads run around a 45% "completely useful" yield. All of these devices are serial-in-serial-out. Typical maximum frequency of operation is 2 or 3 megahertz, although you get much better behavior at a 500 kHz or so rate.

Before you can use any long MOS shift register, you have to ask the following questions:

1. Is the register *static* or *dynamic?*
2. How do you *interface* it with TTL or other logic?
3. What kind of *clock* signals are needed and how many of them?
4. Can it recirculate by itself?
5. Does it have write or read *enables* that lets you combine it with more registers?

Let's take a look at these important concepts in a bit more detail.

Static versus dynamic

Figure 3 shows three different types of shift registers. Our registers of Figs. 1 and 3-a used two flip-flops for storage. They will keep data so long as we apply supply power and are called *static* registers, or sometimes *fully static* registers.

Transformation of information in *any* shift register *has* to be a *two-stage* process or a *two-phase* process. On the beginning of a shift, information is transferred into some form of temporary storage. At the completion of a shift, the information is then sent to a *final* storage. In the case of Fig. 1-a, we have a master (temporary) and a slave (final) storage *within* each JK flip-flop's logic block. The reason for the *necessity* of *two* storage phases per shift is simple—try it with only one, and you get a wild, unchecked race through several stages instead of an orderly progression of one and only one complete stage per clocking.

We don't need a full flip-flop for some applications. Instead, we can use the temporary storage of a capacitor. So, Fig. 3-b shows us a *dynamic* shift register. The capacitor will hold information for us for a reasonably short time, but eventually the leakage will get to us and destroy the information in the cell. Capacitor storage is much simpler and more economical than flip-flops as it usually uses the "free" capacitance found in normay strays. Most dynamic MOS shift registers will hold their information for UP TO one-tenth of a second. Should you fail to clock them in that time, the information is lost.

So, if you are only going to keep your information in your shift register for under a fraction of a second before finally using it, it doesn't matter whether you use a static or a dynamic register. The trouble is that most applications call for data to be reused or held longer than a fraction of a second. So, if you are to use the cheaper, denser dynamic shift registers, you have to move or *refresh* the data a minimum of several dozen times a second. One way to handle the moving of data is to march the information completely once around at least several dozen times per second. In a computer terminal or TV Typewriter, recirculation at the 60 hertz vertical rate is one good approach.

Figure 3-c shows an interesting compromise between static and dynamic registers. Here, we use a capacitor for the temporary storage and a flip-flop for the final storage. This is a compromise that gives us static performance at slightly over half the normal cost. Strictly speaking, this is called a *quasi-static* operation, but practically all the "static" MOS reg-

isters use this technique. There is only one restriction, the clock line must remain in a specified level during the static part of the operation, and there is a *maximum* allowable clock pulse width during the dynamic transfer process.

Interface

Most of the long MOS registers will interface with TTL, DTL, and RTL, but most often a few resistors are needed. You have to read the data sheets very carefully. Unless the data sheet specifically states otherwise, the clock lines are NOT compatible with TTL and take special drive circuitry. More on this in just a bit. Remember that the inputs, enables, recirculates, and output pins can be made TTL compatible, but the clock almost always takes special circuitry.

There are lots of different MOS technologies, and each takes one of the interface circuits shown in Fig. 4. You can usually tell the technology by the supply voltage used or recommended.

If the supplies are ±15 volts, chances are it is a *metal gate* or *high threshold P channel device.* These are the oldest MOS integrated circuits and the hardest to interface. To drive them, you need an *open circuit* TTL logic block that can withstand 15 volts. Suitable devices are the 7406 and 7416. A pull-up resistor is provided to produce the ground and ±15-volt logic inputs. Two resistors are normally used in going from the MOS to TTL, one down to −15 to provide the −1.6 mA needed for a TTL "0", and one series resistor to limit the positive swing to 5 volts or less.

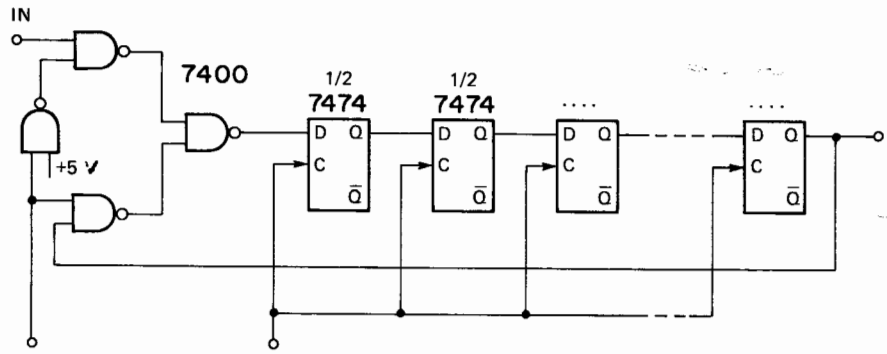
Silicon gate circuits are presently the most common. They have a +5 and −12-volt supply. Usually a 2.2K pull-up resistor is recommended when they are driven by TTL, and their output drive capability depends on the particular output structure used. Often a single 6.8K resistor to −12 volts does the trick.

N-channel circuits often work with a single +5-volt supply and are directly TTL compatible without resistors on output and input. *CMOS* integrated circuits also work off a single +5- to +15-volt supply. At +5 volts, they are directly TTL compatible on an input, but may not have enough output drive current for regular TTL, so low-power TTL is often used as an output sense amplifier.

Its usually tricky to simultaneously drive another MOS stage along with TTL as the voltage and current swings don't usually work out too well. To get around this, you usually run through a single TTL inverter and use its output to drive the MOS following.

Clocks

More problems happen with long shift registers over clocks and clocking than over any other single difficulty. First and foremost, consult the individual data sheets for the device you are going to use. Unless it specifically says so otherwise (boldly and in large print!), the clock lines are not compatible with TTL. Usually the clock lines need almost the entire supply swing, such as a 16- or 17-volt swing for a silicon gate circuit on +5-, −12-volt power supplies. Further, what-



⏚ = ENTER
+5V = RECIRCULATE

FIG. 2—RECIRCULATING SHIFT REGISTER. Data can be fed from the output to the input.

TABLE I

A FEW OF THE MORE POPULAR LONG MOS SHIFT REGISTERS

ELECTRONIC ARRAYS:

EA1003 Dual 32, static, rec.
EA1004 Dual 100, static
EA1007 Dual 32, static
EA1200 Quad 32, dynamic
EA1203 Variable 1-64 dynamic
EA1210 Dual 526 dynamic
EA1212 Single 512 Dynamic

FAIRCHILD:

3325 Quad 64, Dynamic
3330 480 Bit, Dynamic
3342 Quad 64, Static
3343 Dual 128, Static
3346 Dual 144, Static
3383 Single 256, Dynamic

INTEL:

1402 Quad 256, Dyn, Mpx.
1403 Dual 512, Dyn, Mpx.
1404 Single 1024, Dyn, Mpx.
1405 Single 512, Dyn, Recirc.
1506 Dual 100 dynamic
2401 2048 dynamic, recirc.
2405 1024 dynamic, recirc.

MOSTEK:

MK1002 Dual 128, Static
MK1007 4 x 80, dynamic

MOTOROLA

MC1141G Triple 66 dynamic
MC1142G Single 200 dynamic
MC1160G dual 100 dynamic
MC1161G Dual 50 bit static
MC2360G Dual 100 Static
MC2361G Dual 128 Static
MC2362G Dual 250 Static
MC2363G Dual 256 Static
MC2380G Dual 100 dynamic

NATIONAL:

MM400 Dual 25 Dynamic
MM402 Dual 50 Dynamic
MM406 Dual 100 Dynamic
MM4001 Dual 64 Dynamic
MM4006 Dual 100 Dynamic
MM4012 Dual 256 Dynamic
MM4013 Single 512, dyn, rec.
MM4105 Quad 64, static
MM5054 Dual 64/72/80 static

SIGNETICS:

2505 Single 512 dyn, rec.
2506 Dual 100, dynamic
2509 Dual 50 Static
2510 Dual 100 Static
2511 Dual 200 Static
2512 Single 1024, dyn, rec.
2518 Hex 32, static, rec.
2519 Hex 40, static, rec.
2521 Dual 128, static
2522 Dual 128, static
2524 Single 512, dyn, rec.
2525 Single 1024, dyn, rec.
2527 Dual 256 static
2528 Dual 250 Static
2529 Dual 240 Static
2532 Quad 80 static
2533 1024 static, rec.

TEXAS INSTRUMENTS:

TMS3000 Dual 25 static
TMS3001 Dual 32 static
TMS3002 Dual 50 static
TMS3012 Dual 128, stat, rec.
TMS3102 Dual 80, static
TMS3112 Hex 32, static, rec.
TMS3113 Dual 133 static, rec.
TMS3304 Triple 66, dynamic
TMS3309 Dual 512, dynamic
TMS3314 Triple 60+4 dynamic
TMS3412 Single 1024 Dynamic

TABLE II

SOME LONG MOS SHIFT REGISTER SOURCES

ELECTRONIC ARRAYS INC.
501 Ellis Street
Mountain View, California 94040

FAIRCHILD SEMICONDUCTOR
464 Ellis Street
Mountain View, California 94040

INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051

MOSTEK
1215 West Crosby Road
Carrolton, Texas 75006

MOTOROLA SEMICONDUCTOR
Box 20912
Phoenix, Arizona 85036

NATIONAL SEMICONDUCTOR
2900 Semiconductor Drive
Santa Clara, California 95051

SIGNETICS
811 East Arques Avenue
Sunnyvale, California 94086

TEXAS INSTRUMENTS
Box 5012
Dallas, Texas 75222

ever is driving the clock has to drive a bunch of internal switches in a long register, so the clock-line capacitance may be several hundred picofarads. Since you need sharp rise and fall times on the clock, it usually takes a special circuit called a *clock driver* to get the job done,

the peak currents involved in charging and discharging the clock line capacitances may be several hundred milliamperes or more. Except for the simplest circuits, a push-pull "totem pole" drive circuit is needed, and a small current limiting resistor (usually 10 ohms) must be provided between the registers and clock lines to prevent short circuit damages and risetimes that raise havoc with the supply lines and decoupling. The clocks must NEVER be allowed to "overshoot" and exceed the positive supply voltage, even briefly for this will destroy or selectively change the information in the register. Clocks must be the proper widths and must not overlap. Where two clocks are used, the "daylight" or space between them is just as important as their widths.

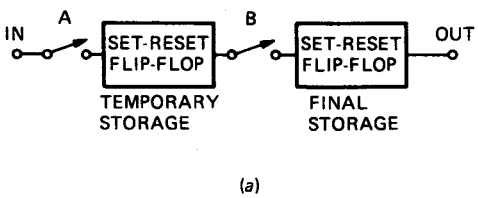
As a general rule, always use clock widths near the *minimum* called for on the data sheets. With most registers, the wider the clock pulses, the more the supply current, and the hotter the IC runs, leading to potential temperature and bit pattern sensitivity problems. Clock widths should be precisely derived from system timing instead of randomly adjusted through monostables or half-monostable pulse shapers, since the position and widths can be quite critical.

On your first design with a new long MOS register, you also have to watch for the number of clocks needed per cycle. Generally static registers need a single clock and each clock pulse advances the information one stage. Static registers are also usually much easier to drive on their clock lines.

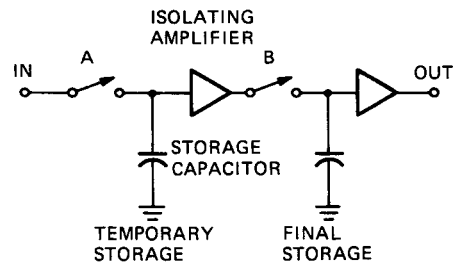
Most dynamic registers have two clock lines and need two clock drivers. One clock is the *input* clock; one is the *output* clock. A *pair* of clock pulses is needed to advance the information one stage.

Finally, there are a few dynamic *multiplexed* registers such as the *Intel* 1402, 1403, and 1404. These are tricky and hard to use. They contain *two* internal shift registers with a *common* input and output. What is an input clock for one side is the output clock for the other half and vice versa. The data *externally* appears to travel one stage *per* clock pulse, although a *pair* of clock pulses is needed to *complete* each transfer operation. If you are not very careful, you can end up one clock pulse short or long of what you really need, and change the effective register length.

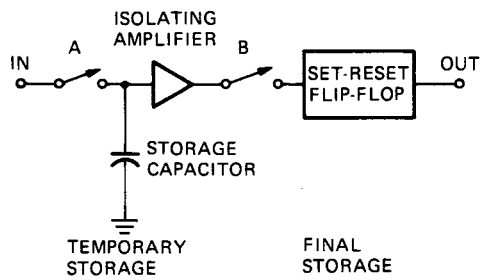
Note that any of these devices can have the clocks spaced out in time. They need not be continuous. They can be in bursts or random, so long as you don't exceed the minimum clock width and "daylight" spacing, and so long as you don't wait



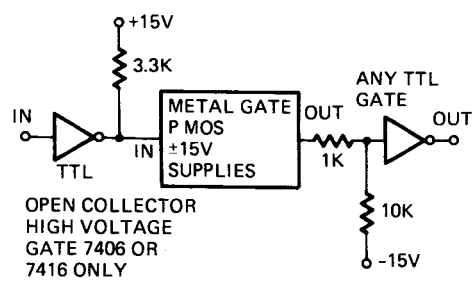
(a)

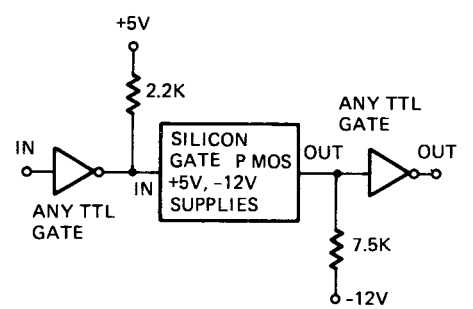


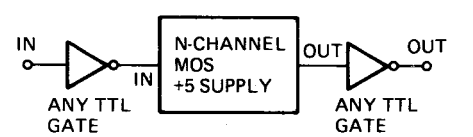
(b)



(c)







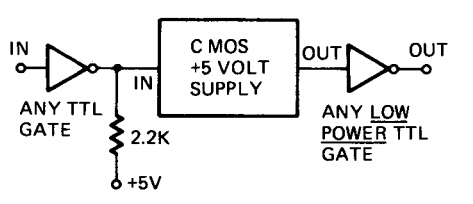


FIG. 3 (top of page)—STATIC shift register. b) DYNAMIC shift register. c) QUASI-STATIC shift register.

FIG. 4 (bottom of page)—INTERFACING DIFFERENT MOS logic with TTL gates. The type of MOS logic can be identified by the supply requirements.

longer than the dropout time on a dynamic register. Outside of the capacitance you may have to charge and discharge rapidly, *all* of the inputs on any MOS integrated circuit are essentially open circuits and neither source nor sink current.

Enables

An *enable* pin lets you combine either the outputs or inputs of a shift register group without using any fancy selector switches or external logic. Output enables are sometimes called *read enables*. You can combine memories simply by shorting all the outputs together provided you enable only one circuit at a time. Two common types of enables are the *open collector* and the *tri-state*. The latter provides a "1", a "0", or a high-impedance open circuit on command. Write enables also exist, but only on a few of the long registers.

Applications

We only have enough room to quickly run down some obvious applications of long shift registers. Two important ones were shown in the TV Typewriter story (**Radio-Electronics,** September 1973). Six recirculating 512-bit registers were used as a main memory character store and a final hex 32-bit shift register was used as a line register needed for formatting the dot matrix characters.

Pocket calculators and computers use long shift registers for number and program storage. Often, they are combined with internal multiplexing, calculation, and control circuitry into a single package.

Some music synthesizers use long shift registers as tune computers or composer storage. Several far out tricks that can be done with them is the separation of pitch and tempo, and the ability to play an upside down scale, or a reversed or backwards score. To reverse a shift register, you simply run it ahead N-1 clock pulses as fast as you can go. For instance, a 512-bit shift register can be clocked ahead 511 bits in well under a millisecond, and it appears to have backed up one slot at the end of the burst.

Long shift registers are ideal for sequence generation of noise that repeats for cryptography, computer security, music, and audio testing applications.

Long shift registers make good *buffers* or *data concentrators*. Input information can be loaded into a shift register at a random, slow, or asynchronous outside-world rate and then transferred to the rest of your circuit later on synchronously at high speed.

You can build an electrically variable delay line out of long shift registers. The clocking controls the delay time independently of the input data frequencies. You can get a delay to risetime ratio of 500:1 out of a 1024-bit register, something that's hard to do with analog delay lines. Speech compression (for talking book tapes and records), vibrato (for music synthesizers), and spectrum translation are three typical use examples.

In fancier circuits, shift registers are used as the key element in digital filters,

(continued on page 97)

MOS SHIFT REGISTERS

(continued from page 62)

correlators, and Fourier series calculators. And, as a final and obvious application, shift registers are being used to replace magnetic discs as medium-speed, high-density storage systems for computers. These are often called *silicon disc* files.

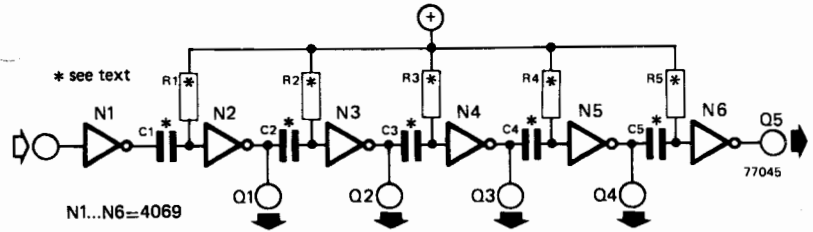
Getting started

If you are new to shift registers, pick up a few of the bargain surplus units and try experimenting with them. You'll get best results if you stick with the static units at first and avoid the older metal gate ±15 volt circuits as they are hard to interface. Remember to pick up several units at once if you are buying seconds. Above all, have the exact data sheet on hand, and if possible, some application notes as well. Be sure to have your power supplies well decoupled and regulated and make sure your clock lines and drivers *exactly* meet the specified requirements. Keep your clock pulse widths down around the minimum recommended values to minimize internal heating and try to derive the clock widths and spacing from digital logic and timing rather than using adjustable monostable delays. **R-E**

95

self-shifting register

A.M. Bosschaert



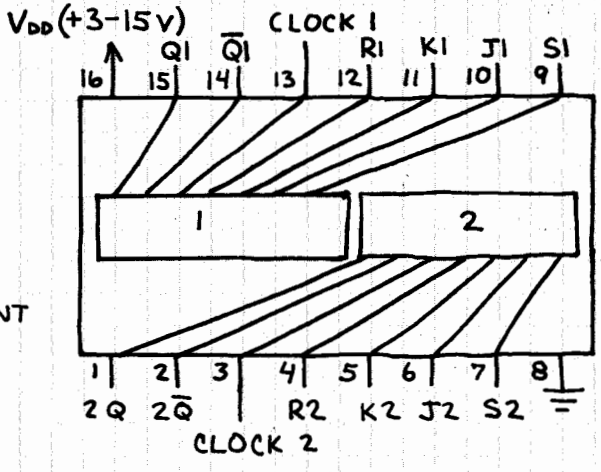
The unusual feature of this shift register is that it will transfer pulses from its input, through several stages, to the output without the need for an external clock generator. The shift speed is fixed, and is determined by the component values in the circuit.

When the input goes high, the output of N1 will go low and the input of N2 will be held low for a period determined by the time constant R1 · C1. During this time the output of N2 will be high. When the input of N2 goes high again, the output of N2 will hold the input of N3 low for a time determined by R2 · C2. In this way the pulse is shifted through the register. When the input goes low, C1 will simply discharge through R1 and the output of N1, ready for the next pulse.

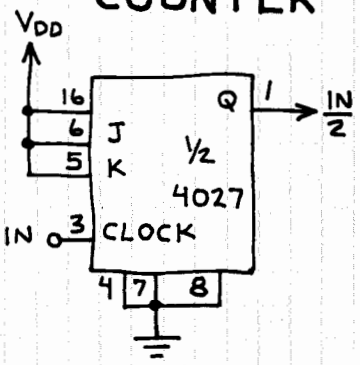
It is apparent that, provided the length of the input pulse is longer than the time constant R1 · C1, the length of the output pulse is determined solely by the circuit time constants. In general the time constants R1 · C1, R2 · C2 etc. will all be equal, and in this case the maximum input pulse rate is determined by the fact that the interval between two pulses may not be less than the time constant R1 · C1, otherwise pulses may overlap.

DUAL JK FLIP FLOP 4027

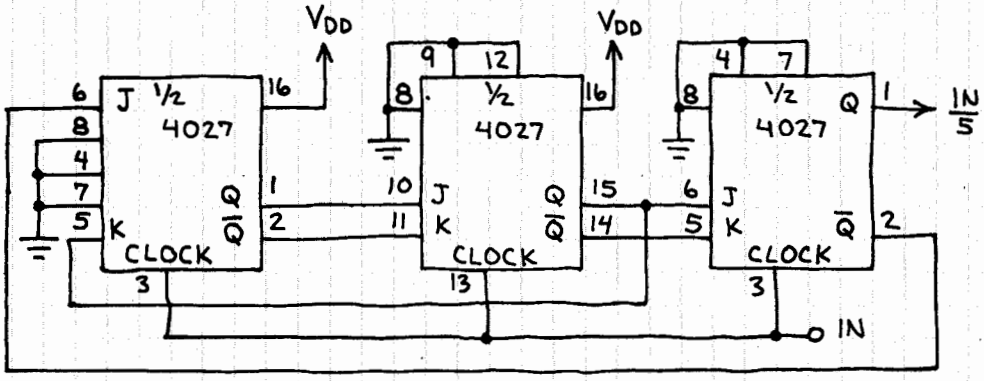
USE FOR DIVIDERS, COUNTERS AND REGISTERS. S (SET) AND R (RESET) INPUTS MUST BE LOW FOR CLOCKING TO OCCUR. MAKING S OR R HIGH SETS OR RESETS FLIP-FLOP INDEPENDENT OF CLOCK. IMPORTANT: ALL INPUTS MUST GO SOMEWHERE!



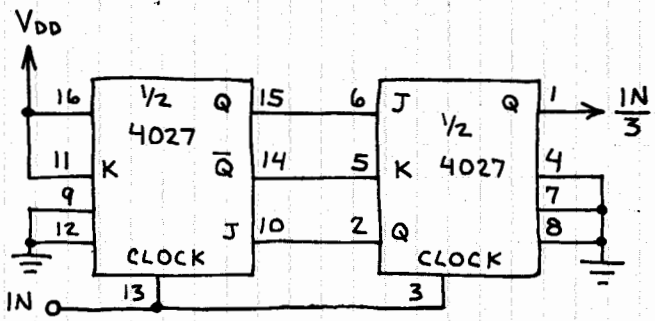
DIVIDE-BY-2 COUNTER



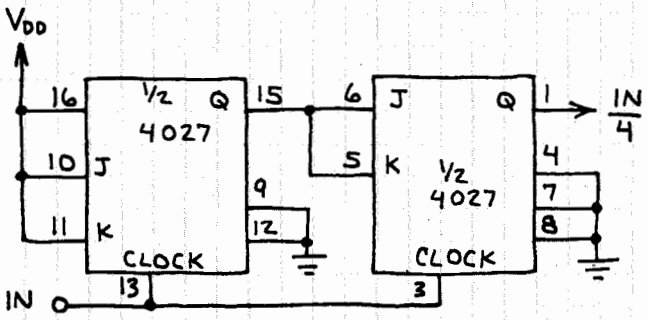
DIVIDE-BY-5 COUNTER



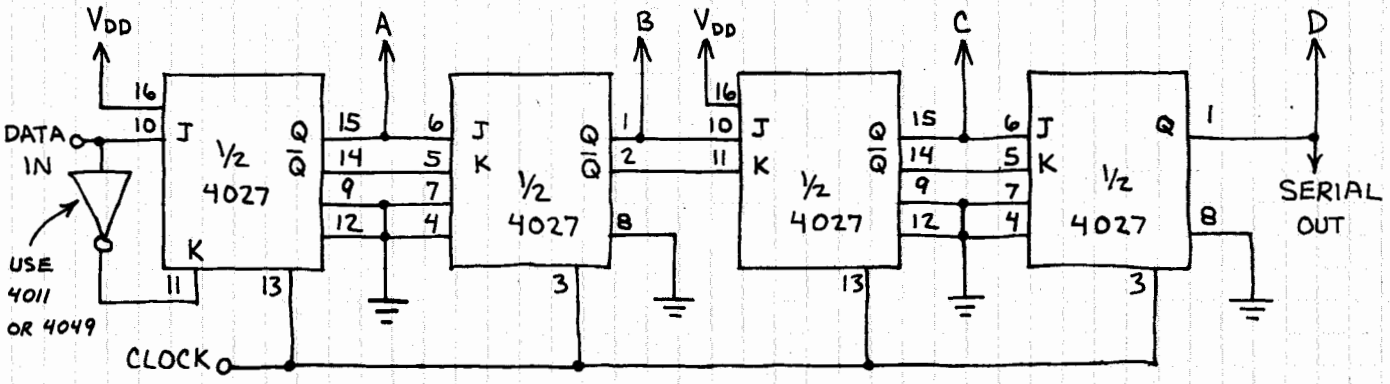
DIVIDE-BY-3 COUNTER



DIVIDE-BY-4 COUNTER

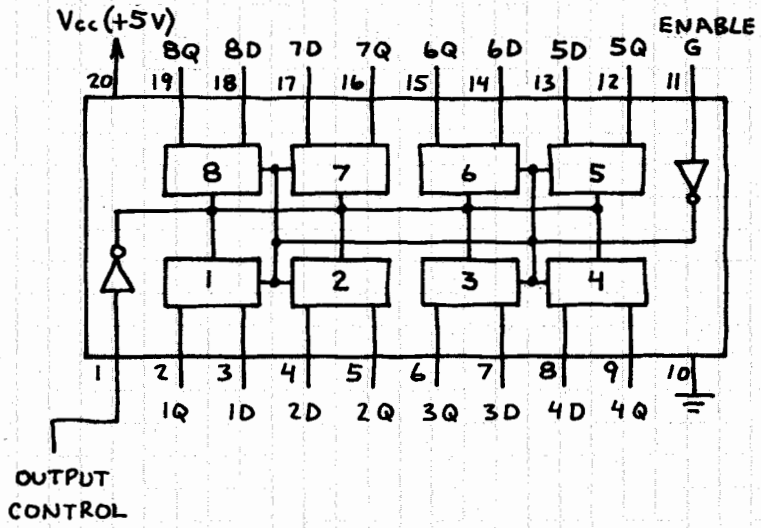


4-BIT SERIAL SHIFT REGISTER

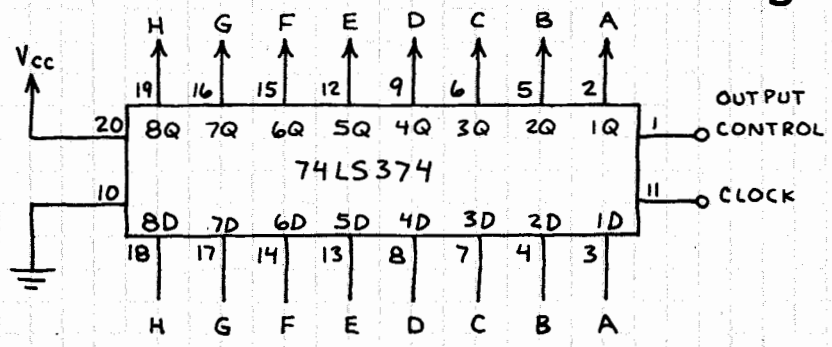


OCTAL D FLIP-FLOP 74LS374

EIGHT D-TYPE EDGE TRIGGERED FLIP-FLOPS. UNLIKE 74LS373, OUTPUTS DO NOT FOLLOW INPUTS. INSTEAD, A RISING CLOCK PULSE AT PIN 11 LOADS DATA APPEARING AT INPUTS. THIS CHIP HAS 3-STATE OUTPUTS WHICH ARE CONTROLLED BY PIN 1.



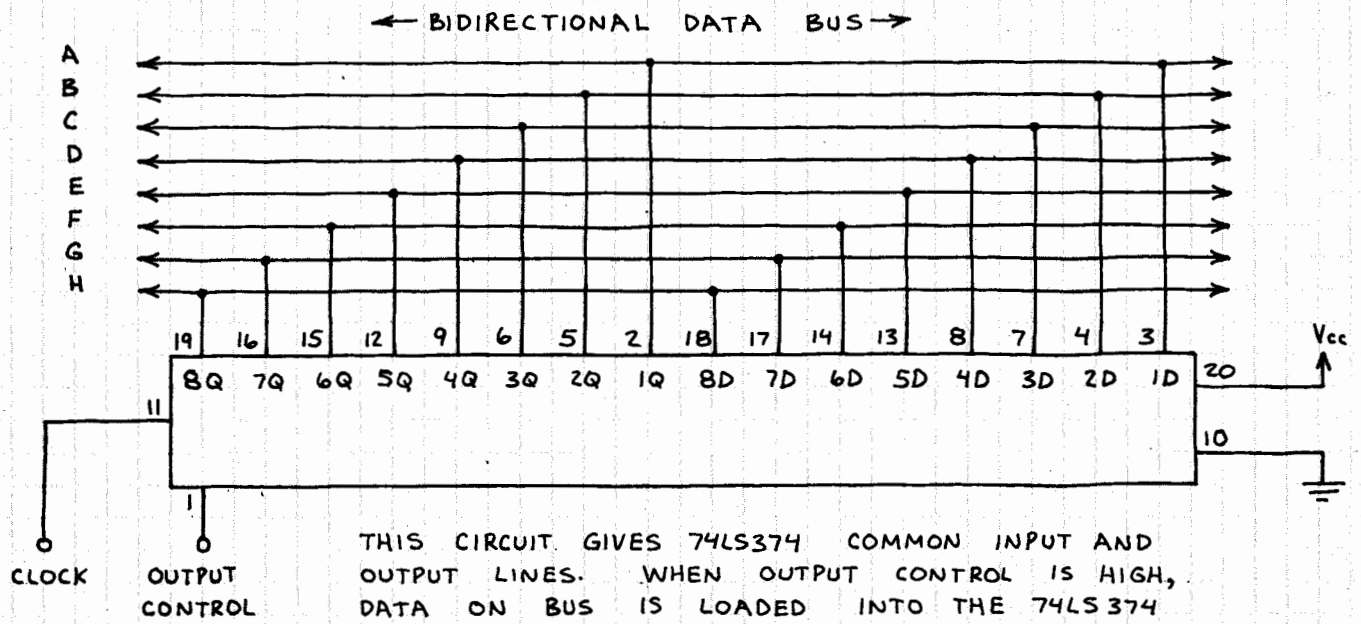
CLOCKED 3-STATE REGISTER



GENERAL PURPOSE CLOCKED REGISTER. HERE'S THE TRUTH TABLE:

OUTPUT CONTROL	CLOCK	D	Q
L	⎍	H	H
L	⎍	L	L
L	H	X	Q
H	X	X	HI-Z

COMMON INPUT/OUTPUT BUS REGISTER



THIS CIRCUIT GIVES 74LS374 COMMON INPUT AND OUTPUT LINES. WHEN OUTPUT CONTROL IS HIGH, DATA ON BUS IS LOADED INTO THE 74LS374 ON THE RISING EDGE (⎍) OF THE CLOCK PULSE. WHEN OUTPUT CONTROL IS LOW, DATA IN THE 74LS374 IS WRITTEN ONTO THE BUS.