AT89LP In-System Programming Specification

1. Overview

The Atmel AT89LP microcontrollers feature 2K bytes to 64K bytes of on-chip Flash program memory. Some devices may also support Flash data memory. In-System Programming (ISP) allows programming and reprogramming of any AT89LP microcontroller positioned inside the end system. Using a simple 4-wire SPI interface, the In-System programmer communicates serially with the AT89LP microcontroller, reprogramming all nonvolatile memories on the chip. In-System programming eliminates the physical removal of chips from the system. This will save time, and money, both during development in the lab, and when updating the software or parameters in the field. This application note describes how to program the Flash program or data memory on an AT89LP microcontroller using the In-System programming interface. This document applies to all AT89LP microcontrollers with 64K bytes or less of code memory with the exception of AT89LP2052 and AT89LP4052, which follow a slightly different ISP protocol (see "Programming the AT89LP2052/LP4052" on page 27).

Table 1-1. AT89LP Family Features

Four-Wire SPI Programming Interface
Active-low Reset Entry into Programming
Slave Select Allows Multiple Devices on Same Interface
Programming Support for up to 64K Bytes of Code Memory
Programming Support for up to 64K Bytes of Data Memory
User Signature Row
Flexible Page Programming
Row Erase Capability
Page Write with Auto-Erase Command
Programming Status Register

2. The Programming Interface

In-System programming utilizes the Serial Peripheral Interface (SPI) of an AT89LP microcontroller. The SPI is a full-duplex synchronous serial interface consisting of four wires: Serial Clock (SCK), Master-In/Slave-Out (MISO), Master-Out/Slave-In (MOSI), and an active-low Slave Select ($\overline{SS}$). **Note:** The AT89LP ISP interface uses the SPI clock mode 0 (CPOL = 0, CPHA = 0) exclusively. When programming an AT89LP device, the programmer always operates as the SPI master, and the target system always operates as the SPI slave. To enter or remain in In-System programming mode the device's reset line ($\overline{RST}$) must be held active (low). With the addition of VCC and GND, an AT89LP microcontroller can be programmed with a minimum of seven connections as shown in Figure 2-1 and Table 2-1.

AT89LP In-System Programming

Application Note

3593A–MICRO–7/06

Figure 2-1. Device Connections Required for Programming

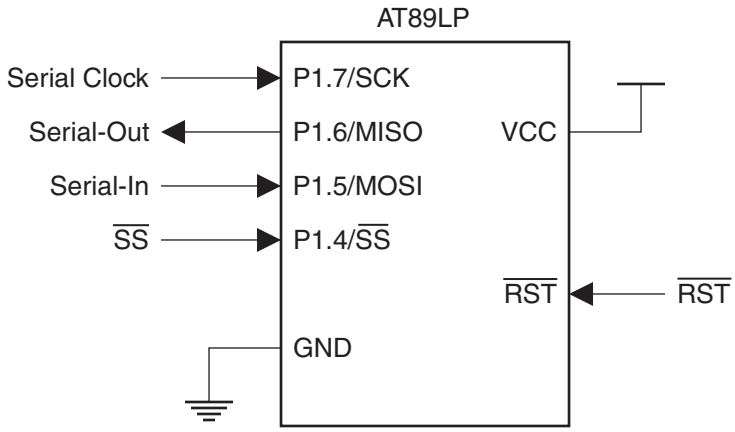


Table 2-1. Connections Required for Programming

Pin	Name	Comment
SCK (P1.7)	Serial Clock	Programming clock generated by the programmer (master). Serial bits on MISO are output on the falling edge of SCK. Serial bits on MOSI are sampled at the rising edge of SCK.
MISO (P1.6)	Serial Output	Communication line from the target AT89LP being programmed (slave) to the programmer (master).
MOSI (P1.5)	Serial Input	Communication line from the programmer (master) to the target AT89LP being programmed (slave).
$\overline{SS}$ (P1.4)	Slave Select	Active-low select of the target AT89LP being programmed. Must be driven low to enable communication on the SPI pins.
$\overline{RST}$	Reset	To enable In-System programming, the reset of the target AT89LP must be kept active. To simplify this, the In-System programmer should control the target Reset.
GND	Ground	The programmer and target AT89LP systems must share the same common ground to ensure correct communication.
VCC	Power Supply	To allow programming of targets operating at any voltage, the In-System programmer can draw power from the target. Alternatively, the target can have power supplied through the In-System programming connector for the duration of the programming cycle

The In-System programming Interface is the only means of externally programming an AT89LP microcontroller. The ISP Interface can be used to program a device both in-system and in a stand-alone serial programmer. The ISP Interface does not require any clock other than SCK and is not limited by the system clock frequency. During In-System programming, the system clock source of the target device can operate normally. For stand-alone programmers the XTAL1 pin should be driven low to prevent the oscillator input from floating.

2.1 Hardware Design Considerations

To allow In-System programming of an AT89LP microcontroller, the In-System programmer must be able to override the pin functionality during programming. This section describes the details of each pin used for the programming operation.

2.1.1 GND

The In-System programmer and target system need to operate with the same reference voltage. This is done by connecting ground of the target to ground of the programmer. No special considerations apply to this pin.

2.1.2 VCC

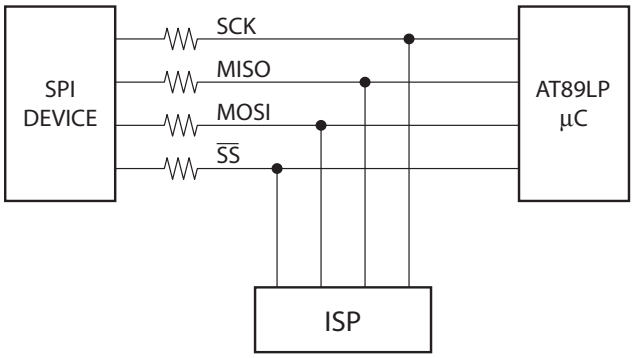
When programming the target microcontroller, the programmer outputs need to stay within the ranges specified in the DC Characteristics. To easily adapt to any target voltage, the programmer can draw all power required from the target system. As an alternative, the target system can have its power supplied from the programmer through the same connector used for the communication. This would allow the target to be programmed without applying power to the target externally.

2.1.3 RESET

The target AT89LP microcontroller will enter programming mode only when its reset line ($\overline{RST}$) is active (low). To simplify this operation, it is recommended that the target reset can be controlled by the In-System programmer.

Immediately after Reset has gone active, the In-System programmer will start to communicate on the dedicated SPI wires SCK, MISO, MOSI, and $\overline{SS}$. To avoid driver contention, a series resistor should be placed on each of the four dedicated lines if there is a possibility that external circuitry could be driving these lines. The connection is shown in Figure 2-2. The value of the resistors should be chosen depending on the circuitry connected to the SPI bus. Note that the AT89LP microcontroller will automatically set all its I/O pins to inputs when Reset is active.

Figure 2-2. Connecting ISP to Target SPI Bus



To avoid problems, the In-System programmer should be able to keep the entire target system Reset for the duration of the programming cycle. The target system should never attempt to drive the four SPI lines while Reset is active.

In some AT89LP microcontrollers, the Reset may be disabled to gain an extra I/O pin. In these cases, the $\overline{RST}$ pin will always function as a reset during power up. To enter programming, the $\overline{RST}$ pin must be driven low prior to the end of Power-On Reset (POR). After POR has completed, the device will remain in ISP mode until $\overline{RST}$ is brought high. Once the initial ISP session has ended (by bringing $\overline{RST}$ high), the power to the target device must be cycled OFF and ON to enter another programming session.

2.1.4 SLAVE SELECT

When programming an AT89LP microcontroller, the In-System programmer uses the slave select ($\overline{SS}$) pin to control the Serial Peripheral Interface (SPI). This pin is always driven by the programmer, and the target system should never attempt to drive this wire when the target reset is active. Immediately after the Reset goes active, this pin should be driven high by the programmer. While $\overline{SS}$ is high, the target device will ignore clock and data on SCK and MOSI, MISO will remain tristated, and the ISP interface is reset to its default state. While $\overline{SS}$ is low, the target device can receive a command and output data. $\overline{SS}$ should be driven low before the programmer issues a command and should return high after the command has been transmitted. The $\overline{SS}$ signal maintains synchronization between the programmer (master) and target (slave), and defines each command frame. If a target system has multiple AT89LP devices which must be programmable by the same programmer, the $\overline{SS}$ pin can be used to enable only a single device at a time, provided that the $\overline{SS}$ pin for each device can be driven independently.

The target AT89LP microcontroller will always set up its $\overline{SS}$ pin to be an input whenever Reset is active. See also the description of “RESET” on page 3 pin.

2.1.5 SCK

When programming an AT89LP microcontroller, the In-System programmer supplies the clock waveform on the SCK pin. This pin is always driven by the programmer, and the target system should never attempt to drive this pin when the target reset is active. The programmer should always drive SCK low before $\overline{SS}$ is brought low and SCK should remain low while $\overline{SS}$ is brought high.

The target AT89LP microcontroller will always set up its SCK pin to be an input whenever Reset is active. See also the description of “RESET” on page 3 pin.

2.1.6 MISO

When Reset is applied to the target AT89LP microcontroller, the MISO pin is set up to be an input. Only after the “Programming Enable” command has been correctly transmitted to the target will the target AT89LP microcontroller set its MISO pin to become an output. MISO will remain tristated while $\overline{SS}$ is high and will only output data if $\overline{SS}$ is low. Serial data bits on MISO change at the falling edge of SCK.

2.1.7 MOSI

When programming an AT89LP microcontroller, the In-System programmer supplies data to the target on the MOSI pin. This pin is always driven by the programmer, and the target system should never attempt to drive this pin when the target reset is active. The ISP interface samples serial data bits from MOSI on the rising edge of SCK.

The target AT89LP microcontroller will always set up its MOSI pin to be an input whenever Reset is active. See also the description of “RESET” on page 3 pin.

2.2 Interface Timing

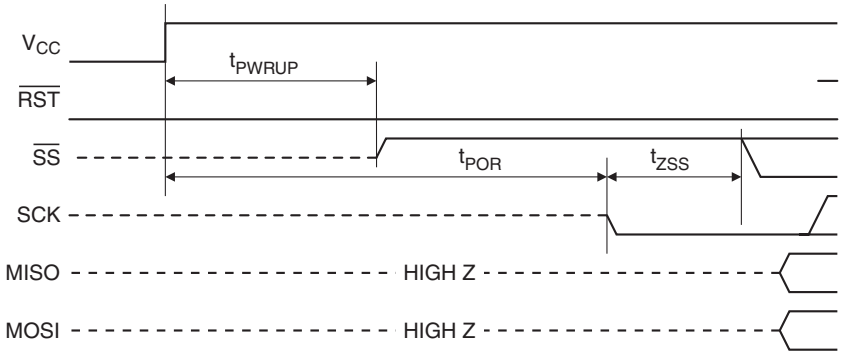
This section details general system timing sequences and constraints for entering or exiting In-System programming as well as parameters related to the Serial Peripheral Interface during ISP. The values of some timing parameters may differ between different members of the AT89LP family. For the specific timing requirements of an AT89LP device, see that device’s datasheet. The general timing parameters for the following waveform figures are listed in Section “Timing Parameters” on page 8.

2.2.1 Power-up Sequence

Execute this sequence to enter programming mode immediately after power-up. In some instances this is the only method to enter programming (see “RESET” on page 3).

1. Apply power between VCC and GND pins. $\overline{RST}$ should remain low.
2. Wait at least t_{PWRUP} and drive $\overline{SS}$ high.
3. Wait at least t_{POR} for the internal Power-on Reset to complete. The value of t_{POR} will depend on the current settings of the target device.
4. Start programming session.

Figure 2-3. Serial Programming Power-up Sequence

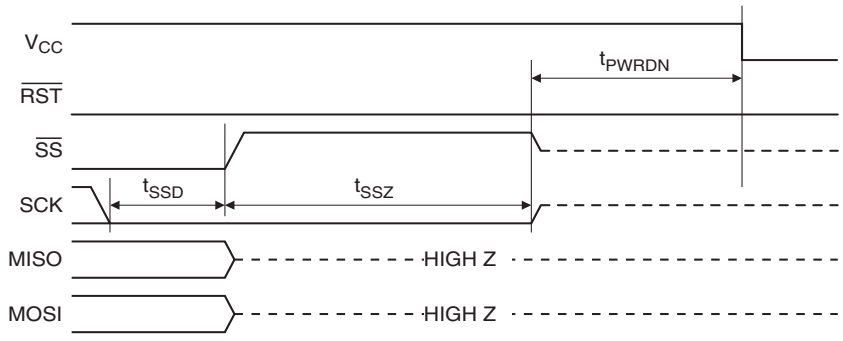


2.2.2 Power-down Sequence

Execute this sequence to power-down the device **after** programming.

1. Drive SCK low.
2. Wait at least t_{SSD} and bring $\overline{SS}$ high.
3. Tristate MOSI.
4. Wait at least t_{SSZ} and then tristate $\overline{SS}$ and SCK.
5. Wait no more than t_{PWRDN} and power off VCC.

Figure 2-4. Serial Programming Power-down Sequence



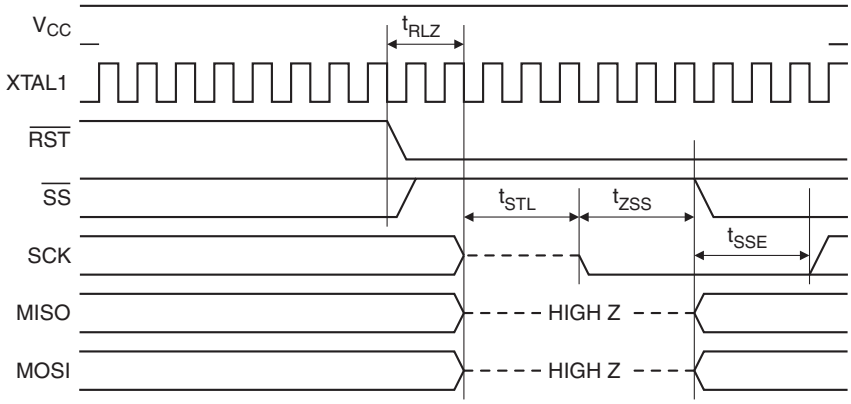
Note: The waveforms on this page are not to scale.

2.2.3 ISP Start Sequence

Execute this sequence to exit CPU execution mode and enter ISP mode when the device has passed Power-On Reset and is already operational.

1. Drive $\overline{RST}$ low.
2. Drive $\overline{SS}$ high.
3. Wait $t_{RLZ} + t_{STL}$.
4. Start programming session.

Figure 2-5. In-System Programming Start Sequence

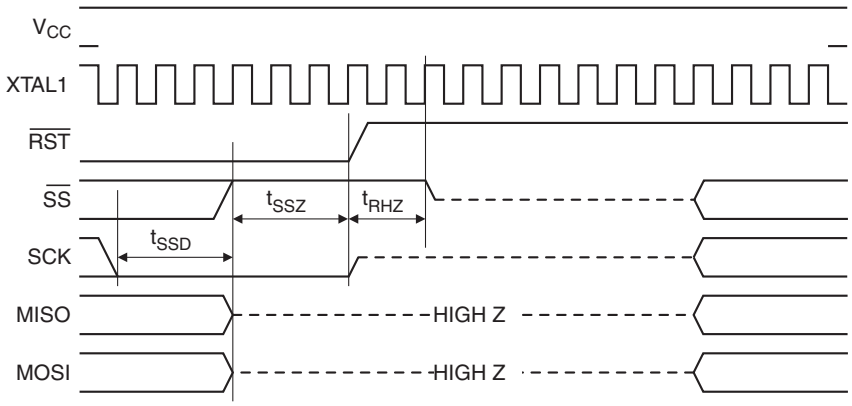


2.2.4 ISP Exit Sequence

Execute this sequence to exit ISP mode and resume CPU execution mode.

1. Drive SCK low.
1. Wait at least t_{SSD} and drive $\overline{SS}$ high.
2. Tristate MOSI.
3. Wait at least t_{SSZ} and bring $\overline{RST}$ high.
4. Tristate SCK.
5. Wait t_{RHZ} and tristate $\overline{SS}$.

Figure 2-6. In-System Programming Exit Sequence



Note: The waveforms on this page are not to scale.

2.2.5 Serial Peripheral Interface

The Serial Peripheral Interface is a byte-oriented full-duplex synchronous serial communication channel. During In-System programming, the programmer always acts as the SPI master and the target device always acts as the SPI slave. The target device receives serial data on MOSI and outputs serial data on MISO. The Programming Interface implements a standard SPI Port with a fixed data order and for In-System programming, bytes are transferred MSB first as shown in Figure 2-7. The SCK phase and polarity follow SPI clock mode 0 (CPOL = 0, CPHA = 0) where bits are sampled on the rising edge of SCK and output on the falling edge of SCK. For more detailed timing information see Figure 2-8.

Figure 2-7. ISP Byte Sequence

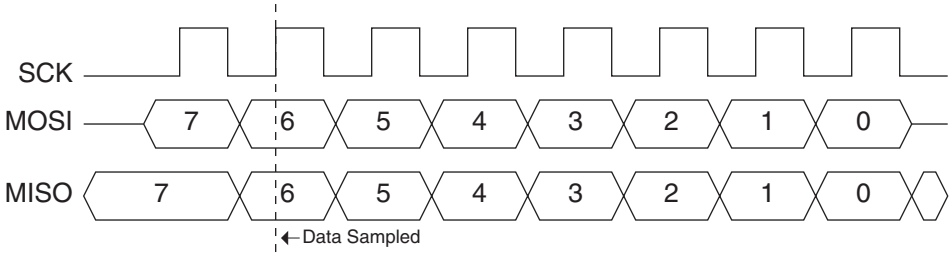
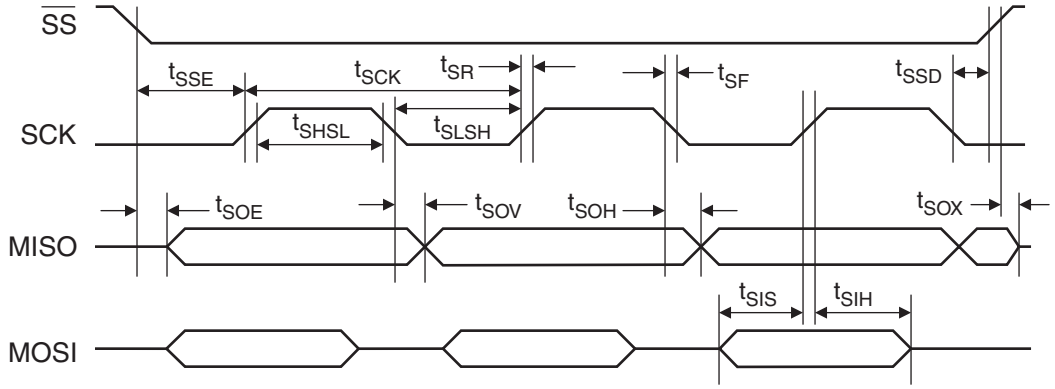


Figure 2-8. Serial Programming Interface Timing



2.2.6 Timing Parameters

The timing parameters for Figures 2-3 through 2-6, and Figure 2-8 are shown in Table 2-2.

Table 2-2. Programming Interface Timing Parameters

Symbol	Parameter	Min	Max	Units
t_{CLCL}	System Clock Cycle Time	Device Dependent		ns
t_{PWRUP}	Power-on to $\overline{SS}$ High Time	10		µs
t_{POR}	Power-on Reset Time	Device Dependent		ns
t_{PWRDN}	$\overline{SS}$ Tristate to Power Off		1	µs
t_{RLZ}	$\overline{RST}$ Low to I/O Tristate	t_{CLCL}	2 t_{CLCL}	ns
t_{STL}	$\overline{RST}$ Low Settling Time	100		ns
t_{RHZ}	$\overline{RST}$ High to $\overline{SS}$ Tristate	0	2 t_{CLCL}	ns
t_{SCK}	Serial Clock Cycle Time	Device Dependent(1)		ns
t_{SHSL}	Clock High Time	Device Dependent		ns
t_{SLSH}	Clock Low Time	Device Dependent		ns
t_{SR}	Rise Time	Device Dependent		ns
t_{SF}	Fall Time	Device Dependent		ns
t_{SIS}	Serial Input Setup Time	10		ns
t_{SIH}	Serial Input Hold Time	10		ns
t_{SOH}	Serial Output Hold Time		10	ns
t_{SOV}	Serial Output Valid Time		35	ns
t_{SOE}	Output Enable Time		10	ns
t_{SOX}	Output Disable Time		25	ns
t_{SSE}	$\overline{SS}$ Enable Lead Time	t_{SLSH}		ns
t_{SSD}	$\overline{SS}$ Disable Lag Time	t_{SLSH}		ns
t_{ZSS}	SCK Setup to $\overline{SS}$ Low	25		ns
t_{SSZ}	SCK Hold after $\overline{SS}$ High	25		ns

Note: 1. t_{SCK} is independent of t_{CLCL}. All devices can operate with minimum t_{SCK} = 1 µs, although some devices may operate at faster speeds.

3. Memory Organization

Atmel AT89LP microcontrollers offer from 2K bytes to 64K bytes of In-System programming nonvolatile Flash code memory. In addition, some devices offer nonvolatile Flash data memory. The AT89LP devices also contain a read-only Atmel Signature Array for the device ID, a User Signature Array for user configuration information, a User Fuse Row for system configuration fuses, and a memory Lock Bits for software security. Each memory type resides in its own address space and is accessed by commands specific to that memory. The memory organization of a typical AT89LP microcontroller is shown in Figure 3-1. The memory can be accessed one page at a time. One page is the largest amount of data which can be programmed at one time. For devices that can operate up to 5V, one page typically represents one row in the memory array. For devices that can operate only up to 3V, one page represents one-half of a row in the memory array. The size and number of pages for typical code densities are listed in Table 3-1. Generally all nonvolatile memory spaces within a device share the same page size. For the specific memory structure of an AT89LP device, see that device's datasheet.

The AT89LP microcontrollers feature a Row Erase capability in addition to full Chip Erase capability. For most 5V devices the row is identical to the page, but for 3V-only devices the row usually contains two pages and both pages will be erased by a Row Erase operation. Chip Erase is still needed to unlock a device which has been previously protected. However, for an unprotected part Row Erase saves time when only a few rows need reprogramming due to minor code changes or updates. In summary, a page is the largest amount of data which can be programmed at one time, while a row is the smallest amount of data which can be erased at one time.

User configuration fuses are mapped as if they were a row in the memory, with each byte address representing one fuse. From a programming standpoint, fuses are treated the same as normal code bytes except they are not affected by Chip Erase. Fuses can be enabled at any time by writing 00h to the appropriate locations in the fuse row. However, to disable a fuse, i.e. set it to FFh, the **entire** fuse row must be erased and then reprogrammed. The programmer should read the state of all the fuses into a temporary location, modify those fuses which need to be disabled, then issue a Fuse Write with Auto-Erase command using the temporary data.

Figure 3-1. Memory Organization

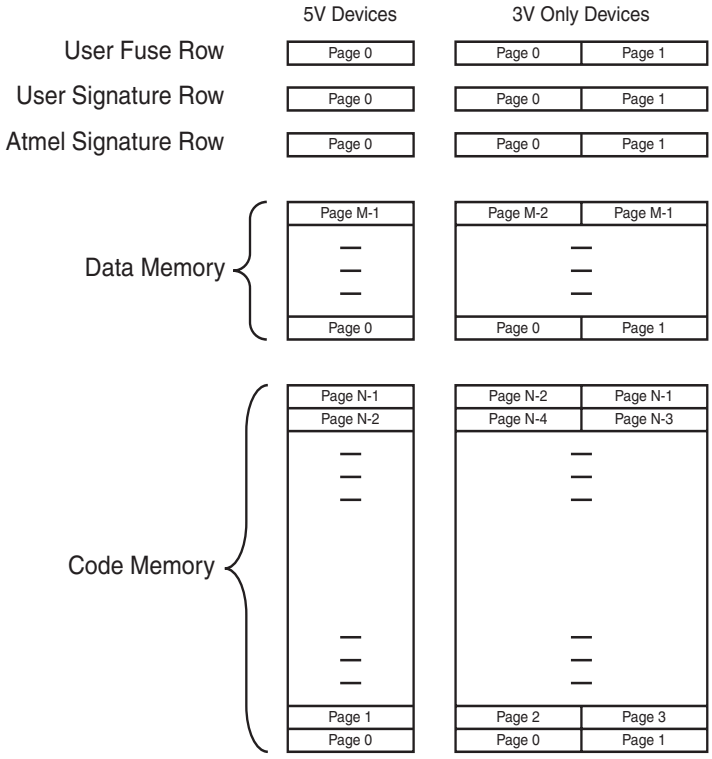


Table 3-1. Typical Memory Page Sizes

Density (Kbytes)	Page Size (Bytes)	# Pages	Address Range
2	32	64	0000H - 07FFH
4	32	128	0000H - 0FFFH
8	64	128	0000H - 1FFFH
12	64	192	0000H - 2FFFH
16	64	256	0000H - 3FFFH
32	64	512	0000H - 7FFFH
64	64	1024	0000H - FFFFH

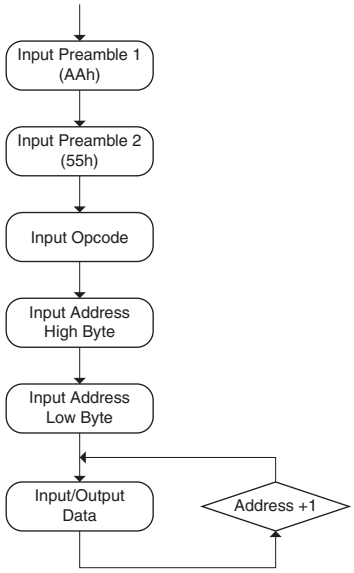
4. Programming Protocol

After Reset goes active on the target AT89LP microcontroller, the chip is ready to enter programming mode. The internal Serial Peripheral Interface is activated, and is ready to accept instructions from the programmer. Commands are entered one byte at a time over the SPI pins. Refer to "The Programming Interface" on page 1 for more details on the SPI.

4.1 Command Format

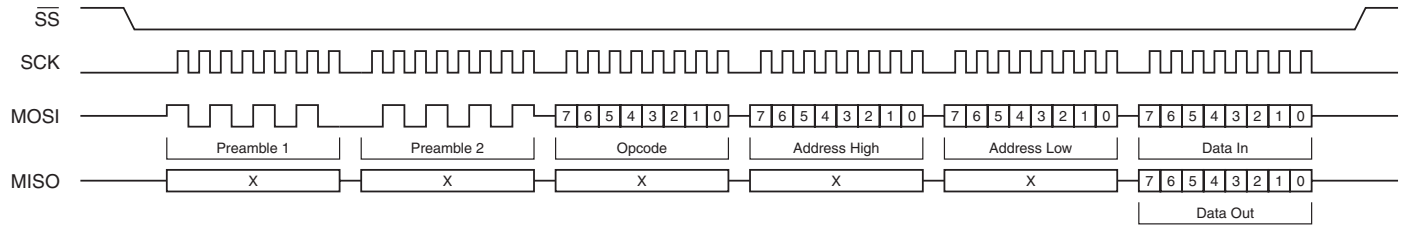
Programming commands consist of an opcode byte, two address bytes, and zero or more data bytes. In addition, all command packets must start with a two byte preamble of AAH and 55H. The preamble increases the noise immunity of the programming interface by making it more difficult to issue unintentional commands. Figure 4-1 shows a simplified flow chart of a command sequence.

Figure 4-1. Command Sequence Flow Chart



A sample command packet is shown in Figure 4-2. The $\overline{SS}$ pin defines the packet frame. The $\overline{SS}$ must be brought low before the first byte in a command is sent and brought back high after the final byte in the command has been sent. The command is not complete until $\overline{SS}$ returns high. Command bytes are issued serially on MOSI. Data output bytes are received serially on MISO. Packets of variable length are supported by returning $\overline{SS}$ high when the final required byte has been transmitted. In some cases command bytes have a don't care value. Don't care bytes in the middle of a packet must be transmitted. Don't care bytes at the end of a packet may be ignored.

Figure 4-2. ISP Command Packet



Page-oriented instructions always include a full 16-bit address. The higher order bits select the page and the lower order bits select the byte within that page. The number of bits allocated for page and byte addresses will vary depending on the memory size. See Table 3-1 for more information. The page to be accessed is always fixed by the page address as transmitted. The byte address specifies the starting address for the first data byte. After each data byte has been transmitted, the byte address is incremented to point to the next data byte. This allows a page command to linearly sweep the bytes within a page. If the byte address is incremented past the last byte in the page, the byte address will roll over to the first byte in the same page. While loading bytes into the page buffer, overwriting previously loaded bytes will result in data corruption.

4.2 Status Register

The current state of the memory may be accessed by reading the status register. The status register is shown in Table 4-1. The status register can be used to monitor the completion of a programming command by polling the state of the $\overline{\text{BUSY}}$ bit.

Table 4-1. Status Register (Read Only)

	–	–	–	–	$\overline{\text{LOAD}}$	SUCCESS	$\overline{\text{WRTINH}}$	$\overline{\text{BUSY}}$
Bit	7	6	5	4	3	2	1	0

Symbol	Function
$\overline{\text{LOAD}}$	Load flag. Cleared low by the load page buffer command and set high by the next memory write. This flag signals that the page buffer was previously loaded with data by the load page buffer command.
SUCCESS	Success flag. Cleared low at the start of a programming cycle and will only be set high if the programming cycle completes without interruption from the brownout detector.
$\overline{\text{WRTINH}}$	Write Inhibit flag. Cleared low by the brownout detector (BOD) whenever programming is inhibited due to V_{CC} falling below the minimum required programming voltage. If a BOD episode occurs during programming, the SUCCESS flag will remain low after the cycle is complete. $\overline{\text{WRTINH}}$ low also forces $\overline{\text{BUSY}}$ low.
$\overline{\text{BUSY}}$	Busy flag. Cleared low whenever the memory is busy programming or if write is currently inhibited.

4.3 $\overline{\text{DATA}}$ Polling

The AT89LP microcontrollers implement $\overline{\text{DATA}}$ polling to indicate the end of a programming cycle. While the device is busy, any attempted read of the last byte written will return the data byte with the MSB complemented. Once the programming cycle has completed, the true value will be accessible. During Erase, the data is assumed to be FFH and $\overline{\text{DATA}}$ polling will return 7FH. When writing multiple bytes in a page, the $\overline{\text{DATA}}$ value will be the last data byte loaded before programming begins, not the written byte with the highest physical address within the page.

4.4 Programming Command Summary

Command	Opcode	Addr High	Addr Low	Data 0	Data n
Programming Enable[1]	1010 1100	0101 0011			
Chip Erase	1000 1010				
Read Status	0110 0000	xxxx xxxx	xxxx xxxx	Status Out	
Load Page Buffer[2]	0101 0001	aaaa aaaa	aaaa aaaa	Data In 0 ... Data In n	
Write Code Page[2]	0101 0000	aaaa aaaa	aaaa aaaa	Data In 0 ... Data In n	
Write Code Page with Auto-Erase[2][3]	0111 0000	aaaa aaaa	aaaa aaaa	Data In 0 ... Data In n	
Read Code Page[2]	0011 0000	aaaa aaaa	aaaa aaaa	Data Out 0 ... Data Out n	
Write Data Page[2]	1101 0000	aaaa aaaa	aaaa aaaa	Data In 0 ... Data In n	
Write Data Page with Auto-Erase[2][3]	1101 0010	aaaa aaaa	aaaa aaaa	Data In 0 ... Data In n	
Read Data Page[2]	1011 0000	aaaa aaaa	aaaa aaaa	Data Out 0 ... Data Out n	
Write User Fuses[4][5]	1110 0001	aaaa aaaa	aaaa aaaa	Data In 0 ... Data In n	
Write User Fuses with Auto-Erase[3][4][5]	1111 0001	aaaa aaaa	aaaa aaaa	Data In 0 ... Data In n	
Read User Fuses[4]	0110 0001	aaaa aaaa	aaaa aaaa	Data Out 0 ... Data Out n	
Write Lock Bits[4]	1110 0100	aaaa aaaa	aaaa aaaa	Data In 0 ... Data In n	
Read Lock Bits[4]	0110 0100	aaaa aaaa	aaaa aaaa	Data Out 0 ... Data Out n	
Write User Signature Page[2]	0101 0010	aaaa aaaa	aaaa aaaa	Data In 0 ... Data In n	
Write User Signature Page with Auto-Erase[2][3]	0111 0010	aaaa aaaa	aaaa aaaa	Data In 0 ... Data In n	
Read User Signature Page[2]	0011 0010	aaaa aaaa	aaaa aaaa	Data Out 0 ... Data Out n	
Read Atmel Signature Page[2]	0011 1000	aaaa aaaa	aaaa aaaa	Data Out 0 ... Data Out n	

Notes:
1. Programming Enable must be the **first** command issued after entering into programming mode.
2. Any number of data bytes from 0 to the page size may be read or loaded.
3. Auto-Erase erases an entire memory row. For devices with code memory ≥ 32K bytes, both pages in a row will be erased but only one may be written. Use Write Page to program the other page in the row.
4. Refer to a particular device's datasheet for specific fuse or lock bit assignments and addresses.
5. User Fuses can only be enabled with the Write User Fuse command. If a fuse needs to be disabled, use Fuse Write with Auto-Erase to disable all the fuses and then re-enable only the required fuses.

4.4.1 Programming Enable

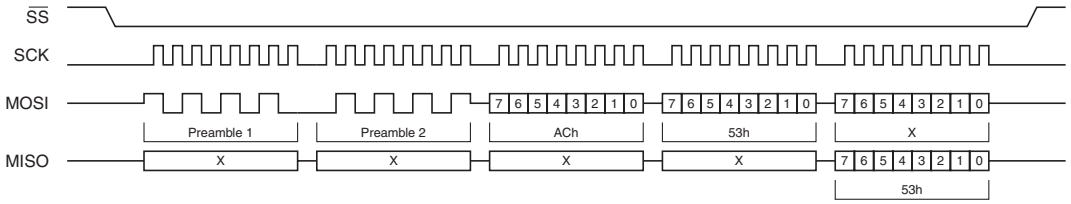
Function:

- Enables the programming interface to receive commands and configures MISO as an output.
- Program Enable must be the first command issued in any programming session. During In-System programmin, a session is active while $\overline{RST}$ remains at low and is terminated by $\overline{RST}$ high or power off.

Usage:

1. Bring $\overline{SS}$ low.
2. Send preamble byte AAh.
3. Send preamble byte 55h.
4. Send opcode byte ACh.
5. Send high address byte 53h.
6. Send dummy low address byte. 53h should be returned on MISO if ISP is enabled.
7. Bring $\overline{SS}$ high.

Figure 4-3. Program Enable Sequence



4.4.2 Chip Erase

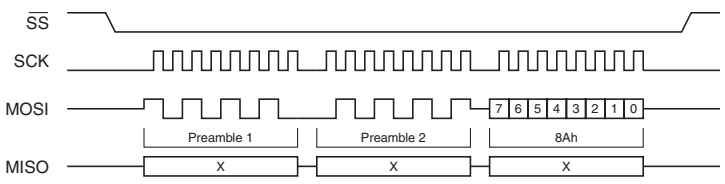
Function:

- Erases (programs FFh to) the entire code and data memory arrays.
- Erases the User Signature Row if User Row Programming Fuse is enabled.
- Lock bits are programmed to “unlock” state.
- Chip Erase does **not** affect the User Fuse Row.

Usage:

1. Drive $\overline{SS}$ low.
2. Send preamble byte AAh.
3. Send preamble byte 55h.
4. Send opcode byte 8Ah.
5. Drive $\overline{SS}$ high.
6. Poll data or status.

Figure 4-4. Chip Erase Sequence



4.4.3 Load Page Buffer

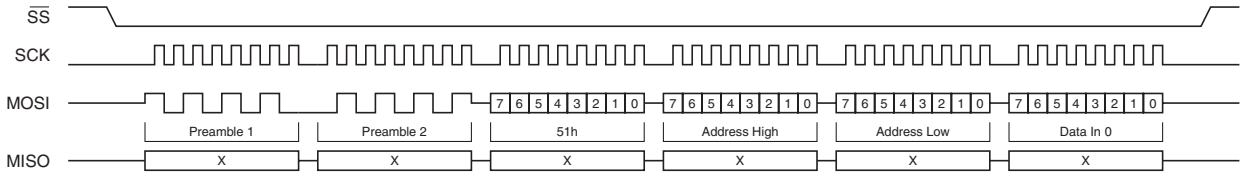
Function:

- Loads one page of data into the temporary page buffer but does not start programming.
- Use for interruptible loads or loading non-contiguous bytes to a page.
- The byte address (offset in page) is initialized from the low order bits of the address. The internal byte address is incremented by one after each successive data byte. The address will wrap around to the 1st byte of the page when incremented past the page end, however, previously loaded bytes should not be re-loaded.
- The Load Page Buffer command needs to be followed by a write command as the internal buffer is not cleared until either the next write has completed or the programming session ends.
- Clears Bit 3 ($\overline{\text{LOAD}}$) of the status byte to signal that the buffer contains data.
- Load Page Buffer can be used before any write or write with auto-erase command.

Usage:

1. Drive $\overline{\text{SS}}$ low.
2. Send preamble byte AAh.
3. Send preamble byte 55h.
4. Send opcode byte 51h.
5. Send high address byte.
6. Send low address byte.
7. Send 1st data byte. Repeat for additional bytes.
8. Drive $\overline{\text{SS}}$ high.

Figure 4-5. Load Page Buffer Sequence (Single Data Byte)



4.4.4 Write Code Page

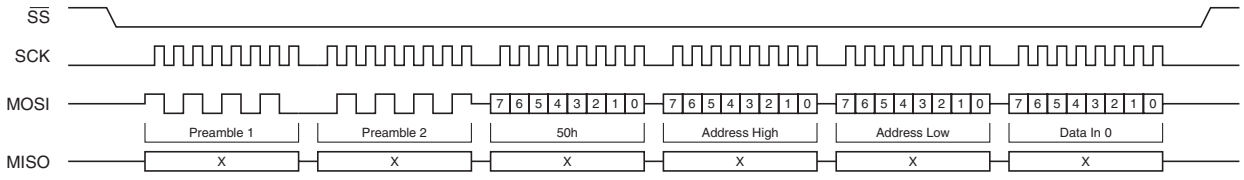
Function:

- Programs one page of data into the Code Memory array.
- Page address determined by high order bits of loaded address.
- The byte address (offset in page) is initialized from the low order bits of the address. The internal byte address is incremented by one after each successive data byte. The address will wrap around to the 1st byte of the page when incremented past the page end, however, previously loaded bytes should not be re-loaded. It is not possible to skip bytes while loading data during write. To load non-contiguous bytes in a page, use the Load Page Buffer command.
- See Figure 4-10 for an example of a multiple data byte page write command.

Usage:

1. Drive $\overline{SS}$ low.
2. Send preamble byte AAh.
3. Send preamble byte 55h.
4. Send opcode byte 50h.
5. Send high address byte.
6. Send low address byte.
7. To write only previously loaded data skip to (8), otherwise send 1st data byte. Repeat for additional bytes.
8. Drive $\overline{SS}$ high.
9. Poll data or status.

Figure 4-6. Write Code Page Sequence (Single Data Byte)



4.4.5 Write Code Page with Auto-Erase

Function:

- Erase one row in the Code Memory array and programs one page of data. For devices with code memory ≥ 32K bytes, both pages in the row will be erased but only one may be programmed. Use Write Code Page to program the other page.
- Row erase may be performed by not loading any data bytes.
- Page address determined by high order bits of loaded address.
- The byte address (offset in page) is initialized from the low order bits of the address. The internal byte address is incremented by one after each successive data byte. The address will wrap around to the 1st byte of the page when incremented past the page end, however, previously loaded bytes should not be re-loaded. It is not possible to skip bytes while loading data during write. To load non-contiguous bytes in a page, use the Load Page Buffer command.
- See Figure 4-10 for an example of a multiple data byte page write with auto-erase command.

Usage:

1. Drive $\overline{SS}$ low.
2. Send preamble byte AAh.
3. Send preamble byte 55h.
4. Send opcode byte 70h.
5. Send high address byte.
6. Send low address byte.
7. To perform row erase or to write only previously loaded data skip to (8), otherwise send 1st data byte. Repeat for additional bytes.
8. Drive $\overline{SS}$ high.
9. Poll data or status.

Figure 4-7. Write Code Page with Auto-Erase Sequence (Single Data Byte)

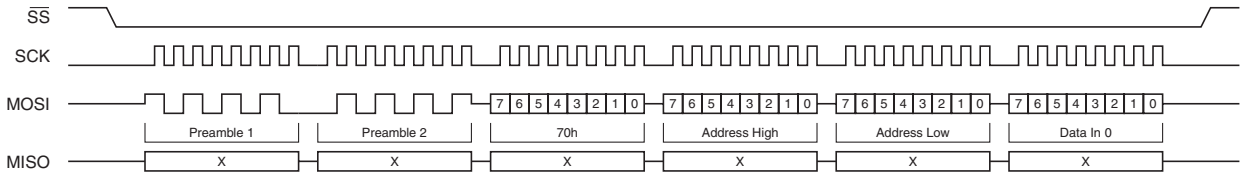
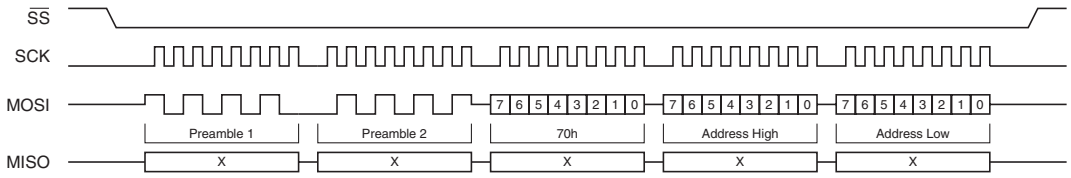


Figure 4-8. Code Row Erase Sequence



4.4.6 Read Code Page

Function:

- Read one page of data from the Code Memory array.
- Page address determined by high order bits of loaded address.
- The byte address (offset in page) is initialized from the low order bits of the address. The internal byte address is incremented by one after each successive data byte. The address will wrap around to the 1st byte of the page when incremented past the page end.
- See Figure 4-10 for an example of a multiple data byte page read command.

Usage:

1. Drive $\overline{SS}$ low.
2. Send preamble byte AAh.
3. Send preamble byte 55h.
4. Send opcode byte 30h.
5. Send high address byte.
6. Send low address byte.
7. Receive 1st data byte. Repeat for additional bytes.
8. Drive $\overline{SS}$ high.

Figure 4-9. Read Code Page Sequence (Single Data Byte)

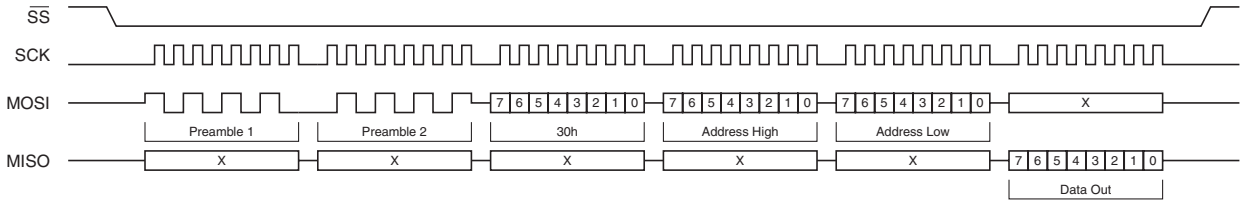
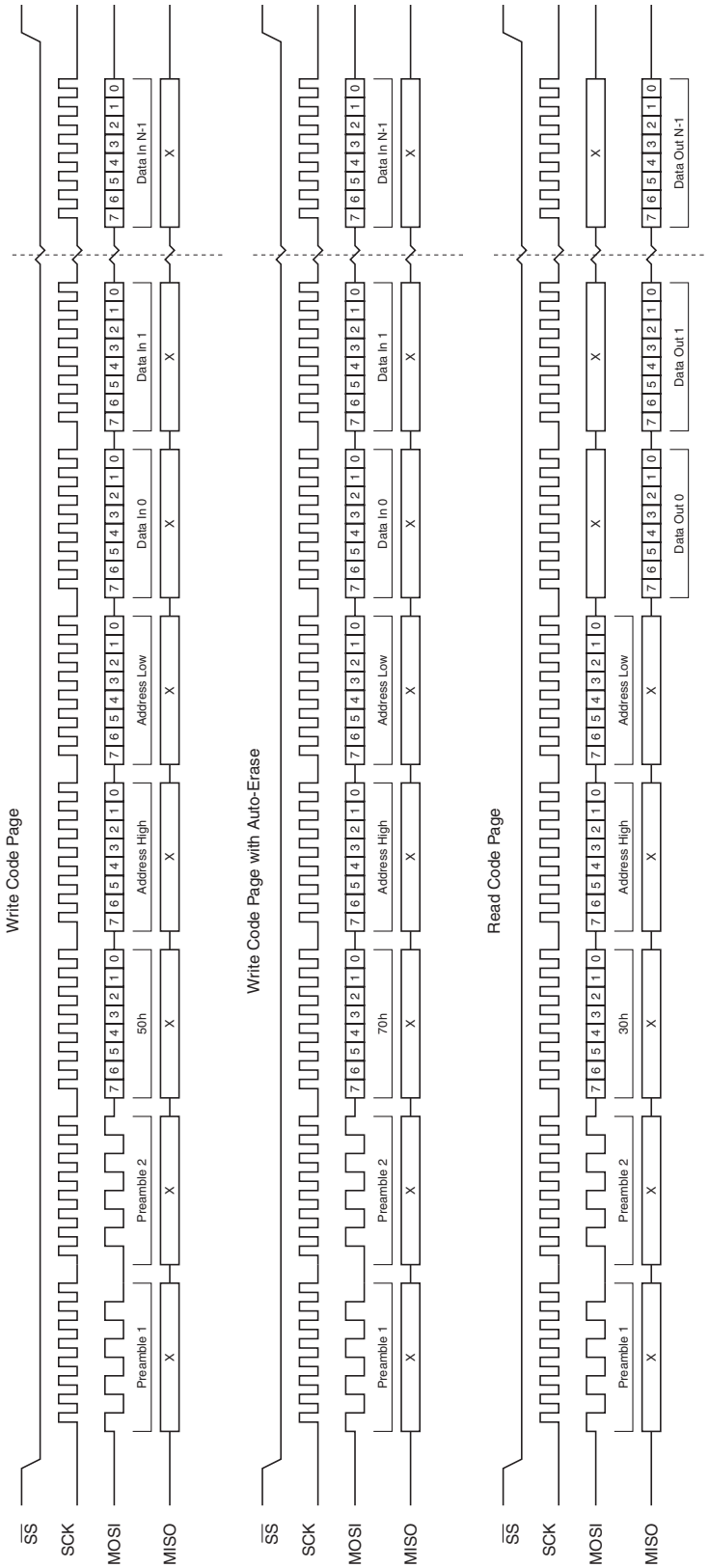


Figure 4-10. Code Page Write, Write with Auto-Erase, and Read Commands with N Code Bytes



4.4.7 Write Data Page

Function:

- Programs one page of data into the Data Memory array.
- Page address determined by high order bits of loaded address.
- The byte address (offset in page) is initialized from the low order bits of the address. The internal byte address is incremented by one after each successive data byte. The address will wrap around to the 1st byte of the page when incremented past the page end, however, previously loaded bytes should not be re-loaded. It is not possible to skip bytes while loading data during write. To load non-contiguous bytes in a page, use the Load Page Buffer command.

Usage:

- Identical to "Write Code Page" but with opcode D0h.

4.4.8 Write Data Page with Auto-Erase

Function:

- Erase one row in the Data Memory array and programs one page of data. For devices with code memory ≥ 32K bytes, both pages in the row will be erased but only one may be programmed. Use Write Data Page to program the other page.
- Row erase may be performed by not loading any data bytes.
- Page address determined by high order bits of loaded address.
- The byte address (offset in page) is initialized from the low order bits of the address. The internal byte address is incremented by one after each successive data byte. The address will wrap around to the 1st byte of the page when incremented past the page end, however, previously loaded bytes should not be re-loaded. It is not possible to skip bytes while loading data during write. To load non-contiguous bytes in a page, use the Load Page Buffer command.

Usage:

- Identical to "Write Code Page with Auto-Erase" but with opcode D2h.

4.4.9 Read Data Page

Function:

- Read one page of data from the Data Memory array.
- Page address determined by high order bits of loaded address.
- The byte address (offset in page) is initialized from the low order bits of the address. The internal byte address is incremented by one after each successive data byte. The address will wrap around to the 1st byte of the page when incremented past the page end.

Usage:

- Identical to "Read Code Page" but with opcode B0h

4.4.10 Write User Signature Page

Function:

- Programs one page of data into the User Signature Row.
- The User Row Programming Fuse must be enabled prior to executing this command.
- Page address determined by high order bits of loaded address.
- The byte address (offset in page) is initialized from the low order bits of the address. The internal byte address is incremented by one after each successive data byte. The address will wrap around to the 1st byte of the page when incremented past the page end, however, previously loaded bytes should not be re-loaded. It is not possible to skip bytes while loading data during write. To load non-contiguous bytes in a page, use the Load Page Buffer command.

Usage:

- Identical to "Write Code Page" but with opcode 52h.

4.4.11 Write User Signature Page with Auto-Erase

Function:

- Erases the User Signature Row and programs one page of data. For devices with code memory ≥ 32K bytes both pages in the row will be erased but only one may be programmed. Use Write User Signature Page to program the other page.
- Row erase may be performed by not loading any data bytes.
- Page address determined by high order bits of loaded address.
- The byte address (offset in page) is initialized from the low order bits of the address. The internal byte address is incremented by one after each successive data byte. The address will wrap around to the 1st byte of the page when incremented past the page end, however, previously loaded bytes should not be re-loaded. It is not possible to skip bytes while loading data during write. To load non-contiguous bytes in a page, use the Load Page Buffer command.

Usage:

- Identical to "Write Code Page with Auto-Erase" but with opcode 72h.

4.4.12 Read User Signature Page

Function:

- Read one page of data from the User Signature Row.
- Page address determined by high order bits of loaded address.
- The byte address (offset in page) is initialized from the low order bits of the address. The internal byte address is incremented by one after each successive data byte. The address will wrap around to the 1st byte of the page when incremented past the page end.

Usage:

- Identical to "Read Code Page" but with opcode 32h

4.4.13 Read Atmel Signature Page

Function:

- Read one page of data from the Atmel Signature Row.
- Page address determined by high order bits of loaded address.
- The byte address (offset in page) is initialized from the low order bits of the address. The internal byte address is incremented by one after each successive data byte. The address will wrap around to the 1st byte of the page when incremented past the page end.
- Atmel Device IDs are stored at locations 00h, 01h, and 02h.

Usage:

- Identical to "Read Code Page" but with opcode 38h

4.4.14 Write Lock Bits

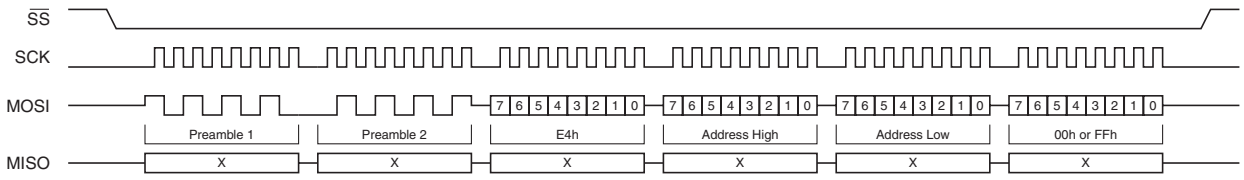
Function:

- Program (lock) memory Lock Bits.
- Lock Bits can only be erased (unlocked) by Chip Erase.
- Each Lock Bit is accessed at a separate byte address. The lock bit address is initialized from the low order bits of the address. The internal bit address is incremented by one after each successive data byte. To program (lock) a bit, write 00h to its location. To leave a lock bit unchanged, write FFh to its location. Refer to a particular device's datasheet for specific lock bit assignments and addresses.

Usage:

1. Drive $\overline{SS}$ low.
2. Send preamble byte AAh.
3. Send preamble byte 55h.
4. Send opcode byte E4h.
5. Send high address byte.
6. Send low address byte.
7. Send 1st data byte, with 00h for lock or FFh for unchanged. Repeat for additional bytes.
8. Drive $\overline{SS}$ high.
9. Poll data or status.

Figure 4-11. Write Lock Bits Sequence (Single Lock Bit)



4.4.15 Read Lock Bits

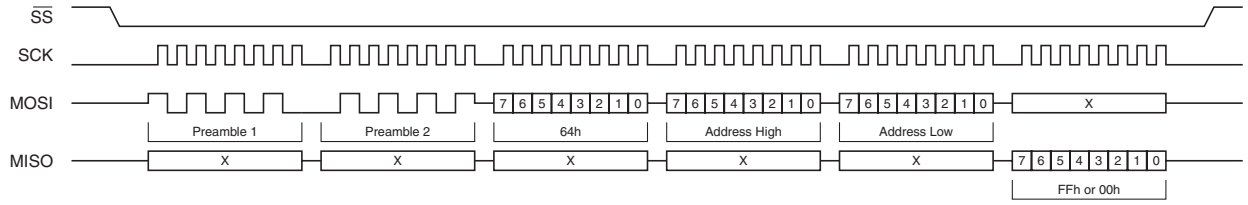
Function:

- Read status of memory Lock Bits.
- Each Lock Bit is accessed at a separate byte address. The lock bit address is initialized from the low order bits of the address. The internal bit address is incremented by one after each successive data byte. A lock bit will read as FFh for unlocked or 00h for locked. Refer to a particular device's datasheet for specific lock bit assignments and addresses.

Usage:

1. Drive $\overline{SS}$ low.
2. Send preamble byte AAh.
3. Send preamble byte 55h.
4. Send opcode byte 64h.
5. Send high address byte.
6. Send low address byte.
7. Receive 1st data byte. Repeat for additional bytes.
8. Drive $\overline{SS}$ high.

Figure 4-12. Read Lock Bits Sequence (Single Lock Bit)



4.4.16 Write User Fuses

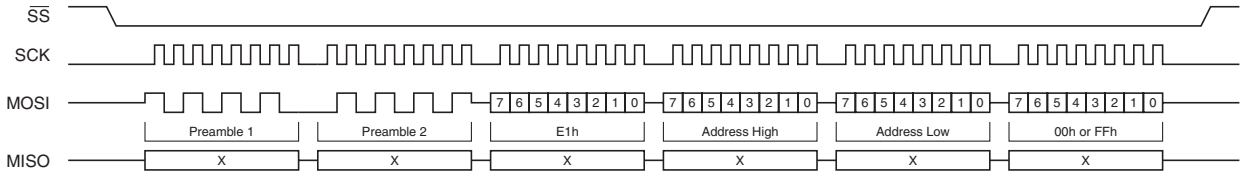
Function:

- Enable User Configuration Fuses.
- User Fuses can only be disabled by Write User Fuses with Auto-Erase.
- The Write User Fuse command is similar to Write Code Page where each fuse is accessed as if it were a full byte within a fuse page. The fuse address is initialized from the low order bits of the address. The internal fuse address is incremented by one after each successive data byte. To program (enable) a fuse, write 00h to its location. To leave a fuse unchanged, write FFh to its location. Refer to a particular device's datasheet for specific user fuse assignments and addresses.

Usage:

1. Drive $\overline{SS}$ low.
2. Send preamble byte AAh.
3. Send preamble byte 55h.
4. Send opcode byte E1h.
5. Send high address byte.
6. Send low address byte.
7. Send 1st data byte, with 00h for enable or FFh for unchanged. Repeat for additional bytes.
8. Drive $\overline{SS}$ high.
9. Poll data or status.

Figure 4-13. Write User Fuses Sequence (Single Fuse Bit)



4.4.17 Write User Fuses with Auto-Erase

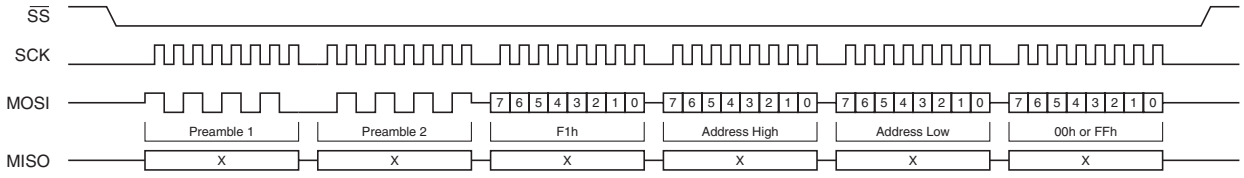
Function:

- Disables all User Fuses then enables selected User Configuration Fuses.
- The Write User Fuse with Auto-Erase command is similar to Write Code Page with Auto-Erase where each fuse is accessed as if it were a full byte within a fuse page. The fuse address is initialized from the low order bits of the address. The internal fuse address is incremented by one after each successive data byte. To program (enable) a fuse, write 00h to its location. To leave a fuse disabled, write FFh to its location. Refer to a particular device's datasheet for specific user fuse assignments and addresses.
- The Write User Fuse with Auto-Erase command will erase the entire fuse row. If the programmer does not want to modify certain fuses, the programmer should first read the state of the fuses and then write back the same value to the fuse when issuing the write with auto-erase command.

Usage:

1. Drive $\overline{SS}$ low.
2. Send preamble byte AAh.
3. Send preamble byte 55h.
4. Send opcode byte F1h.
5. Send high address byte.
6. Send low address byte.
7. Send 1st data byte, with 00h for enable or FFh for disable. Repeat for additional bytes.
8. Drive $\overline{SS}$ high.
9. Poll data or status.

Figure 4-14. Write User Fuses with Auto-Erase Sequence (Single Fuse Bit)



4.4.18 Read User Fuses

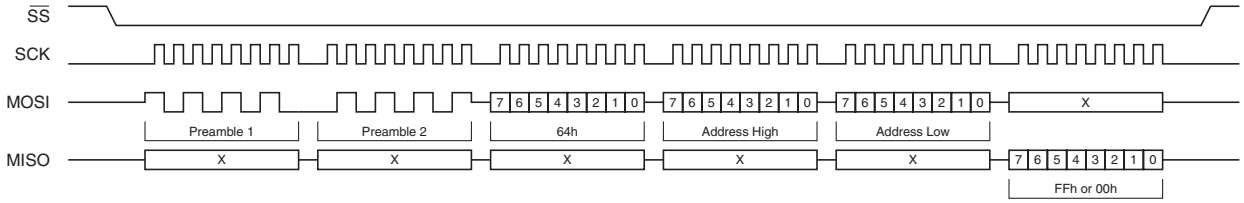
Function:

- Read status of User Configuration Fuses.
- The Read User Fuse command is similar to Read Code Page where each User Fuse is accessed as if it were a full byte within a fuse page. The fuse address is initialized from the low order bits of the address. The internal fuse address is incremented by one after each successive data byte. A fuse will read as FFh for disabled or 00h for enabled. Refer to a particular device's datasheet for specific fuse assignments and addresses.

Usage:

1. Drive $\overline{SS}$ low.
2. Send preamble byte AAh.
3. Send preamble byte 55h.
4. Send opcode byte 61h.
5. Send high address byte.
6. Send low address byte.
7. Receive 1st data byte. Repeat for additional bytes.
8. Drive $\overline{SS}$ high.

Figure 4-15. Read User Fuses Sequence (Single Fuse Bit)



4.4.19 Read Status

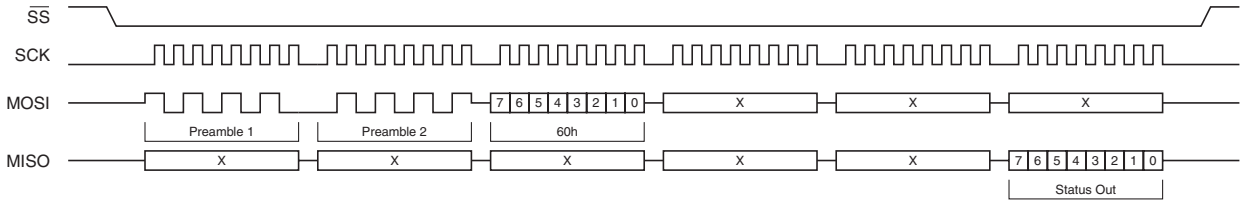
Function:

- Read memory status byte.

Usage:

1. Drive $\overline{SS}$ low.
2. Send preamble byte AAh.
3. Send preamble byte 55h.
4. Send opcode byte 60h.
5. Send dummy high address byte.
6. Send dummy low address byte.
7. Receive 1st data byte. Repeat to continue polling.
8. Drive $\overline{SS}$ high.

Figure 4-16. Read Status Sequence



5. Programming the AT89LP2052/LP4052

The AT89LP2052 and AT89LP4052 microcontrollers are slightly different from the rest of the AT89LP family. These devices are pin compatible with the AT89C2051 and AT89C4051 microcontrollers and maintain the classic 8051 active-HIGH reset. After a cold power-up, RST must be kept low (inactive) for at least t_{PWRUP} before being driven high (active) in order to avoid latch-up. The AT89LP2052 and AT89LP4052 In-System programming protocol differs in that only a single preamble byte (AAh) is used instead of two and they do not support any Auto-Erase or Row Erase commands. The User Fuse and Lock Bit organization is also different with all the lock bits stored as bits in a single byte and all the user fuses stored as bits in another single byte. In addition the AT89LP2052 and AT89LP4052 support high voltage parallel programming. ISP can be enabled or disabled by setting or clearing the ISP Enable fuse during parallel programming. For details on programming the AT89LP2052 or AT89LP4052 in either parallel or serial mode, see the associated datasheet for those devices.

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe

Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia

Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan

9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Atmel Operations

Memory

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards

Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive

Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High-Speed Converters/RF Datacom

Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature