APPLICATION NOTE

ABSTRACT
This application note describes the three methods that can be used to program the Flash code memory of the 89C51Rx+/Rx2/66x families of microcontrollers. It discusses in detail the operation of the In-System Programming (ISP) capability which allows these microcontrollers to be programmed while mounted in the end product. These microcontrollers also have an In-Application Programming (IAP) capability which allows them to be programmed under firmware control of the embedded application. This capability is also described.

AN461

In-circuit and In-application programming of the 89C51Rx+/Rx2/66x microcontrollers

Supersedes data of 2002 Jun 24

2003 Mar 11

Philips Semiconductors

PHILIPS

In-circuit and In-application programming of the 89C51Rx+/Rx2/66x microcontrollers

AN461

INTRODUCTION

This document gives a brief list of features for the 89C51Rx+/Rx2/66x family of microcontrollers with Flash memory, and the ways that the Flash memory can be programmed.

MCU FEATURES

- 80C51 CPU
- 8K, 16K, 32K, 64 kbyte Flash EPROM
- Flash EPROM is sectored to allow the user to erase and reprogram sectors
- 1 kbyte Masked BOOTROM for In-System Programming of the Flash EPROM
- User callable BOOTROM subroutines for Flash erase and programming
- Can automatically run user program or BOOTROM program at power-up
- Three security bits
- Fully static operation: 0 to 33 MHz @ 12 clocks/instruction; 0 to 20 MHz @ 6 clocks/instruction
- 100% code and pin compatibility with 80C52
- Packages: 44-pin PLCC, 44-pin QFP, 40-pin DIP

The Flash Program Memory can be programmed using three different methods:

- The traditional parallel programming method (not described in this Application Note)
- A new In-System Programming method (ISP) through the serial port
- In Application programming method (IAP) under control of a running microcontroller application program

Programming functions support the following functions:

- erase and blank check Flash memory
- program and read / verify Flash memory
- program and verify security bits, status byte and boot vector
- read signature bytes
- full-chip erase

Memory Spaces

Code memory on Philips Flash microcontrollers is organized into sectors of 4 kbyte, 8 kbyte, or 16 kbyte, as indicated below. Different amounts of memory are present depending on the specific device as shown in Table 1 and Table 2 below.

Table 1. Memory Block of Philips ISP Flash Microcontrollers

Memory Block	P89C51RB2H/RB+/660	P89C51RC2H/RC+/662	P89C51RD2H/RD+/664/68
8 kbyte (0–1FFF)	X	X	X
8 kbyte (2000–3FFF)	X	X	X
16 kbyte (4000 – 7FFF)		X	X
16 kbyte (8000–BFFF)			X
16 kbyte (C000–FFFF)			X
Total Flash Memory	16 kbyte	32 kbyte	64 kbyte

Table 2. Memory Block of Philips P89C51Rx2 ISP Flash Microcontrollers

Memory Block	P89C51RA2	P89C51RB2	P89C51RC2	P89C51RA2
4 kbyte, (0–FFF)	X	X	X	X
4 kbyte, (1000– 1FFF)	X	X	X	X
4 kbyte, (2000–2FFF)		X	X	X
4 kbyte, (3000–3FFF)		X	X	X
4 kbyte, (4000– 4FFF)			X	X
4 kbyte, (5000– 5FFF)			X	X
4 kbyte, (6000– 6FFF)			X	X
4 kbyte, (7000– 7FFF)			X	X
4 kbyte, (8000– 8FFF)				X
4 kbyte, (9000– 9FFF)				X
4 kbyte, (A000– AFFF)				X
4 kbyte, (B000– BFFF)				X
4 kbyte, (C000– CFFF)				X
4 kbyte, (D000– DFFF)				X
4 kbyte, (E000– EFFF)				X
4 kbyte, (F000– FFFF)				X
Total Flash Memory	8 kbyte	16 kbyte	32 kbyte	64 kbyte

General Overview of In-System Programming (ISP)

In-System Programming (ISP) is a process whereby a blank device mounted to a circuit board can be programmed with the end-user code without the need to remove the device from the circuit board. Also, a previously programmed device can be erased and reprogrammed without removal from the circuit board.

In order to perform ISP operations the microcontroller is powered up in a special “ISP mode”. ISP mode allows the microcontroller to communicate with an external host device through the serial port, such as a PC or terminal. The microcontroller receives commands and data from the host, erases and reprograms code memory, etc. Once the ISP operations have been completed the device is reconfigured so that it will operate normally the next time it is either reset or power removed and reapplied.

All of the Philips microcontrollers shown in Table 1 and Table 2 have a 1 kbyte factory-masked ROM located in the upper 1 kbyte of code memory space from FC00 to FFFF. This 1 kbyte ROM is in addition to the memory blocks shown in Table 1 and Table 2. This ROM is referred to as the “Bootrom”. This Bootrom contains a set of instructions which allows the microcontroller to perform a number of Flash programming and erasing functions. The Bootrom also provides communications through the serial port. The use of the Bootrom is key to the concepts of both ISP and In-Application Programming (IAP). The contents of the bootrom are provided by Philips and masked into every device.

When the device is reset or power applied, and the EA/ pin is high or at the V_{PP} voltage, the microcontroller will start executing instructions from either the user code memory space at address 0000h (“normal mode”) or will execute instructions from the Bootrom (ISP mode). Selection of these modes will be described later.

General Overview of In-Application Programming (IAP)

Some applications may have a need to be able to erase and program code memory under the control fo the application. For example, an application may have a need to store calibration information or perhaps need to be able to download new code portions. This ability to erase and program code memory in the end-user application is “In-Application Programming” (IAP).

The Bootrom routines which perform functions on the Flash memory during ISP mode such as programming, erasing, and reading, are also available to end-user programs. Thus it is possible for an end-user application to perform operations on the Flash memory. A common entry point (FFF0h) to these routines has been provided to simplify interfacing to the end-users application. Functions are performed by setting up specific registers as required by a specific operation and performing a call to the common entry point. Like any other subroutine call, after completion of the function, control will return to the end-user’s code.

The Bootrom is shadowed with the user code memory in the address range from FC00h to FFFFh. This shadowing is controlled by the ENBOOT bit (AUXR1.5). When set, accesses to internal code memory in this address range will be from the boot ROM. When cleared, accesses will be from the user’s code memory. It will be NECESSARY for the end-user’s code to set the ENBOOT bit prior to calling the common entry point for IAP operations, even for devices with 16 kbyte, 32 kbyte, and 64 kbyte of internal code memory. (ISP operation is selected by certain hardware conditions and control of the ENBOOT bit is automatic when ISP mode is activated).

Using the Watchdog Timer (WDT)

The 89C51Rx2 and 89C66x devices support the use of the WDT in IAP. The user specifies that the WDT is to be fed by setting the most significant bit of the function parameter passed in R1 prior to calling PGM_MTP. The WDT function is only supported for Block Erase when using the Quick Block Erase. The Quick Block Erase is specified by performing a Block Erase with Register R0 = 0. Requesting a WDT feed during IAP should only be performed in applications that use the WDT since the process of feeding the WDT will start the WDT if the WDT was not running.

IN-SYSTEM PROGRAMMING (ISP)

The Philips In-System Programming (ISP) facility has made in-circuit programming in an embedded application possible with a minimum of additional expense in components and circuit board area.

The ISP function uses five pins: TxD, RxD, V_{SS}, V_{CC}, and V_{PP} (see Figure 1). Only a small connector needs to be available to interface your application to an external circuit in order to use this feature. The V_{PP} supply should be decoupled and V_{PP} not allowed to exceed datasheet limits.

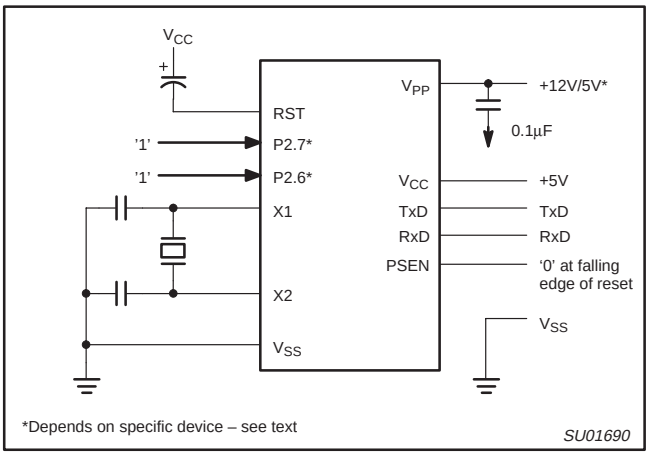

Figure 1. In-System Programming with a Minimum of Pins

In order to understand how ISP works it is necessary to first discuss two special Flash registers; the BOOT VECTOR and the STATUS BYTE. At the falling edge of reset the MCU examines the contents of the Status Byte. If the Status Byte is set to zero, power-up execution starts at location 0000H which is the normal start address of the user's application code. When the Status Byte is set to a value other than zero, the contents of the Boot Vector is used as the high byte of the execution address and the low byte is set to 00H. The factory default setting is 0FCH, corresponds to the address 0FC00H for the factory masked-ROM ISP boot loader (Boot ROM). A custom boot loader can be written with the Boot Vector set to the custom boot loader.

Note: When erasing the Status Byte or Boot Vector, both bytes are erased at the same time. It is necessary to reprogram the Boot Vector after erasing and updating the Status Byte. For proper operation of P89C51Rx2H and P89C660x devices, erasing the status byte and boot vector needs to be implemented six times in a row unless FlashMagic version 1.70 or above is used.

The bootloader may also be executed by meeting the following conditions at the falling edge of reset:

- PSEN is held low with a pull-down resistor of 2 kΩ to limit current that can exceed 30 mA
- EA\ is greater than VIH
- P2.6 and P2.7 are high or floating[1]
- ALE is high or floating

This has the same effect as a non–zero status byte. This allows an application to be built that will normally execute the end users code but can be manually forced into ISP operation.

The ISP feature allows programming of the Flash EPROM through the serial port.

The ISP programming is accomplished by serial boot loader subroutines found in the BOOTROM. These routines use Intel hex records to receive commands and data from external sources such as a host PC. (Details of these hex records are described in a later section of this application note.)

1. P2.6 and P2.7 are not required to be in any particular state in the 2nd generation P89C51Rx2 devices. These devices are distinguished by the absence of an "H" in the suffix.

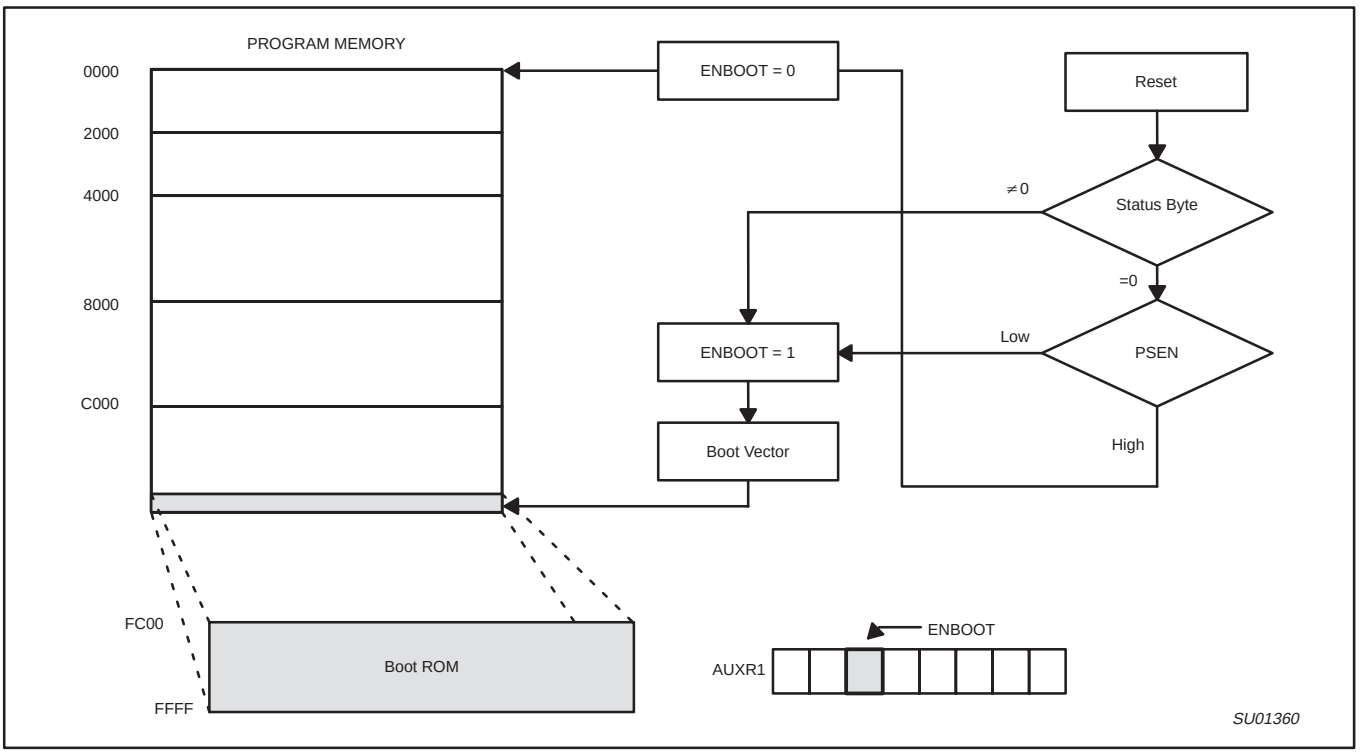

Figure 2. ISP Flow Chart

The Boot ROM code is located at memory address FC00H and can be invoked by having the Status byte non-zero and having the Boot Vector = FCH. (If the Boot Vector is a value other than FCH, an attempt to enter the ISP mode will start execution at the wrong address and may result in incorrect responses). After programming the Flash, the status byte should be programmed to zero in order to allow execution of the user's application code beginning at address 0000H.

We recommend using the following sequence for ISP programming. Refer to Table 3 for data record structure:

1. Enter the ISP mode by applying one of the methods previously described (non-zero Status Byte, $\overline{\text{PSEN}}$, etc.).
2. Send an uppercase "U" from the host to the microcontroller to autobaud.
3. Send a record from the host to the microcontroller to specify the oscillator frequency.
4. Send a record from the host to the microcontroller to erase the desired block(s).
5. Send records from the host to the microcontroller to program desired data into the device.
6. Send a record to erase both Status Byte and Boot Vector after ISP has been successfully done. There is no way to erase the Status Byte without erasing the Boot Vector.
7. Send a record to program the Boot Vector back to the original value (0FCH) if the you want to keep the default serial loader as the ISP communication channel.
8. Write 00H to the Status Byte so that the program will begin at address 0000H after reset.

Using the In-System Programming (ISP)

The ISP feature allows for a wide range of baud rates to be used in your application, independent of the oscillator frequency. It is also adaptable to a wide range of oscillator frequencies. This is accomplished by measuring the bit-time of a single bit in a received character. This information is then used to program the baud rate in terms of timer counts based on the oscillator frequency. The ISP feature requires that an initial character (an uppercase U) be sent to the 89C51Rx+/Rx2/66x to establish the baud rate. The ISP firmware provides auto-echo of received characters.

Once baud rate initialization has been performed, the ISP firmware will only accept Intel Hex-type records. Intel Hex records consist of ASCII characters used to represent hexadecimal values and are summarized below:

:NNAAAARRDD..DDCC<crlf>

In the Intel Hex record, the "NN" represents the number of data bytes in the record. The 89C51Rx+/Rx2/66x will accept up to 16 (10H) data bytes. The "AAAA" string represents the address of the first byte in the record. If there are zero bytes in the record this field is often set to 0000. The "RR" string indicates the record type. A record type of "00" is a data record. A record type of "01" indicates the end-of-file mark. In this application additional record types will be added to indicate either commands or data for the ISP facility. The maximum number of data bytes in a record is limited to 16 (decimal). ISP commands are summarized in Table 3.

As a record is received by the 89C51Rx+/Rx2/66x the information in the record is stored internally and a checksum calculation is performed. The operation indicated by the record type is not performed until the entire record has been received. Should an error occur in the checksum, the 89C51Rx+/Rx2/66x will send an "X" out

the serial port indicating a checksum error. If the checksum calculation is found to match the checksum in the record then the command will be executed. In most cases successful reception of the record will be indicated by transmitting a "." character out the serial port (displaying the contents of the internal program memory is an exception).

In the case of a Data Record (record type 00) an additional check is made. A "." character will NOT be sent unless the record checksum matched the calculated checksum and all of the bytes in the record were successfully programmed. For a data record an "X" indicates that the checksum failed to match and an "R" character indicates that one of the bytes did not properly program. It is necessary to send a type 02 record (specify oscillator frequency) to the 89C51Rx+/Rx2/66x before programming data.

The ISP facility was designed so that specific crystal frequencies were not required in order to generate baud rates or time the programming pulses. The user thus needs to provide the 89C51Rx+/Rx2/66x with information required to generate the proper timing. Record type 02 is provided for this purpose.

Software utilities to implement ISP programming with a PC are available online (see next section).

> **Note:** Some ISP and IAP functions are only available for the 2nd generation P89C51Rx2 devices. The shaded areas in the following tables denote these functions. These Rx2 devices can be distinguished from older Rx2 devices by the absence of an "H" in the part number.

Table 3. Intel-Hex Records Used by In-System Programming

RECORD TYPE	COMMAND/DATA FUNCTION

00	Program Data
	`:nnaaaa00dd....ddcc`
	Where:
	`nn = number of bytes (hex) in record`
	`aaaa = memory address of first byte in record`
	`dd....dd = data bytes`
	`cc = checksum`
	Example:
	`:10008000AF5F67F0602703E0322CFA92007780C3FD`

01	End of File (EOF), no operation
	`:xxxxxx01cc`
	Where:
	`xxxxxx = required field, but value is a "don't care"`
	`cc = checksum`
	Example:
	`:00000001FF`

02	Specify Oscillator Frequency
	`:01xxxx02ddcc`
	Where:
	`xxxx = required field, but value is a "don't care"`
	`dd = integer oscillator frequency rounded down to nearest MHz`
	`cc = checksum`
	Example:
	`:0100000210ED (dd = 10h = 16, used for 16.0–16.9 MHz)`

RECORD TYPE	COMMAND/DATA FUNCTION

03	Miscellaneous Write Functions
	`:nnxxxx03ffssddcc`
	Where:
	`nn = number of bytes (hex) in record`
	`xxxx = required field, but value is a "don't care"`
	`03 = Write Function`
	`ff = subfunction code`
	`ss = selection code`
	`dd = data input (as needed)`
	`cc = checksum`
	
	Subfunction Code = 01 (Erase Blocks)
	`ff = 01`
	`ss = block code as shown below:`
	`block 0, 0k to 8k, 00H`
	`block 1, 8k to 16k, 20H`
	`block 2, 16k to 32k, 40H`
	Example:
	`:020000030120DA erase block 1`
	
	Subfunction Code = 04 (Erase Boot Vector and Status Byte)[1]
	`ff = 04`
	`ss = don't care`
	Example:
	`:020000030400F7 erase boot vector and status byte`
	
	Subfunction Code = 05 (Program Security Bits)
	`ff = 05`
	`ss = 00 program security bit 1 (inhibit writing to Flash)`
	`01 program security bit 2 (inhibit Flash verify)`
	`02 program security bit 3 (disable external memory)`
	Example:
	`:020000030501F5 program security bit 2`
	
	Subfunction Code = 06 (Program Status Byte or Boot Vector)
	`ff = 06`
	`ss = 00 program status byte`
	`01 program boot vector`

2nd gen Rx2	`02 program 6x/12x bit (dd:80) (bit 7=0 represents 12x, bit 7=1 represents 6x)`

	`dd = data`
	Example:
	`:030000030601FCF7 program boot vector with 0FCH`

2nd gen Rx2	Example: `:0300000306028072 program config[7]`

	Subfunction Code = 07 (Full Chip Erase)[1]
	Erases all blocks, security bits, and sets status and boot vector to default values
	`ff = 07`
	`ss = don't care`
	`dd = don't care`
	Example:
	`:0100000307F5 full chip erase`

NOTES:
1. Unless FlashMagic version 1.70 or above is used, erasing the status byte and boot vector needs to be implemented six times in a row for proper operation of the P89C51Rx2H and P89C66x devices.

RECORD TYPE	COMMAND/DATA FUNCTION
2nd gen Rx2	Subfunction Code = 0C (Erase Block : each block is 4k sizes) ff = 0C ss = block code as shown below: Block 0 , 0k~4k , 00H Block 1 , 4k~8k , 10H Block 2 , 8k~12k , 20H Block 3 , 12k~16k , 30H Block 4 , 16k~20k , 40H Block 5 , 20k~24k , 50H Block 6 , 24k~28k , 60H Block 7 , 28k~32k , 70H Block 8 , 32k~36k , 80H Block 9 , 36k~40k , 90H Block 10 , 40k~44k , A0H Block 11 , 44k~48k, B0H Block 12 , 48k~52k, C0H Block 13 , 52k~56k, D0H Block 14 , 56k~60k, E0H Block 15 , 60k~64k, F0H Example: :0200000030C20CF (Erase 4k block 2)
04	Display Device Data or Blank Check – Record type 04 causes the contents of the entire Flash array to be sent out the serial port in a formatted display. This display consists of an address and the contents of 16 bytes starting with that address. No display of the device contents will occur if security bit 2 has been programmed. Data to the serial port is initiated by the reception of any character and terminated by the reception of any character. General Format of Function 04 :05xxxx04sssseeeeffcc Where: 05 = number of bytes (hex) in record xxxx = required field, but value is a “don’t care” 04 = “Display Device Data or Blank Check” function code ssss = starting address eeee = ending address ff = subfunction 00 = display data 01 = blank check
2nd gen Rx2	02 = display data in data block (the valid address: 0001~0FFFH)
	cc = checksum Example: :0500000440004FFF0069 display 4000–4FFF Example: :0500000400000FFF02E7 display data in data block (the data in address 0000 is invalid)

RECORD TYPE	COMMAND/DATA FUNCTION
05	Miscellaneous Read Functions General Format of Function 05 :02xxxx05ffsscc Where: 02 = number of bytes (hex) in record xxxx = required field, but value is a “don’t care” 05 = “Miscellaneous Read” function code ffss = subfunction and selection code 0000 = read signature byte – manufacturer id (15H) 0001 = read signature byte – device id # 1 (C2H) 0002 = read signature byte – device id # 2
2nd gen Rx2 2nd gen Rx2	0003 = read 12x/6x bit (bit 7) 0080 = read ROM code revision
	0700 = read security bits 0701 = read status byte 0702 = read boot vector cc = checksum Example: :020000050001F8 read signature byte – device id # 1 Example: :020000050003F6 read config (The bit7=0 represent 12x, bit7=1 represent 6x) Example: :02000005008079 read ROM Code Revision (0A: Rev. A, 0B:Rev. B)
06	Direct Load of Baud Rate (not available with 89C51RB+/RC+/RD+ devices) General Format of Function 06 :02xxxx06hhllcc Where: 02 = number of bytes (hex) in record xxxx = required field, but value is a “don’t care” 06 = ”Direct Load of Baud Rate” function code hh = high byte of Timer 2 ll = low byte of Timer 2 cc = checksum Example: :02000006F500F3
07 2nd gen Rx2	Program data in data block :nnaaaa07dd....ddcc Where: nn = number of bytes (hex) in record aaaa = memory address of first byte in record(the valid address : 0001~0FFFH) dd....dd = data bytes cc = checksum Example: :10008007AF5F67F0602703E0322CFA92007780C3F6

WinISP and FlashMagic: two free Windows–based In–System Programming Utilities available online

How to download WinISP:

1. Direct your browser to the following page: **http://www.semiconductors.com/mcu/download/80c51/flash/**
2. Download “WinISP.exe”
3. Execute WinISP.exe to install the software

How to download FlashMagic:

1. Direct your browser to the following page: **http://www.esacademy.com/software/flashmagic/**
2. Download FlashMagic
3. Execute “flashmagic.exe” to install the software

Some hints on WinISP

Launch the ISP program into a window. Use the mouse to select the part type, the Windows serial port being used, and the oscillator frequency in your application.

CHIP – selects the chip type.

PORT – Selects which port on the host computer is connected to the ISP board. (COM1 – COM4)

RANGE – Selects the beginning and ending address.

WINISP Commands

Load File
Click the LOAD FILE button and enter the desired file name into the dialog box

Erase Blocks
Click the ERASE BLOCKS button and use the mouse to select the desired blocks. Click the ERASE! button.

Blank Check
Click the BLANK CHECK button.

Program Part
Click the PROGRAM PART button.

Read Part
Click the READ PART button.

Verify Part
Click the VERIFY PART button.

Fill Buffer
Enter the starting and ending address in the RANGE boxes. Click the FILL BUFFER button. Enter the data pattern in the next dialog box.

NOTE: The MCU must be running the BOOT ROM program for WINISP or FlashMagic to be able to communicate with the microcontroller.

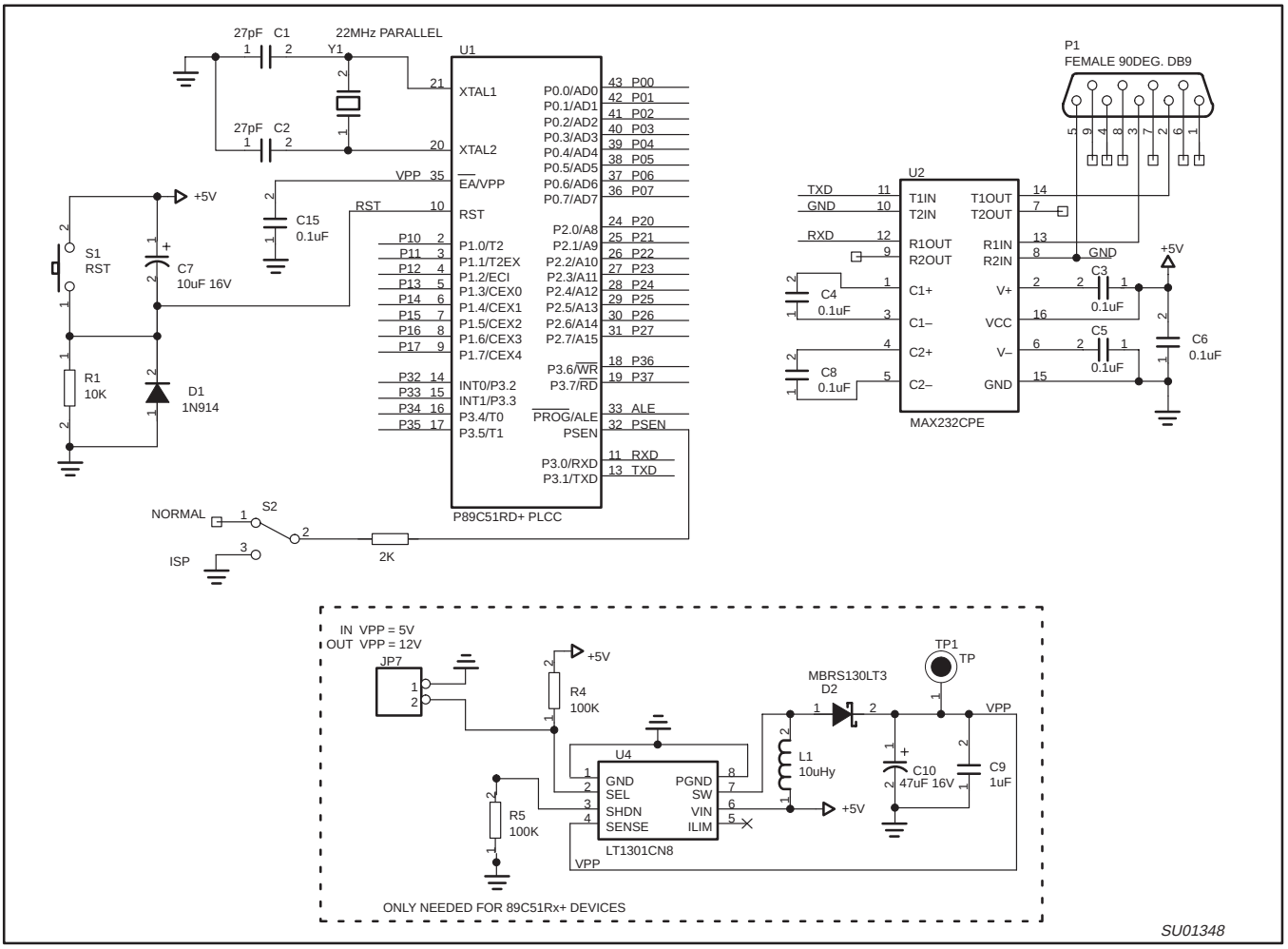

Figure 3. Typical ISP Implementation

In Application Programming Method

Several In Application Programming (IAP) calls are available for use by an application program to permit selective erasing and programming of Flash sectors. All calls are made through a common interface, PGM_MTP. The programming functions are selected by setting up the microcontroller's registers before making a call to PGM_MTP at FFF0H. The oscillator frequency is an integer number rounded down to the nearest megahertz. For example, set R0 to 11 for 11.0592 MHz. Results are returned in the registers. The IAP calls are shown in Table 4.

Interrupts and the watchdog timer must be disabled while IAP subroutines are executing.

Table 4. IAP calls

IAP CALL	PARAMETER
PROGRAM BYTE	Input Parameters: R0 = osc freq (integer) R1 = 02h or R1 = 82h (WDT feed) DPTR = address of byte to program ACC = byte to program Return Parameter ACC = 00 if pass, !00 if fail
ERASE 4K CODE BLOCK (2nd generation Rx2)	Input Parameters: R0 = osc freq (integer) R1 = 0Ch or R1 = 8Ch (WDT feed) DPH = address of 4k code block DPH = 00H , 4k block 0, 0k~4k DPH = 10H , 4k block 1, 4k~8k DPH = 20H , 4k block 2, 8k~12k DPH = 30H , 4k block 3, 12k~16k DPH = 40H , 4k block 4, 16k~20k DPH = 50H , 4k block 5, 20k~24k DPH = 60H , 4k block 6, 24k~28k DPH = 70H , 4k block 7, 28k~32k DPH = 80H , 4k block 8, 32k~34k DPH = 90H , 4k block 9, 34k~38k DPH = A0H , 4k block 10, 38k~42k DPH = B0H , 4k block 11, 42k~46k DPH = C0H , 4k block 12, 46k~50k DPH = D0H , 4k block 13, 50k~54k DPH = E0H , 4k block 14, 54k~58k DPH = F0H , 4k block 15, 58k~62k DPL = 00h Return Parameter ACC = 00 if pass, !=00 if fail
ERASE 8K / 16K CODE BLOCK	Input Parameters: R0 = osc freq (integer) R1 = 01h or R1 = 81h (WDT feed) DPH = address of code block DPH = 00H , block 0 , 0k~8k DPH = 20H , block 1 , 8k~16k DPH = 40H , block 2 , 16~32k DPH = 80H , block 3 , 32k~48k DPH = C0H , block 4 , 48k~64k DPH = 00h Return Parameter ACC = 00 if pass, !=0 if fail
ERASE STATUS BYTE & BOOT VECTOR[1]	Input Parameters: R0 = osc freq (integer) R1 = 04h or R1 = 84h (WDT feed) DPH = 00H DPL = don't care Return Parameter ACC = 00 if pass, !=0 if fail

NOTES:
1. Unless FlashMagic version 1.70 or above is used, erasing the status byte and boot vector needs to be implemented six times in a row for proper operation of the P89C51Rx2H and P89C66x devices.

IAP CALL	PARAMETER
PROGRAM SECURITY BITS	Input Parameters: R0 = osc freq (integer) R1 = 05h or R1 = 85h (WDT feed) DPH = 00h DPL = 00h , security bit #1 DPL = 01h , security bit #2 DPL = 02h , security bit #3 Return Parameter ACC = 00 if pass, !=0 if fail
PROGRAM STATUS BYTE	Input Parameters: R0 = osc freq (integer) R1 = 06h or R1 = 86h (WDT feed) DPH = 00h DPL = 00H – program status byte ACC = status byte Return Parameter ACC = 00 if pass, !=0 if fail
PROGRAM BOOT VECTOR	Input Parameters: R0 = osc freq (integer) R1 = 06h or R1 = 86h (WDT feed) DPH = 00h DPL = 01H – program boot vector ACC = boot vector Return Parameter ACC = 00 if pass, !=0 if fail
PROGRAM 6-CLK/12-CLK CONFIGURATION BIT (2nd generation Rx2)	Input Parameters: R0 = osc freq (integer) R1 = 06h or R1 = 86h (WDT feed) DPH = 00h DPL = 02H – program config bit ACC = 80H (MSB = 6-clk/12-clk bit) Return Parameter ACC = 00 if pass, !=0 if fail
PROGRAM DATA BLOCK (2nd generation Rx2)	Input Parameters: R0 = osc freq (integer) R1 = 0Dh or R1 = 8Dh (WDT feed) DPTR = address of byte to program (valid addresses = 0001h~0FFFh) ACC = data Return Parameter ACC = 00 if pass, !=0 if fail
READ DEVICE DATA	Input Parameters: R0 = osc freq (integer) R1 = 03h or R1 = 83h (WDT feed) DPTR = address of byte to read Return Parameter ACC = value of byte read
READ DATA BLOCK (2nd generation Rx2)	Input Parameters: R0 = osc freq (integer) R1 = 0Eh or R1 = 8Eh (WDT feed) DPTR = address of byte to read (valid addresses = 0001h~0FFFh) Return Parameter ACC = value of byte read
READ MANUFACTURER ID	Input Parameters: R0 = osc freq (integer) R1 = 00h or R1 = 80h (WDT feed) DPH = 00h DPL = 00h – read manufacturer ID Return Parameter ACC = value of byte read

IAP CALL	PARAMETER
READ DEVICE ID #1	Input Parameters: R0 = osc freq (integer) R1 = 00h or R1 = 80h (WDT feed) DPH = 00h DPL = 01h – read device ID #1 Return Parameter ACC = value of byte read
READ DEVICE ID #2	Input Parameters: R0 = osc freq (integer) R1 = 00h or R1 = 80h (WDT feed) DPH = 00h DPL = 02h – read device ID #2 Return Parameter ACC = value of byte read
READ SECURITY BITS	Input Parameters: R0 = osc freq (integer) R1 = 07h or R1 = 87h (WDT feed) DPH = 00h DPL = 00h – read lock byte Return Parameter ACC = value of byte read
READ STATUS BYTE	Input Parameters: R0 = osc freq (integer) R1 = 07h or R1 = 87h (WDT feed) DPH = 00h DPL = 01h – read status byte Return Parameter ACC = value of byte read
READ BOOT VECTOR	Input Parameters: R0 = osc freq (integer) R1 = 07h or R1 = 87h (WDT feed) DPH = 00h DPL = 02h – read boot vector Return Parameter ACC = value of byte read
READ CONFIG (2nd generation Rx2)	Input Parameters: R0 = osc freq (integer) R1 = 00h or R1 = 80h (WDT feed) DPH = 00h DPL = 03h – read config byte Return Parameter ACC = value of byte read
READ REVISION (2nd generation Rx2)	Input Parameters: R0 = osc freq (integer) R1 = 00h or R1 = 80h (WDT feed) DPH = 00h DPL = 80h – read revision of ROM code Return Parameter ACC = value of byte read

REVISION HISTORY

Rev	Date	Description
_10	20030311	**(9397 750 10685)** Modifications: ● Updated status byte/boot vector erase info
_9	20020624	**(9397 750 10137)**
_8	20011011	**(9397 750 08956)**
_7	20000728	Previous release.

In-circuit and In-application programming of the 89C51Rx+/Rx2/66x microcontrollers

AN461

Definitions

Short-form specification — The data in a short-form specification is extracted from a full data sheet with the same type number and title. For detailed information see the relevant data sheet or data handbook.

Limiting values definition — Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 60134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

Application information — Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Disclaimers

Life support — These products are not designed for use in life support appliances, devices or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes, without notice, in the products, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Contact information

For additional information please visit
http://www.semiconductors.philips.com. Fax: **+31 40 27 24825**

For sales offices addresses send e-mail to:
sales.addresses@www.semiconductors.philips.com.

Date of release: 03-03

Document order number: 9397 750 10685

Let's make things better.

Philips Semiconductors

PHILIPS