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

Author: Bill Houghton

2000 May 25
Supersedes data of 2000 Jan 13
IC20 Data Handbook

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 KB Flash EPROM
- Flash EPROM is sectored to allow the user to erase and reprogram sectors
- 1 KB 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 33Mhz @ 12 clocks/instruction; 0 to 20MHz @ 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 8KB or 16KB, as indicated below. Different amounts of memory are present depending on the specific device as shown in Table 1 below.

Table 1. Memory Block of Philips Flash Microcontrollers

Device	Total Memory	8KB (0–1FFF)	8KB (2000–3FFF)	16KB (4000–7FFF)	16KB (8000–BFFF)	16KB (C000–FFFF)
89C51RB+	16KB	X	X			
89C51RB2	16KB	X	X			
89C51RC+	32KB	X	X	X		
89C51RC2	32KB	X	X	X		
89C51RD+	64KB	X	X	X	X	X
89C51RD2	64KB	X	X	X	X	X
89C660	16KB	X	X			
89C662	32KB	X	X	X		
89C664	64KB	X	X	X	X	X

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 have a 1KB factory-masked ROM located in the upper 1KB of code memory space from FC00 to FFFF. This 1KB ROM is in addition to the memory blocks shown in Table 1. 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 0000-h (“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 ENDBOOT 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 16KB and 32KB 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).

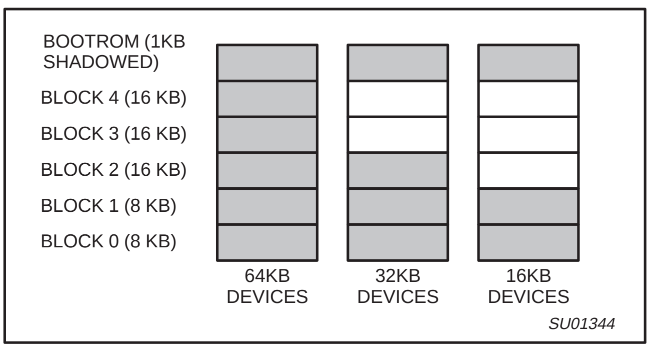

Figure 1. Memory Space in Flash Microcontrollers

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 2). 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.

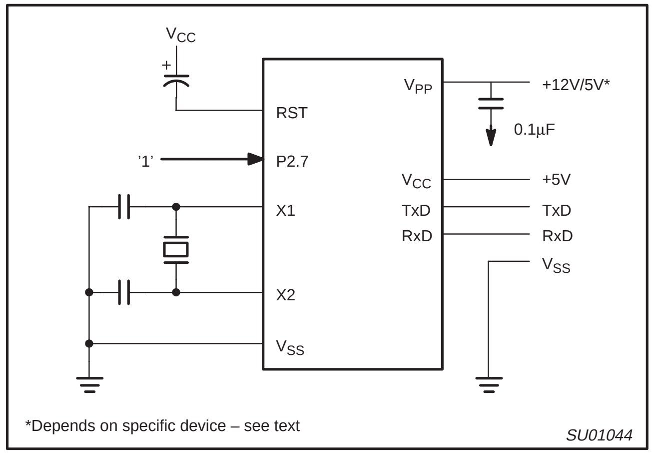

Figure 2. In-System Programming with a Minimum of Pins

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.

The boot loader can also be executed by holding PSEN low, EA' greater than V_{IH} (such as +12V), P2.7 and ALE HIGH (or not connected) at the falling edge of RESET. This is the same effect as having a non-zero status byte. This allows an application to be built that will normally execute the end user's code but can be manually forced into ISP operation.

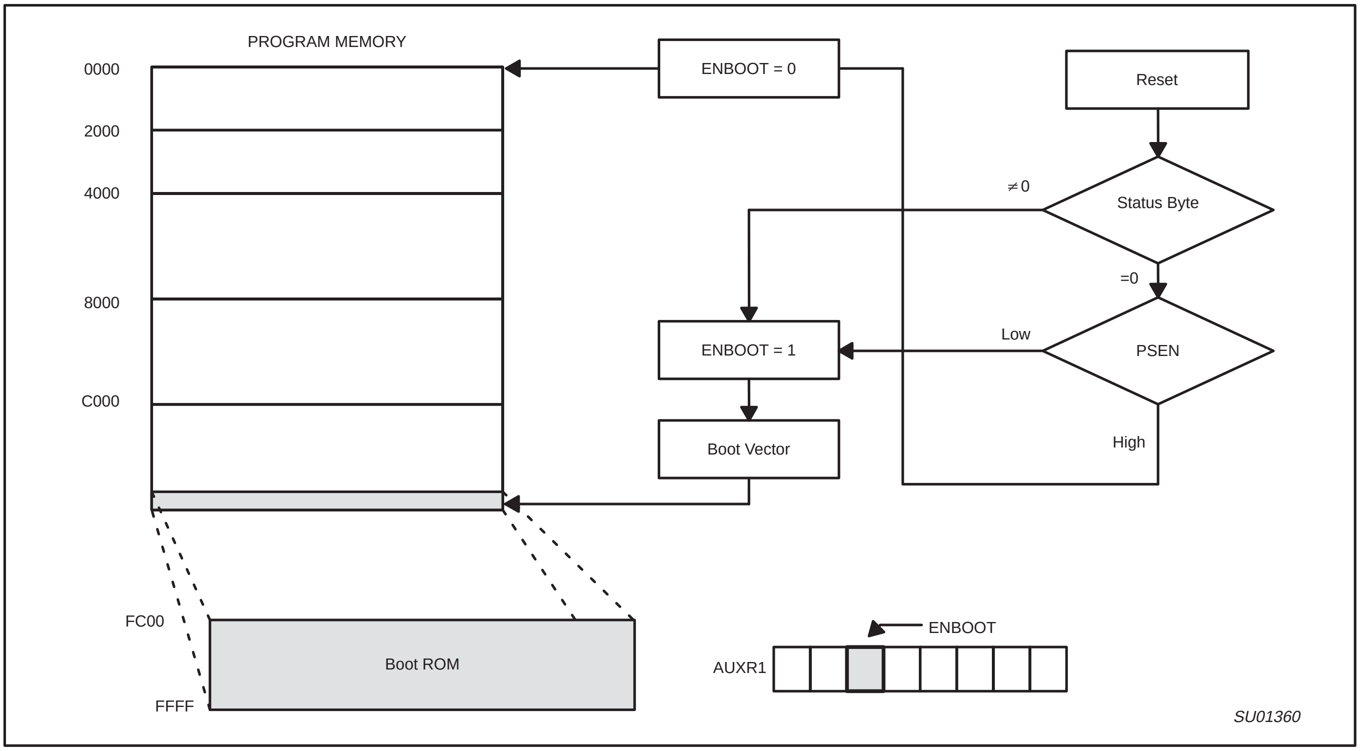

Figure 3. ISP Flow Chart

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.)

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 2 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 2.

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.

WinISP, a software utility to implement ISP programming with a PC, is available from the Philips website (www.semiconductors.philips.com).

Table 2. 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`
	`block 3, 32k to 48k, 80H`
	`block 4, 48k to 64k, C0H`
	Example:
	`:0200000301C03A erase block 4`
	
	Subfunction Code = 04 (Erase Boot Vector and Status Byte)
	`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`
	Example:
	`:030000030601FCF7 program boot vector with 0FCH`
	
	Subfunction Code = 07 (Full Chip Erase – not available with 89C51RB+/RC+/RD+ devices)
	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`

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`
	`cc = checksum`
	Example:
	`:0500000440004FFF0069 display 4000–4FFF`

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`
	
	`0700 = read security bits`
	`0701 = read status byte`
	`0702 = read boot vector`
	`cc = checksum`
	Example:
	`:020000050001F8 read signature byte – device id # 1`

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`

WINISP – The Windows In-System Programmer Utility Program

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:

- 89C51RC+
- 89C51RD+

PORT – Selects which port on the host computer is connected to the ISP board

- COM1
- COM2
- COM3
- COM4

RANGE – Selects the beginning and ending address

- START
- END

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 to be able to communicate with the microcontroller.

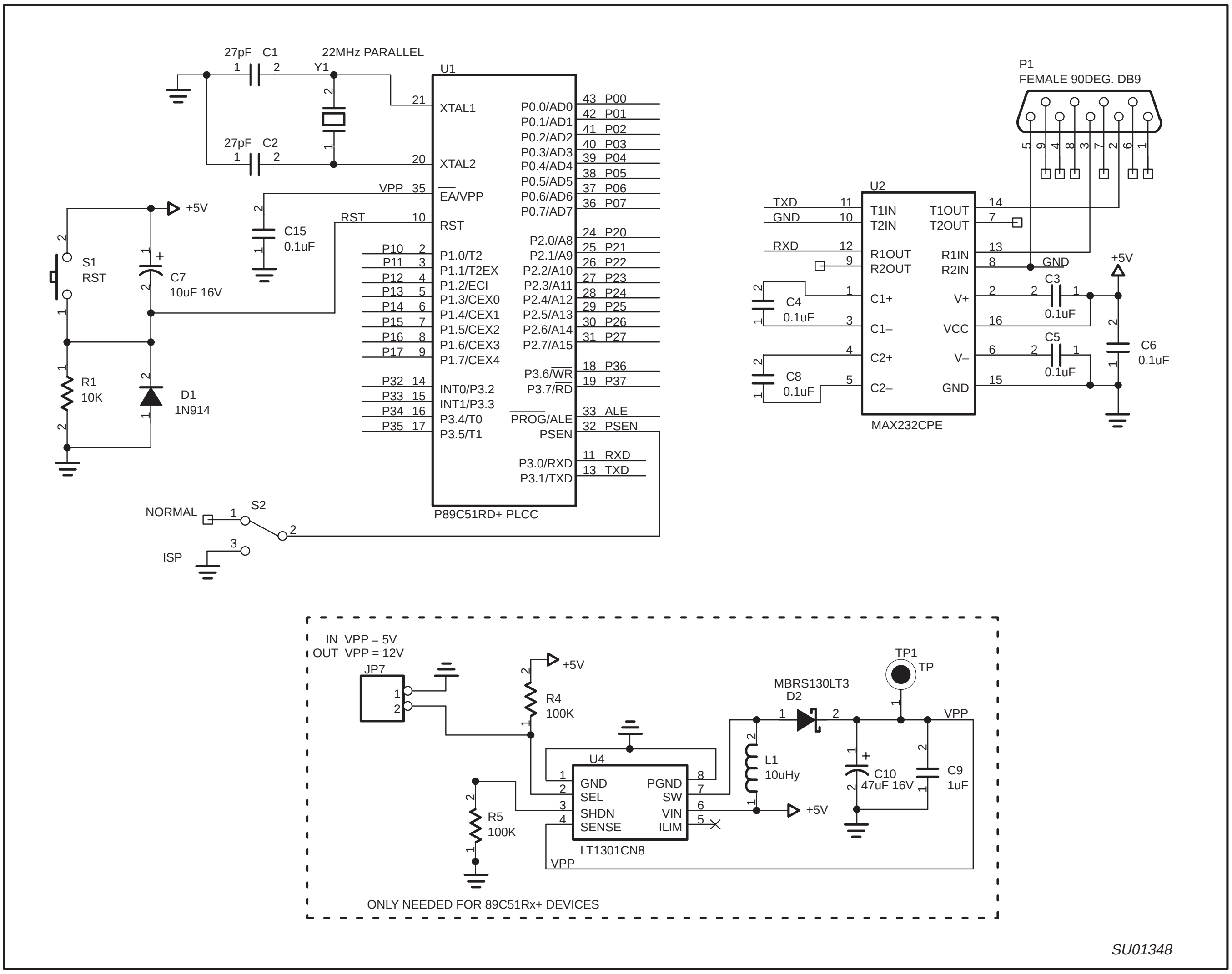

Figure 4. Typical ISP Implementation

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 3.

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

Table 3. IAP calls

<table>
<tr><th>IAP CALL</th><th>PARAMETER</th></tr>
<tr><td>PROGRAM DATA BYTE</td><td>Input Parameters:
R0 = osc freq (integer)
R1 = 02h
DPTR = address of byte to program
ACC = byte to program
Return Parameter
ACC = 00 if pass, !00 if fail
Sample routine:
;***** Program Device Data (DData) *****
;***** ACC holds data to write
;***** DPTR holds address of byte to write *****
;***** Returns with ACC = 00h if successful, else ACC NEQ 00h
WRData:
MOV AUXR1,#20H ;set the ENBOOT bit
MOV R0, #11 ;FOSC
MOV R1,#02H ;program data function
MOV A,Mydata ;data to write
MOV DPTR,Address ;specify address of byte to read
CALL PGM_MTP ;execute the function
RET</td></tr>
<tr><td>ERASE BLOCK</td><td>Input Parameters:
R0 = osc freq (integer)
R1 = 01h
DPH = block code as shown below:
block 0, 0k to 8k, 00H
block 1, 8k to 16k, 20H
block 2, 16k to 32k, 40H
block 3, 32k to 48k, 80H
block 4, 48k to 64k, C0H

DPL = 00h
Return Parameter
none
Sample routine:
;***** Erase Code Memory Block *****
;***** DPH (7:5) indicates which of the 5 blocks to erase
;***** DPTR values for the blocks are:
; 0000h = block 0
; 2000h = block 1
; 4000h = block 2
; 6000h = block 3
; 8000h = block 4

ERSBLK:
MOV AUXR1,#20H ;set the ENBOOT bit
MOV R0, #11 ;FOSC
MOV R1,#01H ;erase block
MOV DPTR,#BLk_NUM ;specify which block
CALL PGM_MTP ;execute the function
RET</td></tr>
</table>

<table>
<tr><th>IAP CALL</th><th>PARAMETER</th></tr>
<tr><td>ERASE BOOT VECTOR & STATUS BYTE</td><td>Input Parameters:<pre>
  R0 = osc freq (integer)
  R1 = 04h
  DPH = 00h
  DPL = don’t care
</pre>Return Parameter<pre>
  none
</pre>Sample routine:<pre>
  ;***** Erase Boot Vector (BV) & Status Byte (SB) *****
  ;***** Note: This command erases BOTH the SB & BV

  ERSBBV;
          MOV   AUXR1,#20H   ;set the ENBOOT bit
          MOV   R0, #11   ;FOSC
          MOV   R1,#04H   ;erase status byte & boot vector
          MOV   DPH,#00h   ;we don’t care about DPL
          CALL   PGM_MTP   ;execute the function
          RET
</pre></td></tr>
<tr><td>PROGRAM SECURITY BIT</td><td>Input Parameters:<pre>
  R0 = osc freq (integer)
  R1 = 05h
  DPH = 00h
  DPL = 00h – security bit # 1 (inhibit writing to Flash)
        01h – security bit # 2 (inhibit Flash verify)
        02h – security bit # 3 (disable external memory)
</pre>Return Parameter<pre>
  none
</pre>Sample routines:<pre>
  ;***** Program Security Bit1 *****
  ;***** DPTR indicates security bit to program *****

  WRSB1:
          MOV   AUXR1,#20H   ;set the ENBOOT bit
          MOV   R0,#11   ;FOSC
          MOV   R1,#05H   ;program security bit function
          MOV   DPTR,#0000h   ;specify security bit 1
          CALL   PGM_MTP   ;execute the function
          RET
  ;***** Program Security Bit2 *****
  ;***** DPTR indicates security bit to program *****

  WRSB2:
          MOV   AUXR1,#20H   ;set the ENBOOT bit
          MOV   R0,#11   ;FOSC
          MOV   R1,#05H   ;program security bit function
          MOV   DPTR,#0001h   ;specify security bit 2
          CALL   PGM_MTP   ;execute the function
          RET
  ;***** Program Security Bit3 *****
  ;***** DPTR indicates security bit to program *****

  WRSB3:
          MOV   AUXR1,#20H   ;set the ENBOOT bit
          MOV   R0,#11   ;FOSC
          MOV   R1,#05H   ;program security bit function
          MOV   DPTR,#0002h   ;specify security bit 3
          CALL   PGM_MTP   ;execute the function
          RET
</pre></td></tr>
</table>

<table>
<tr><th>IAP CALL</th><th>PARAMETER</th></tr>
<tr><td>PROGRAM STATUS BYTE</td><td>Input Parameters:<pre>
R0 = osc freq (integer)
R1 = 06h
DPH = 00h
DPL = 00h – program status byte
ACC = status byte
</pre>Return Parameter<pre>
ACC = status byte
</pre>Sample routine:<pre>
;***** Program Status Byte (SB) *****
;***** DPTR indicates program status byte *****
;***** ACC holds new value of Status Byte to program *****

WRSB:
          MOV   AUXR1,#20H   ;set the ENBOOT bit
          MOV   R0,#11   ;FOSC
          MOV   R1,#06H   ;program status byte or boot vector
          MOV   DPTR,#0000h   ;specify status byte
          MOV   A,NEW_SB   ;
          CALL   PGM_MTP   ;execute the function
          RET
</pre></td></tr>
<tr><td>PROGRAM BOOT VECTOR</td><td>Input Parameters:<pre>
R0 = osc freq (integer)
R1 = 06h
DPH = 00h
DPL = 01h – program boot vector
ACC = boot vector
</pre>Return Parameter<pre>
ACC = boot vector
</pre>Sample routine:<pre>
;***** Program Boot Vector (BV) *****
;***** DPTR indicates program boot vector *****
;***** ACC holds new value of boot vector to program *****

WRBV:
          MOV   AUXR1,#20H   ;set the ENBOOT bit
          MOV   R0,#11   ;FOSC
          MOV   R1,#06H   ;program status byte or boot vector
          MOV   DPTR,#0001h   ;specify boot vector
          MOV   A,NEW_SB   ;new value for the boot vector
          CALL   PGM_MTP   ;execute the function
          RET
</pre></td></tr>
<tr><td>READ DEVICE DATA</td><td>Input Parameters:<pre>
R1 = 03h
DPTR = address of byte to read
</pre>Return Parameter<pre>
ACC = value of byte read
</pre>Sample routine:<pre>
;*****reads the Device Data (DData) *****
;***** DData returned in ACC *****
;***** DPTR holds address of byte to read *****
RDData:
          MOV   AUXR1,#20H   ;set the ENBOOT bit
          MOV   R0,#11   ;FOSC
          MOV   R1,#03H   ;read data function
          MOV   DPTR,Address   ;specify address of byte to read
          CALL   PGM_MTP   ;execute the function
          RET
</pre></td></tr>
</table>

<table>
<tr><th>IAP CALL</th><th>PARAMETER</th></tr>
<tr><td>READ MANUFACTURER ID</td><td>Input Parameters:
R0 = osc freq (integer)
R1 = 00h
DPH = 00h
DPL = 00h (manufacturer ID)
Return Parameter
ACC = value of byte read
Sample routine:
;*****reads the Manufacturer ID (MID) *****
;***** MID returned in ACC (should be 15h for Philips)
RDMID:
MOV AUXR1,#20H ;set the ENBOOT bit
MOV R0,#11 ;FOSC
MOV R1,#00H ;read misc function
MOV DPTR,#0000H ;specify MID
CALL PGM_MTP ;execute the function
RET</td></tr>
<tr><td>READ DEVICE ID # 1</td><td>Input Parameters:
R0 = osc freq (integer)
R1 = 00h
DPH = 00h
DPL = 01h (device ID # 1)
Return Parameter
ACC = value of byte read
Sample routine:
;*****reads the Device ID 1 (DID1) *****
;***** DID1 returned in ACC
RDDID1:
MOV AUXR1,#20H ;set the ENBOOT bit
MOV R0,#11 ;FOSC
MOV R1,#00H ;read misc function
MOV DPTR,#0001H ;specify device id 1
CALL PGM_MTP ;execute the function
RET</td></tr>
<tr><td>READ DEVICE ID # 2</td><td>Input Parameters:
R0 = osc freq (integer)
R1 = 00h
DPH = 00h
DPL = 02h (device ID # 2)
Return Parameter
ACC = value of byte read
Sample routine:
;*****reads the Device ID 2 (DID2) *****
;***** DID2 returned in ACC
RDDID2:
MOV AUXR1,#20H ;set the ENBOOT bit
MOV R0,#11 ;FOSC
MOV R1,#00H ;read misc function
MOV DPTR,#0002H ;specify device id 2
CALL PGM_MTP ;execute the function
RET</td></tr>
<tr><td>READ SECURITY BITS</td><td>Input Parameters:
R0 = osc freq (integer)
R1 = 07h
DPH = 00h
DPL = 00h (security bits)
Return Parameter
ACC = value of byte read
Sample routine:
;*****reads the Security Bits (SBits) *****
;***** SBits returned in ACC (2:0)
RDSBits:
MOV AUXR1,#20H ;set the ENBOOT bit
MOV R0,#11 ;FOSC
MOV R1,#07H ;read misc function
MOV DPTR,#0000H ;specify security bits
CALL PGM_MTP ;execute the function
RET</td></tr>
</table>

<table>
<tr><th>IAP CALL</th><th>PARAMETER</th></tr>
<tr><td>READ STATUS BYTE</td><td>Input Parameters:
<code>R0 = osc freq (integer)</code>
<code>R1 = 07h</code>
<code>DPH = 00h</code>
<code>DPL = 01h (status byte)</code>
Return Parameter
<code>ACC = value of byte read</code>
Sample routine:<pre>;*****reads the Status Byte (SB) *****
;***** SB returned in ACC
RDSB:
          MOV   AUXR1,#20H   ;set the ENBOOT bit
          MOV   R0,#11   ;FOSC
          MOV   R1,#07H   ;read misc function
          MOV   DPTR,#0001H   ;specify status byte
          CALL   PGM_MTP   ;execute the function
          RET</pre></td></tr>
<tr><td>READ BOOT VECTOR</td><td>Input Parameters:
<code>R0 = osc freq (integer)</code>
<code>R1 = 07h</code>
<code>DPH = 00h</code>
<code>DPL = 02h (boot vector)</code>
Return Parameter
<code>ACC = value of byte read</code>
Sample routine:<pre>;*****reads the Boot Vector (BV) *****
;***** BV returned in ACC
RDBV:
          MOV   AUXR1,#20H   ;set the ENBOOT bit
          MOV   R0,#11   ;FOSC
          MOV   R1,#07H   ;read misc function
          MOV   DPTR,#0002H   ;specify boot vector
          CALL   PGM_MTP   ;execute the function
          RET</pre></td></tr>
<tr><td>FULL CHIP ERASE</td><td>Input Parameters:
<code>R0 = osc frequency</code>
<code>R1 = 08h</code>
<code>DPH = don’t care</code>
<code>DPL = don’t care</code>
Return Parameter
<code>none</code>
Sample routine:<pre>;*****Full Chip Erase *****
FCE:
          MOV   AUXR1,#20H   ;set the ENBOOT bit
          MOV   R0,#11   ;FOSC
          MOV   R1,#08H   ;read misc function
          CALL   PGM_MTP   ;execute the function
                                    ;returns to the bootrom at FC00h
                                    ;stack automatically adjusted</pre></td></tr>
</table>

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 134). 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.

Philips Semiconductors
811 East Arques Avenue
P.O. Box 3409
Sunnyvale, California 94088–3409
Telephone 800-234-7381

© Copyright Philips Electronics North America Corporation 2000
All rights reserved. Printed in U.S.A.

Date of release: 7-00

Document order number: 9397 750 07334

Let's make things better.

Philips Semiconductors

PHILIPS