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 elaborate code or sequence. Shift registers in a computer or calculator, intermediate to be displayed on a TV screen, numbers perhaps a "present" or "absent" com­

The data can be the letters A 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 input 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 SISO 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 march 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)
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 normal 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-
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, and 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 nearly as wide as the clock cycles, rather than the data. 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 added by 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 transfer. 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. Static registers are also easily much easier to drive on their clock lines.

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 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 write enable. 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 counters 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. 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 8:1 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.
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