Part 1

Using Serial EEPROMs

If you have a project that needs a modest amount of nonvolatile, read/write memory, serial EEPROM may be the answer. These tiny and inexpensive devices are especially useful when you need to minimize the number of I/O lines, cost, or physical size.

Probably the most common use for serial EEfigROMs is to store various types of user data - settings for remote-control devices, phone numbers, security codes, or anything that used to be set with DIP switches. Other uses include storing error codes and other diagnostic information, usage records (times, dates, counts), and instrument readings.

In some cases, serial EEPROMs can even store program code. Parallax's Basic Stamp and similar products use serial EEPROMs to store user programs in the form of BASIC-language tokens.

This article is a guide to choosing and using serial EEPROMs. I'll compare the three major interface types: Microwire, SPI, and I²C, and the pluses and minuses of using each. Next time, I'll show how to program and read all three types from a PC's standard parallel port.

The Basics

Serial EEPROMs use a synchronous interface - both the EEPROM and the chip that controls it use a common clock, and clock transitions signal when to send and read each bit. For example, a sending device may write each bit on the rising edge of the clock, and the receiving device reads the bit when it detects the clock's falling edge.

Although some other synchronous serial chips require minimum clock frequencies, the clock for serial EEPROMs can be as slow as needed, and the clock signal doesn't have to be symmetrical. The controlling device can toggle the clock at its convenience, up to the maximum speed. There's no need for a fixed timebase

In contrast, in an asynchronous link, as in most RS-232 communications, both ends of the link must agree on a frequency, and each end provides its own timebase. When the receiving device detects a Start bit, it uses its own clock to determine when to read each of the following bits. If the two clocks vary a lot, the receiving end will misread the data.

Serial EEPROMs typically have just eight pins: power and ground, one or two data/address lines, and a clock input, plus up to three other control signals. Unlike parallel EEPROMs, which add pins as the number of address and data lines grows, a serial EEPROM's physical size doesn't have to increase with capacity. The only alternative that uses less space is when you're already using a microcontroller that has EEPROM or nonvolatile RAM on-chip. But this isn't always an option, or you may need more storage than the microcontroller has.

Capacities begin at 128 bytes. As with other memory chips, the maximum capacities available have increased over time: Microchip's 24LC64 is an example of an 8-kilobyte chip.

The EEPROMs use CMOS technology, so they consume very little power, with currents as low as a few microamps in standby mode and a milliamp when active.

The synchronous interfaces aren't intended for use over long distances (for that, use RS-232 or RS-485), but cables may be as long as 4 meters in some cases, or longer if you add stronger drivers and buffers.

Depending on the device, the maximum clock speed for accessing serial EEPROMs can be over 2 Megahertz. But because it takes eight clock cycles to transfer a byte, and the master also has to send instructions and addresses, the maximum rate of data transfer is no more than around 4 microseconds per byte.

Write operations actually take much longer, because the EEPROM needs several milliseconds to program a byte into its memory array. During this time, the master can't read or write to the chip, but it can go on to other tasks that don't involve the EEPROM.

With use, EEPROMs eventually lose their ability to store data, so they're not suited for applications where the data changes constantly. These days, most are rated for a minimum of 10 million erase/write cycles, which is fine for data that changes occasionally, or even every few minutes. (For unlimited read/write cycles, use battery-backed RAM.)

Besides EEPROMs, other components with synchronous serial interfaces include A/D and D/A converters, I/O expanders, clock/calendars, and display interfaces. Multiple devices can connect to one set of lines, with each chip having its own Chip-Select line or firmware address.

There are three major types of interfaces for serial EEPROMs: Microwire, SPI, and I^2C. These give you plenty of choices, but it also means that a single interface and protocol won't work with everything. The different types vary in speed, number of signal lines, and other details.

To see how the different interfaces compare, I'll describe a 4-kilobit EEPROM of each type. Table 1 summarizes the major features, and Figure 1 shows the pinout of each. All EEPROMs with the same interface behave in a

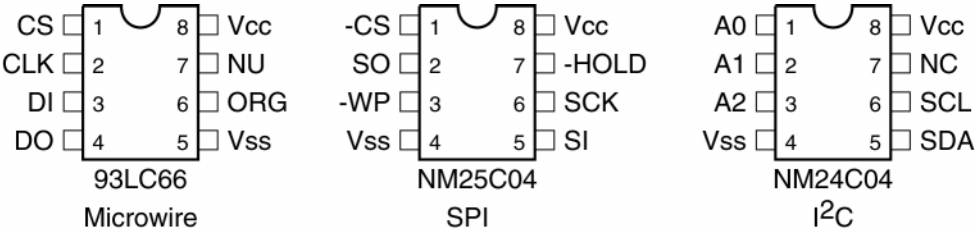

Figure 1. Pinouts for three 512-byte serial EEPROMs. The SOIC version of the 93LC66 is available in this and an alternate pinout. On the NM24C03, pin 7 is Write Protect. (NU = not used, NC = no connection)

similar way, though they may vary in the number of address bits and other details, such as whether or not there is a write-protect pin. To be sure, always read the data sheet for the chip you're using!

Microwire

Microwire is the oldest of the three interfaces. National Semiconductor introduced it, and other manufacturers now support it as well. National's COP888 is an example of a microcontroller with a Microwire interface built-in. Though it's commonly called a 3-wire interface, a complete link actually needs four signal lines plus a common ground.

Microchip's 93LC66 is a 4-kilobit serial EEPROM with a Microwire interface. It has two data pins: DI (data in) and DO (data out), a clock input (CLK), and a chip-select (CS).

Additional inputs are for memory configuration (ORG), which determines whether data format is 8 or 16 bits, and program enable (PE), which must be high to program the chip. Setting ORG high saves time because you can program and read two bytes with one instruction.

The EEPROM understands seven instructions: Erase/Write Enable and Disable, Write, Read, Erase, Erase All (sets all bits to 1), and Write All (writes one byte to all locations).

Table 1. Comparison of Microwire, SPI, and I²C serial EEPROMs

Interface	Microwire	SPI	I²C
Example device (512-byte capacity)	93LC66	NM25C04	NM24C04
Source	Microchip	National Semi	National Semi
Minimum number of interface (excluding GND)	4	4	2
Data width (bits)	8 or 16	8	8
Maximum clock speed (Mhz)	2	2.1	0.4
Write (busy) time (millisecs., max)	10	5	10
Maximum number of bytes programmed in one operation	2	4	16
Writes bit on (clock state)	rising edge	rising edge	low level
Reads bit on (clock state)	rising edge	falling edge	low level
Output low current (min.)	2.1mA@0.4V	1.6mA@0.4V	3mA@0.4V
Output high current (min.)	0.4mA@2.4V	0.8mA@Vcc-0.8V	(open collector)
Chip-select method	hardware	hardware	software
Write-protect method	software	hardware & software	none ('24C03 has hardware write-protect for upper half)

Figure 2 shows the timing for byte read and write operations. Each instruction must begin with a Start condition, which occurs when CS and DI are both high on CLK's rising edge. This occurs naturally when an instruction is written, because all of the instructions begin with 1. The master must bring CS low after each instruction, except sequential reads. When CS is high, the EEPROM is in Standby mode, ignoring all communications until it detects a new Start condition.

To write to, or program, the EEPROM, the master must first write an Erase/Write Enable instruction to DI, followed by a Write instruction, the

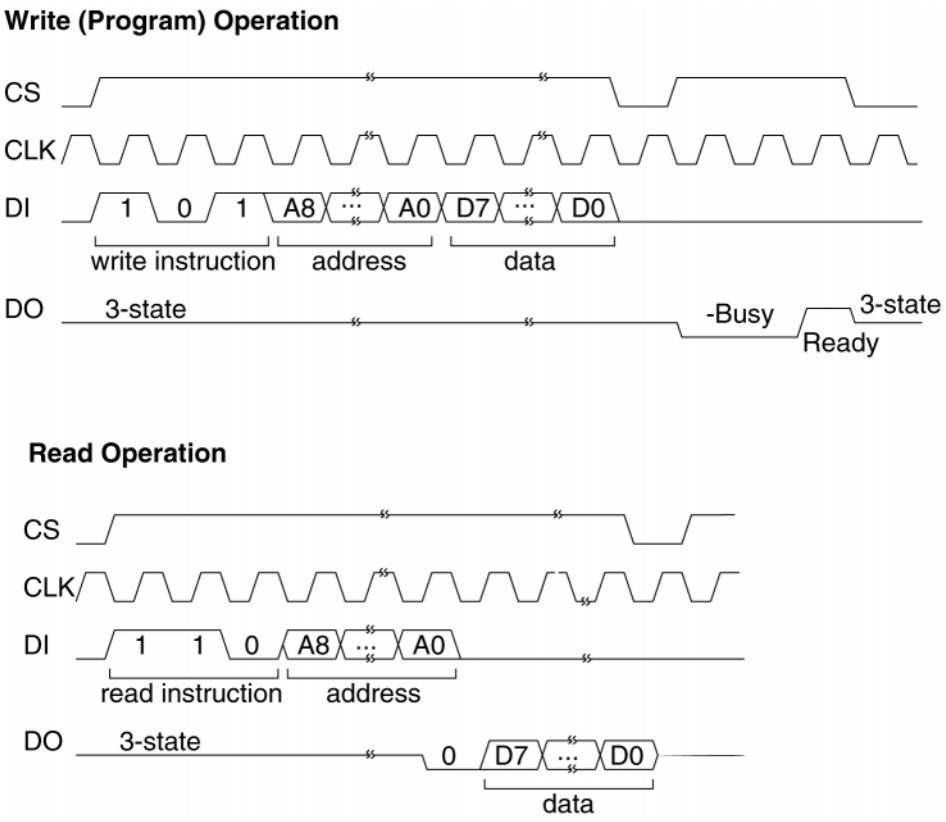

Figure 2. Byte read and write operations on a Microwire 93LC66 EEPROM, configured for 8-bit organization. CLK's rising edges latch inputs at DI and clock data out at DO.

master writes bits on CLK's falling edge, and the EEPROM latches each bit on the next rising edge.

After sending the final data bit in a programming operation, the master must bring CS low before the next rising edge of CLK. This causes the EEPROM to begin its internal programming cycle. (Some devices, such as Microchip's 93C76, don't require CS to go low here; instead, they begin programming on CLK's rising edge after D0.) The programming is self-timed; it requires no clock cycles. If CS returns high before the pro-

gramming cycle completes, D0 will indicate Ready/-Busy status. CS must then go low again to complete the write operation.

The master needs to send the Erase/Write Enable instruction just once per programming session. The device remains write-enabled until it receives an Erase/Write Disable instruction or power is removed.

To read from the EEPROM, the master writes a Read instruction to DI, followed by the address to read. When the EEPROM receives the final address bit, it writes a "dummy zero" to D0, then writes the requested data on CLK's rising edges.

If CS remains high after a Read operation, additional clock transitions will cause the chip to continue to output data at sequential addresses. If CS goes low, the next read operation must begin with the Read instruction and an address.

A 2-line interface is sometimes possible by connecting DO and DI. (An isolation resistor between the lines is recommended.) The EEPROM's DO output is high impedance except when sending data or busy status. In a 2-line interface, the master's DI output must also be high impedance when DO is active. However, there is a brief bus conflict in Read operations, when the EEPROM outputs the dummy zero on receiving the final address bit, and the master's driver must be strong enough to pull the combined DO/DI line high when this occurs and the address bit is 1.

SPI

The second interface type, SPI (Serial Peripheral Interface), originated at Motorola, which includes an SPI interface on its 68HC11 and other microcontrollers. SPI is much like Microwire, though the signal names, polarities, and other details vary.

Like Microwire, SPI is often referred to as a 3-wire interface, though a read/write interface actually requires two data lines, a clock, a chip select, and a common ground.

The signal names differ on the master and slave devices. The data line MOSI (master out, slave in) on the master connects to SI on the slave, and MISO (master in, slave out) on the master connects to SO on the slave. The clock is called SPICK at the master and SCK at the slave. A Master may also have four outputs (SS0 through SS3) that each connect to a chip-select (-CS) on up to four slave devices.

As with Microwire, SPI EEPROMs write bits on the clock's rising edge, but unlike Microwire, they latch input bits on the falling edge. (The SPI protocol actually allows two different clock polarities, but the EEPROMs support only clock polarity = 0.)

An SPI device also has a choice of two phases. The EEPROMs support only clock phase 1, which normally means that the chip-select can remain low if the interface has only one slave device. On the EEPROMs, however, -CS still has to toggle to reset the serial logic between instructions, so you can't just tie it low. Notice also that the polarity of -CS (active low) is opposite from Microwire's convention.

National's NM25C04 is a 4-kilobit EEPROM with an SPI interface. In addition to the four lines mentioned above, the chip has two other inputs. -WP (write protect) must be high to program the device. And for interfaces with multiple slaves, the -Hold input enables the master to pause in the middle of a transfer in order to do something more urgent on the SPI link. The EEPROM ignores all activity on the SPI bus until -Hold returns high and both devices pick up where they left off.

The EEPROM understands six instructions: Set and Reset the Write Enable Latch, Read and Write to the Status Register, and Read and Write to the Memory Array.

The chip has several levels of write protection, which you can use to virtually guarantee that there will be no inadvertent writes to the device. If -WP is low, no changes to the data are allowed. If -WP is high, two nonvolatile bits in the chip's Status Register can block writes to all or a portion of the device. And finally, if -WP is high, before you can write to the Status Register or the portion of memory enabled in the Status Register, the EEPROM must receive a Set Write Enable Latch instruction.

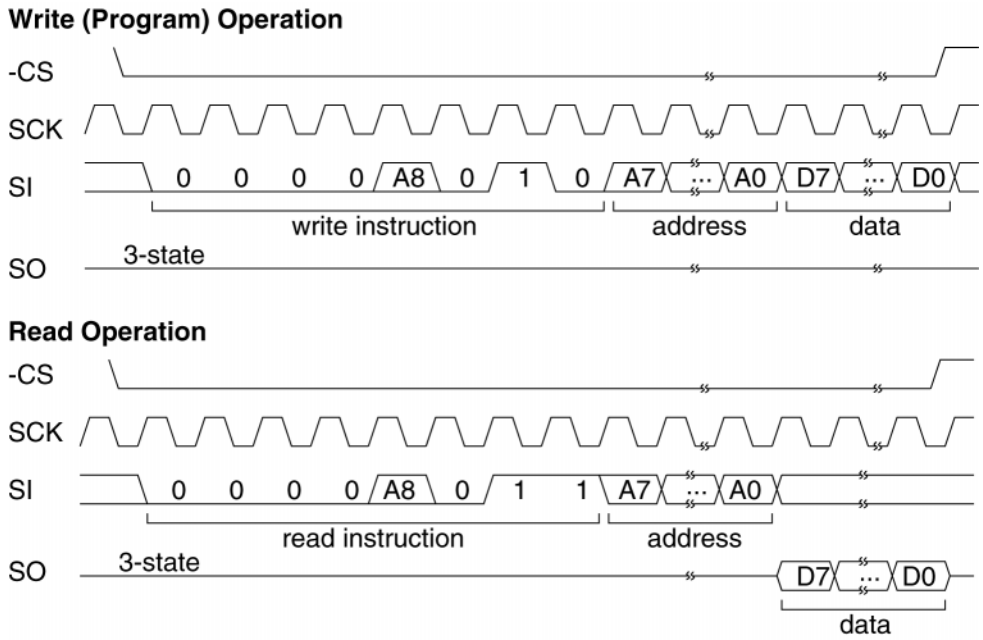

Figure 3. The NM25C04's SPI interface is similar to Microwire's. SCK's falling edges latch inputs at SI, while rising edges clock data out on SO.

Figure 3 shows the timing for byte reads and writes for the '25C04. To write to the EEPROM, the master writes a Set Write Enable Latch instruction to SI, followed by a Write instruction (which contains address bit A8), then the lower eight address bits, then the data to write. The master may send up to four data bytes for sequential addresses in one operation. After clocking the final data bit, with SCK low, CS must go high to begin programming the byte into the EEPROM.

While the EEPROM is programming the data, the master can read the EEPROM's Status register. When bit 0 of the Status Register is 0, the EEPROM has finished programming, and the next write operation can begin. The chip is write-protected after each programming operation, so each write must begin with a Set Write Enable Latch instruction.

To read the EEPROM, the master sends a Read instruction, which contains bit A8 of the address to read, then bits A7-A0. The EEPROM responds with the data bits in sequence on SO. As with Microwire, additional clocks will cause the EEPROM to send additional data bytes in sequence.

For larger capacities, rather than embedding address bits in instructions, the master sends a 16-bit address.

I²C

I²C (Inter-Integrated Circuit Bus) is the final interface type. This one originated with Philips/Signetics, whose 8XC528 (8051 family) is an example of a microcontroller with an I²C interface built-in.

The I²C interface requires just two signal lines, plus a common ground. Serial Data/Address (SDA) is a bidirectional line that requires open-collector or open-drain outputs. Serial Clock (SCL) is the clock. Instead of a chip-select line, the master sends a slave address on SDA. Figure 4 shows the timing for I²C transfers.

An I²C bus can have up to about 40 devices, with the limit determined by a maximum bus capicitance of 400 pF. Each device on the bus can have an address of up to 7 bits.

The open-collector/open-drain interface means that any logic-low output will pull SDA low. A device releases the SDA line by writing 1 to its output.

Unlike Microwire and SPI, which are edge-sensitive, I²C is level-sensitive. Data and address bits on SDA may change only while SCL is low, and the receiving device reads bits after SCL goes high.

There are two occasions when SDA changes state while SCL is high. A Start condition signals the beginning of an operation, and occurs when the master brings SDA low with SCL high. A Stop condition signals the end of an operation, and occurs when SDA goes high with SCL high.

Unlike Microwire and SPI, which offer no feedback except the success or failure of an entire read or write operation, I²C devices send Acknowledge

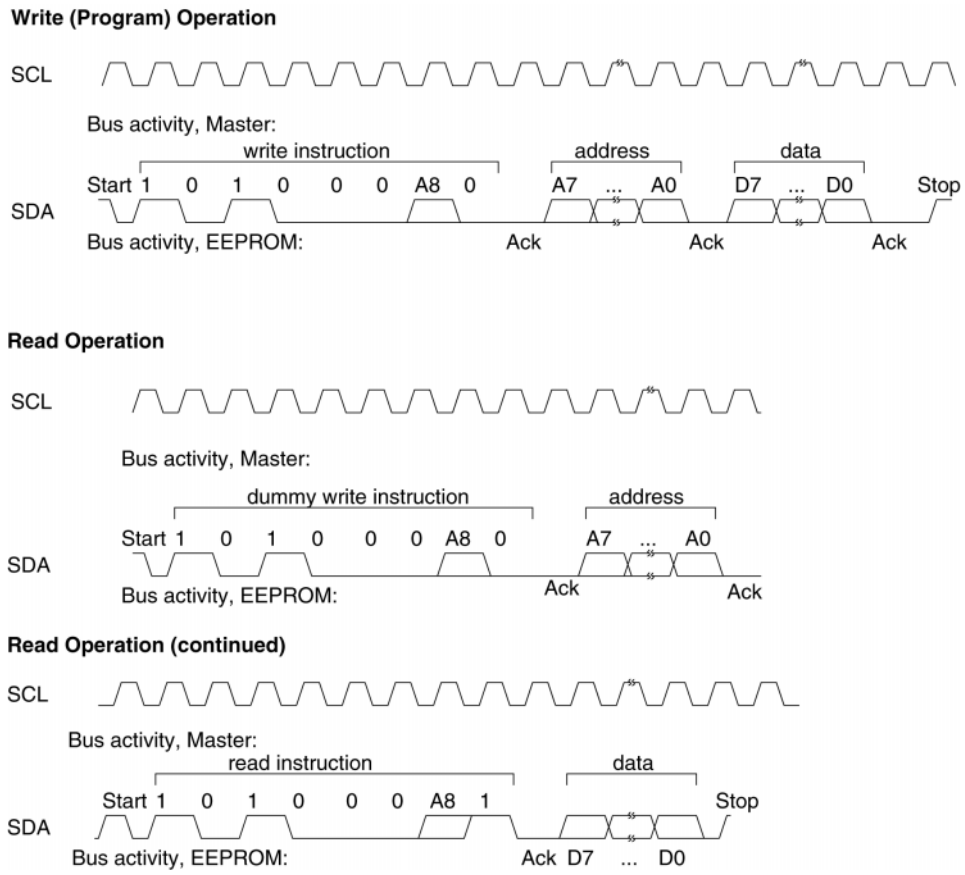

Figure 4. The NM24C04's I²C interface uses a single bidirectional signal line.

signals after receiving eight bits. After most transmissions, whether an instruction, address, or data byte, during the ninth clock cycle, the transmitting device releases SDA and the receiving device pulls SDA low to acknowledge that it received the bits. If the master doesn't see the acknowledge, it knows that something isn't right. The receiving device releases SDA on SCL's next falling edge.

An I²C bus can have multiple masters. If more than one master tries to control the bus at once, an arbitration protocol defined by the I²C standard determines which one wins.

National Semiconductor's NM24C04 is a 4-kilobit EEPROM with an I²C interface. SDA, SCL, power, and ground use four of the eight pins. Three other pins are seldom-used page-address inputs (A0, A1, A2). These allow multiple low-density EEPROMs on a single interface. For example, a link could have eight 256-byte EEPROMs, each with a different hard-wired page address. On the '24C04, A0 is unused and A1 and A2 can select any of up to four 512-byte devices. But it's simpler and cheaper to use a single 2-kilobyte chip.

On some EEPROMs (NM24C03), the remaining pin is a hardware write-protect for the upper half of the memory array.

To write a byte to the EEPROM, the master issues a Start condition, then writes an 8-bit slave address. The address consists of a 4-bit type identifier (1010 for EEPROMs), followed by the selected page (000 or 001), and a request to read (1) or write to (0) the device.

The type identifier is defined by the I²C standard. The page address is a way of specifying address bit A8 (0 or 1). The other two bits in the page address are unused unless there are multiple EEPROMs. In the clock cycle following the slave address, the EEPROM pulls SDA low to acknowledge.

The master then sends an 8-bit address, and the EEPROM acknowledges. (When necessary for larger capacities, the master can send two address bytes.) The master sends the byte to write, waits for an acknowledge, then issues a stop condition. The EEPROM then programs the data into its memory array. When programming is complete, the EEPROM acknowledges.

To write to up to 16 bytes to sequential addresses, instead of issuing a Stop condition after the first data byte, the master may continue to send data bytes, waiting for an acknowledge after each. After sending all of the bytes, the master issues the Stop condition, and the EEPROM programs the bytes and acknowledges. Some I²C EEPROMs (Microchip's 24LC04) can pro-

gram all 16 bytes in parallel, for much faster programming. On others, you can write 16 bytes in sequence, but the chip programs them one at a time.

To read a byte, the master begins as if doing a write operation, sending a slave address followed by a byte address. When the EEPROM acknowledges the byte address, the master issues a new Start condition, followed by the slave address with the final bit set to 1 (read). The slave acknowledges, then writes the data to SDA. On receiving the data, the master doesn't acknowledge, but instead issues a Stop condition.

To read sequential addresses, the master acknowledges receiving the data byte, and the EEPROM responds by sending the next byte in sequence. The EEPROM will continue to send bytes until it receives a Stop condition instead of an Acknowledge.

Decisions

Which EEPROM type should you use? Sometimes the choice is obvious, for example, if you're using a microcontroller with a built-in interface, or you want to use a specific A/D converter in the link.

All three types are easily available and inexpensive. Digi-key has 512-byte devices of each type for under $3 in single quantities.

I^2C is best if you have just two signal lines to spare, or if you have a cabled interface (I^2C has the strongest drivers).

If you want a clock faster than 400 kilohertz, use Microwire or SPI.

For more on using serial EEPROMs, I recommend browsing the manufacturers' pages on the web, especially these sites:

National Semiconductor
http://www.national.com/design/
Many application notes on Microwire.

Motorola Semiconductor
http://www.mcu.motsps.com/mc.html
Microcontroller references contain SPI documentation.

Microchip Technology
http://www.microchip2.com/appnotes/appnotes.htm
Notes on SPI use.

Philips Semiconductor
http://www.semiconductors.philips.com/
I²C information.

Next time, I'll present the design of an EEPROM programmer that runs from a PC's parallel port, with Visual-Basic program code.

***

This 2-part series first appeared in *Circuit Cellar Ink* magazine.

Jan Axelson is the author of *USB Complete, Serial Port Complete, Parallel Port Complete* and *The Microcontroller Idea Book*. You can reach her by email at jan@lvr.com, or via her web site at http://www.lvr.com.