



MSP430 Family

Starter Kit Evaluation Kit Manual

1999

Mixed-Signal Products

SLAS191A

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Contents

<i>Section</i>	<i>Title</i>	<i>Page</i>
1	Getting Started	1-1
1.1	Installing the Software	1-1
1.2	Hardware Installation	1-2
1.2.1	The STK/EVK-PCB Operating Conditions	1-2
1.2.2	How to Install the Hardware	1-3
1.2.3	Programming the Monitor Software Into an Erased EPROM (EVK Only)	1-5
1.2.4	MSP-STK/EVK430x320 Target Connectors	1-6
1.2.5	MSP-EVK430x330 Target Connectors	1-7
1.2.6	The LCD	1-8
1.2.7	Schematic for the MSP-STK/EVK430x320	1-10
1.2.8	Schematic for the MSP-EVK430x330	1-11
1.2.9	Starting the STK Demo Program	1-12
1.2.10	Executing a Program with the STK	1-12
1.2.11	How to Use Breakpoints	1-22
1.2.12	Accessing the Port on the MSP430x320 STK/EVK	1-23
1.2.13	How to Use an Interrupt Routine	1-24
1.3	Loading a Program Into the EPROM Via the Terminal	1-25
2	Monitor Commands	2-1
2.1	Syntax Conventions	2-1
2.2	Memory Organization	2-1
2.3	Commands	2-2
3	Monitor Restrictions	3-1
3.1	Register R4	3-1
3.2	The Instruction CALL R4	3-1
3.3	Peripheral Hardware/Registers	3-2
3.4	RAM Locations for the Monitor	3-2
3.5	Writing Data Into the EPROM	3-5
4	Treatment of Interrupts	4-1
4.1	Use of Interrupts in the Monitor Environment	4-1
5	Half Duplex Monitor Software UART	5-1
5.1	Transmission Parameters of the Software UART	5-1
5.2	Identification of Bit Pattern AA55h	5-1
5.3	Special Treatment of <ESC> in the Software UART	5-4
5.4	Transmitting One Character	5-5
5.5	Transmitting a String	5-7
5.6	Receiving a Character	5-8
6	Using Interrupt Vectors in the EPROM	6-1
6.1	The Identification Bit Pattern After a Reset	6-1

7	Memory Configurations for MSP430 Devices	7-1
A	Difference Between STK and EVK	A-1

List of Illustrations

<i>Figure</i>	<i>Title</i>	<i>Page</i>
1-1	MSP-STK430 Program Group	1-2
1-2	MSP/EVK430 Program Group	1-2
1-3	Terminal Screen	1-4
1-4	Programming the Device	1-5
1-5	MSP-STK/EVK430x320 Target Connectors	1-6
1-6	MSP-EVK430x330 Target Connectors	1-7
1-7	Supplied LCD Mechanical Data	1-8
1-8	LCD Segment Digits	1-9
1-9	LUXMETER Demo Program	1-12
1-10	Properties	1-13
1-11	Getting Started Demo Program	1-14
1-12	ASM430 Assembler Window	1-15
1-13	gs_stk1.asm Window Display	1-16
1-14	gs_stk1.asm	1-17
1-15	Terminal Window	1-18
1-16	Terminal Transfer	1-19
1-17	HyperTerminal Display	1-20
1-18	Terminal execute gs_stk1.txt	1-21
1-19	Terminal Breakpoint	1-23
1-20	Basic Timer Interrupt Routine	1-24
1-21	Terminal Interrupt	1-25
1-22	Programming Voltage and Jumper Location	1-26
1-23	EPROM LCD and Interrupt Routine	1-27
1-24	HyperTerminal Window	1-28
1-25	Transfers \ Send Text File	1-28
1-26	Send Text File Dialog Box	1-29
1-27	Burn Failed Message	1-30
2-1	Memory Organization	2-1
2-2	Help Command	2-2
2-3	Byte, Word Commands	2-3
2-4	Initializing the Terminal Program Command	2-4
2-5	User Reset Command	2-5
2-6	Register Command	2-6
2-7	Register Specified Command	2-7
2-8	Modify One Register	2-8
2-9	Modify Additional Registers	2-9
2-10	Revise Memory Modification	2-10
2-11	Memory Byte, Word Command	2-11
2-12	Memory Modification	2-12
2-13	Revised Memory Modification	2-13

2-14	Transfer Data Command	2-14
2-15	EPROM Erase Check Command	2-15
2-16	Location of a Breakpoint Command	2-16
2-17	Set a Breakpoint Command	2-17
2-18	Clearing a Breakpoint Command	2-18
2-19	Clearing a Breakpoint Location	2-18
2-20	Starting the Application Command	2-19
3-1	CALL R4 Instruction Code	3-1
3-2	RAM Area 272h to 3FEh	3-3
3-3	RAM Area 200h to 3FFh for the MSP430x32x Family	3-4
3-4	Temporary Burn Routine in the MSP430x32x RAM Area	3-5
4-1	Monitor Interrupts for the MSP430x32x Family	4-1
4-2	P0.0 Interrupt Example	4-2
5-1	Identification of Bit Pattern AA55h for the MSP430X32x Family	5-2
5-2	Special Treatment of ESC	5-4
5-3	Transmitting the s Character	5-6
6-1	Identifying AA55h After Reset	6-2
7-1	Memory Map of the STK/EVK	7-1
7-2	Memory Map of the STK/EVK430X32x	7-2
7-3	Memory Map of the STK/EVK430X33x	7-3

List of Tables

<i>Table</i>	<i>Title</i>	<i>Page</i>
1-1	LCD Connector	1-9
3-1	Peripheral Registers and Bits	3-2
4-1	Type of Interrupt	4-2
4-2	Interrupt Vectors for the MSP430x32x Family	4-3
4-3	Interrupt Vectors for the MSP430x33x Family	4-3
5-1	Function/Vector	5-1

1 Getting Started

This chapter provides installation and programming instructions for the starter Kit MSP-STK430x320, and the evaluation Kits MSP-EVK430x320 and MSP-EVK430x320.

NOTE:

This manual covers both the MSP-STK430x320 and MSP-EVK430x320 kits. The actual icons and/or windows on the computer screen may differ from those shown in the book due to software version upgrades.

In this documentation, the term STK/EVK is used interchangeably to represent both kits. The programs used in this manual are provided in the installation directory *..lstkexamples*, or in *..levkexamples*. These programs demonstrate the user-friendly environment of the MSP430 starter and evaluation kits.

The main difference between the MSP-EVK430x320 and the MSP-EVK430x330, besides the peripheral blocks, is the memory map. The memory map is described in chapter 2 under *Memory Organization*.

1.1 Installing the Software

Exit all MS-Windows™ programs prior to loading this software.

The setup program installs all necessary files for running programs on the STK/EVK. The setup program adds the appropriate program group and icons to the Windows program manager.

To install the software:

1. Insert MSP430-STK/EVK disk 1 in the floppy drive and run *setup.exe*.
2. In the Select Components window, choose the applications to load. The Starter Kit and Simulation Environment button are the default. Make a choice and select Next.
3. Choose the desired COM port for the hardware connection. The default is set to COM port 2. Select the port to be used, then select Next.
4. Read the licensing agreement and answer the question at the bottom of the screen. If YES is selected to the licensing agreement, a prompt to close all other applications running in Windows will appear.
5. The *setup.exe* program automatically creates and loads the MSP430-STK/EVK software to the recommended directory (C:\ADT430), unless otherwise indicated. Choose the directory and select Next.
6. Insert Disk 2 when prompted. Select OK when disk 2 is in the floppy drive.
7. Setup.exe places MSP430 icons in the ADT430 program folder, unless otherwise indicated. Select Next.
8. Setup is complete. Please take a moment to **fill out and return the registration card** to ensure receiving further software updates.

The ADT430 program folder should contain the following icons: Simulation Environment, Read me SIM, Help SIM, ASM430 Assembler, Sensor Demo, and UnInstall the STK430 and EVK430 Terminals.

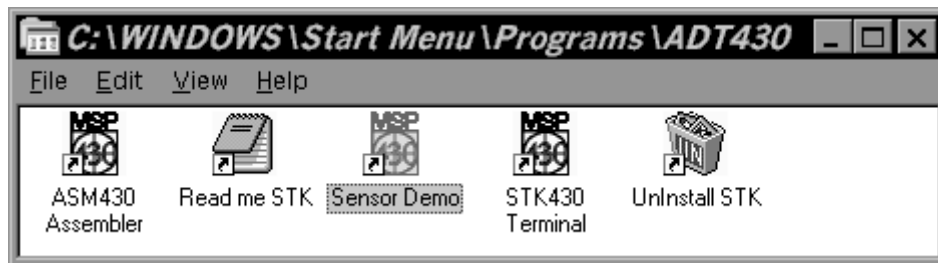


Figure 1-1. MSP-STK430 Program Group



Figure 1-2. MSP/EVK430 Program Group

NOTE: EVK/STK430 Terminal icon

Before clicking the EVK/STK430 Terminal icon, ensure the standard Windows Applications Terminal and Notepad are installed in the Windows directory. The Windows terminal emulator is configured to use serial port COM2 by default.

1.2 Hardware Installation

The STK/EVK kit hardware includes the STK/EVK-PCB and one 9-pin cable for connection to the PC's serial communication port. The EVK also contains the PRG430 along with the necessary cables.

1.2.1 The STK/EVK-PCB Operating Conditions

Temperature range	10°C – 45°C
Humidity	40% – 70%
Current consumption (Approximate Values)	≈0.7 mA at 3 V, 25°C, no connection to serial port ≈1 mA at 5 V, 25°C, no connection to serial port ≈5 mA at 3 V, 25°C, serial port connected ≈5.5 mA at 5 V, 25°C, serial port connected Note: The current consumption is measured without any additional external connections.
Operating voltage	3 V or 5 V Note: The operating voltage is selectable as 3 V or 5 V on the STK/EVK-PCB by making the appropriate connections (see Figure 1-5 or 1-6, depending on the system you have).

1.2.2 How to Install the Hardware

This section targets three main parts: the setup of the serial interface, the programming adapter, and the power supply.

LCD and UVEPROM

The EVK is supplied with the LCD and a UVEPROM, separate from the EVK PCB. Install the EPROM and LCD into the EVK PCB before proceeding. Refer to the STK/EVK target connectors section for the correct LCD orientation. The STK has a one-time programmable (OTP) device mounted on the PCB.

Serial Communication

Connect the STK/EVK-PCB to the serial communications port of the PC using the 9-pin cable that is supplied. Use the *Settings/Communications. . .* command to set the communication port to which the cable is connected. The default selection is COM2.

The HyperTerminal (which uses the Terminal icon) is used to communicate to the STK/EVK MSP430 chip via a Monitor Program that has already been downloaded into the EPROM of the chip. The HyperTerminal settings used are: data bits—7; parity—even; stop bits—1; flow control—none.

Voltage Sources

The following four sources supply the voltage for the STK/EVK-PCB:

1. **PC serial interface**
Typically, the serial interface of the PC supplies the STK/EVK-PCB voltage using the 9-pin cable. The user must determine if the serial interface can meet the electrical requirements of the STK/EVK PCB. Some PC manufacturers do not source enough current and voltage from the serial port to power the STK/EVK devices. If this is the case, a 9-volt battery can be connected in parallel to the serial port.
2. **3.6 V lithium battery**
A battery can be assembled onto the STK/EVK that supplies the STK/EVK-PCB with power.
3. **Programming Adapter**
The Programming Adapter's power supply is stabilized to 5 V (this can only be used on the EVK).
4. **Target system**
The power supply from a target system can be connected to the STK/EVK-PCB. A supply voltage of 3 V to 5 V can be used.

CAUTION: Different Power Supplies

If there is more than one power supply connected, the STK/EVK is supplied with the higher voltage, provided the data sheet supply voltage ranges are not exceeded.

If the HyperTerminal program is running on the PC, the serial communication cable has been installed, the proper com port is selected, and the proper voltage is supplied, the screen will look like Figure 1–3.

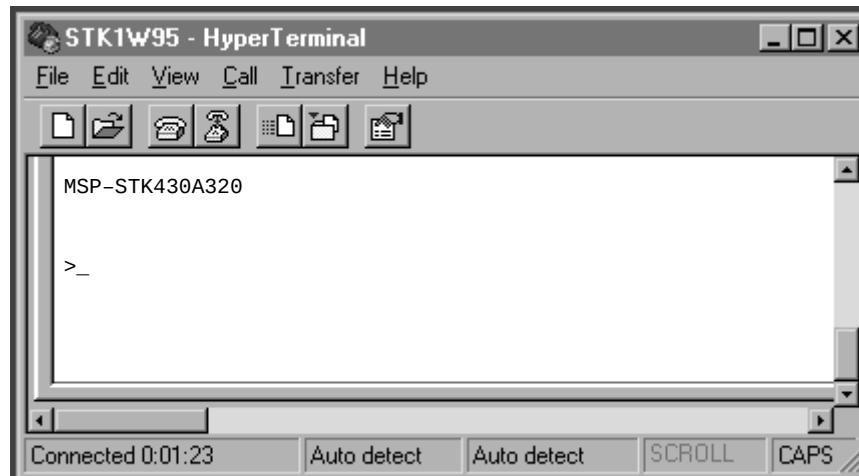


Figure 1–3. Terminal Screen

A help message is displayed on the screen automatically if the Monitor message MSP-STK430x320 is not received by the STK/EVK control software, where x is the current revision letter.

Reset Button

NOTE: Pins for the STK Demo Program

Some pins of the MSP430 device will be used for the sensor demo program (only available on the STK):

- A5 (connected to the sensor)
- Seg0. . .Seg13 (connected to the LCD Display)
- Com1. . .Com4 (connected to the LCD Display)
- R03. . .R33 (connected to the LCD R-Ladder)
- P0.0 (connected to the Trigger button)
- The string MSP-STK430x320 should appear in the HyperTerminal window (white background area of the STKW95 HyperTerminal™ window). In addition, the LCD display on the evaluation board (EVK or STK) should display MSP430.
- If both do not appear, simply press the reset button on the evaluation board (identified with an RES etched next to it) one time.

NOTE:

The MSP430 evaluation board requires very low supply voltages due to its ultra-low power consumption characteristics. In this case, the RS232 interface, or a 3.6 V lithium battery (if installed) provides the supply voltage for the STK/EVK.

Due to PC com port differences, it may be necessary to jumper the board down to 3 V or supply an external battery. To jumper the board, refer to the diagram in the STK/EVK Target Connectors section, and place one jumper across the Vcc holes, and another across the Vcc/LCD holes where the default for both of these is open.

HyperTerminal is a trademark of Hilgraeve.

Changing the RS232 interface to another COM port (other than COM2) of a personal computer, requires a change to the COM port assignment in the current version of the terminal emulator.

1.2.3 Programming the Monitor Software Into an Erased EPROM (EVK Only)

The following steps are recommended to program the Monitor into an EPROM after it has been erased:

1. Connect the EVK-PCB to the programming adapter.
2. Connect the programming adapter to the PC.
3. Apply the power supply to the programming adapter (not included in the kit).
4. Start the Program Device software and:
 - select the file mon_140.txt in the STK directory
 - select *with Verify*
 - select EPROM device
 - select the correct parallel port where the adapter is connected
 - click on the Program button.

For detailed information see the *MSP430 Family Programming Adapter Manual*.

Figure 1–4 shows the pop-up that will appear when programming the Monitor software into an erased EPROM EVK.

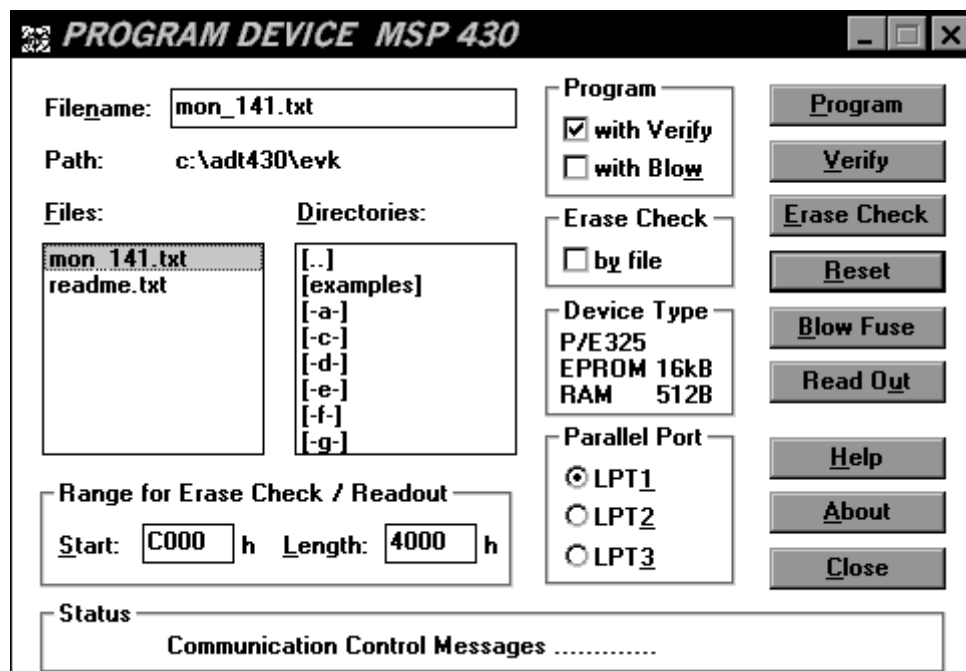


Figure 1–4. Programming the Device

1.2.4 MSP-STK/EVK430x320 Target Connectors

External Supply 3–5V dc
Only Required If Stand
Alone Operation

Pin 1	A4
Pin 2	A5
Pin 3	A2
Pin 4	A3
Pin 5	SVCC
Pin 6	REXT
Pin 7	A0
Pin 8	A1

Digital Signals:

Pin 1	XIN	Pin 9	NC	Pin 17	PO.0
Pin 2	XOUT	Pin 10	CI	Pin 18	PO.1
Pin 3	XBUF	Pin 11	TP.0	Pin 19	PO.2
Pin 4	RST/NMI	Pin 12	TP.1	Pin 20	PO.3
Pin 5	TCK	Pin 13	TP.2	Pin 21	PO.4
Pin 6	TMS	Pin 14	TP.3	Pin 22	PO.5
Pin 7	TDI	Pin 15	TP.4	Pin 23	PO.6
Pin 8	TDO	Pin 16	TP.5	Pin 24	PO.7

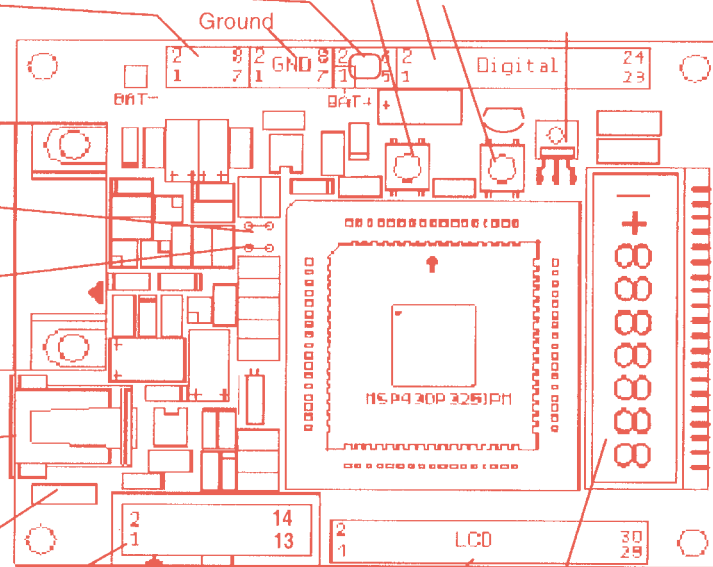
VCC		
3 V	Close	
5 V	Open	
VCC LCD		
5 V 3 V	Open	
5 V 5 V	Close	
3 V 3 V	Close	
Default		

Vpp-Input
Apply 12-20 V dc If
Programming Using Serial
Port



Vpp Fuse F250 mA

Connector To Programming
Adapter MSP-PRG430x
(Only EVK)



LCD Signals: LCD (Only STK)

Pin 1	R33	Pin 11	SEG6	Pin 21	SEG16
Pin 2	R23	Pin 12	SEG7	Pin 22	SEG17
Pin 3	R13	Pin 13	SEG8	Pin 23	SEG18
Pin 4	R03	Pin 14	SEG9	Pin 24	SEG19
Pin 5	SEG0	Pin 15	SEG10	Pin 25	SEG20
Pin 6	SEG1	Pin 16	SEG11	Pin 26	COM0
Pin 7	SEG2	Pin 17	SEG12	Pin 27	COM1
Pin 8	SEG3	Pin 18	SEG13	Pin 28	COM2
Pin 9	SEG4	Pin 19	SEG14	Pin 29	COM3
Pin 10	SEG5	Pin 20	SEG15	Pin 30	NC

Figure 1–5. MSP-STK/EVK430x320 Target Connectors

1.2.5 MSP-EVK430x330 Target Connectors

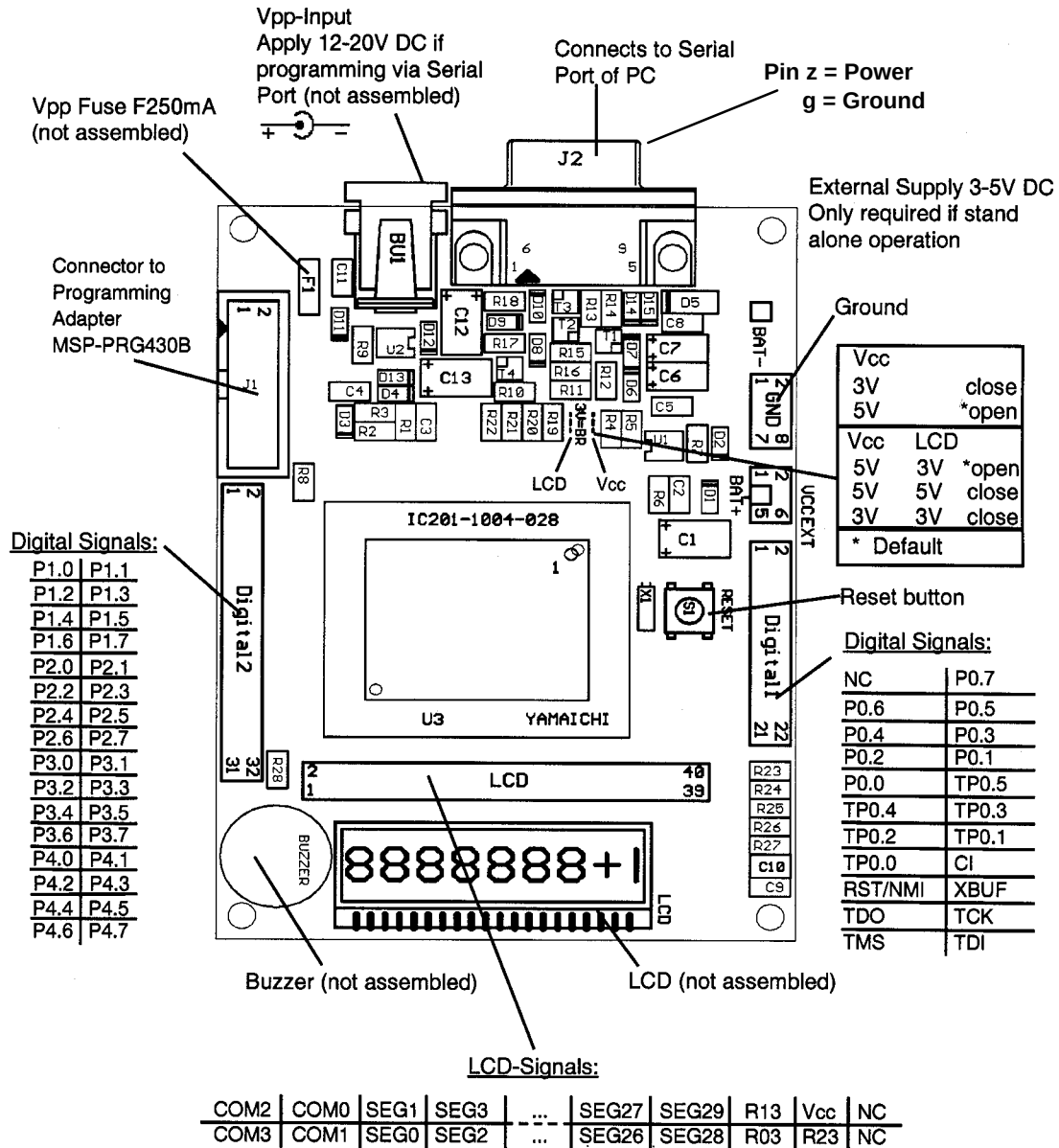


Figure 1-6. MSP-EVK430x330 Target Connectors

1.2.6 The LCD

Figure 1–7 shows the segment type LCD supplied with the STK/EVK.

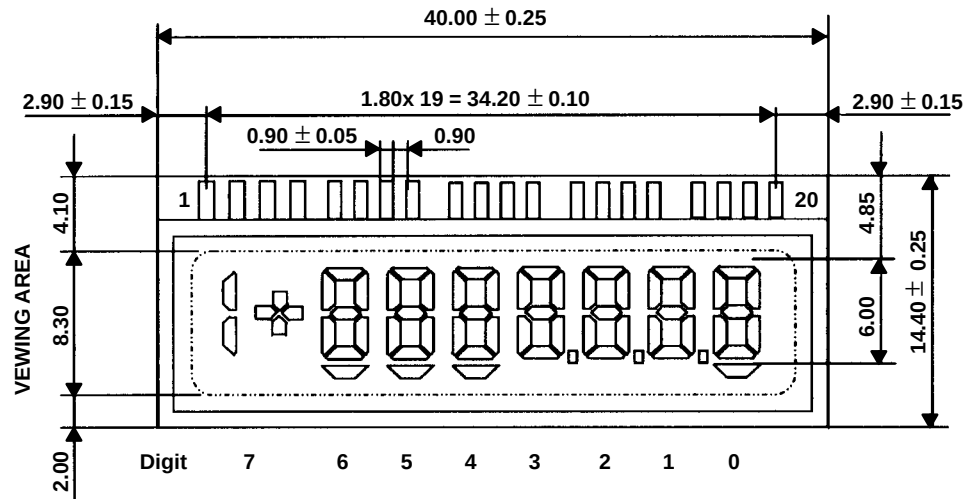


Figure 1–7. Supplied LCD Mechanical Data

The LCD is connected to the STK/EVK as shown in Table 1–1.

Table 1–1. LCD Connector

PIN NO.	COM1	COM2	COM3	COM4
1	–	–	COM3	–
2	–	–	–	COM4
3	–	COM2	–	–
4	COM1	–	–	–
5	–	–	–	–
6	6C	6F	6H	6E
7	6A	6B	6D	6G
8	5C	5F	5H	5E
9	5A	5B	5D	5G
10	4C	4F	4H	4E
11	4A	4B	4D	4G
12	3C	3F	3H	3E
13	3A	3B	3D	3G
14	2C	2F	2H	2E
15	2A	2B	2D	2G
16	1C	1F	1H	1E
17	1A	1B	1D	1G
18	0C	0F	0H	0E
19	0A	0B	0D	0G
20	COM1	–	–	–

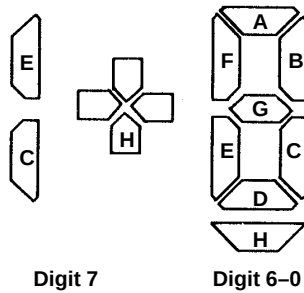
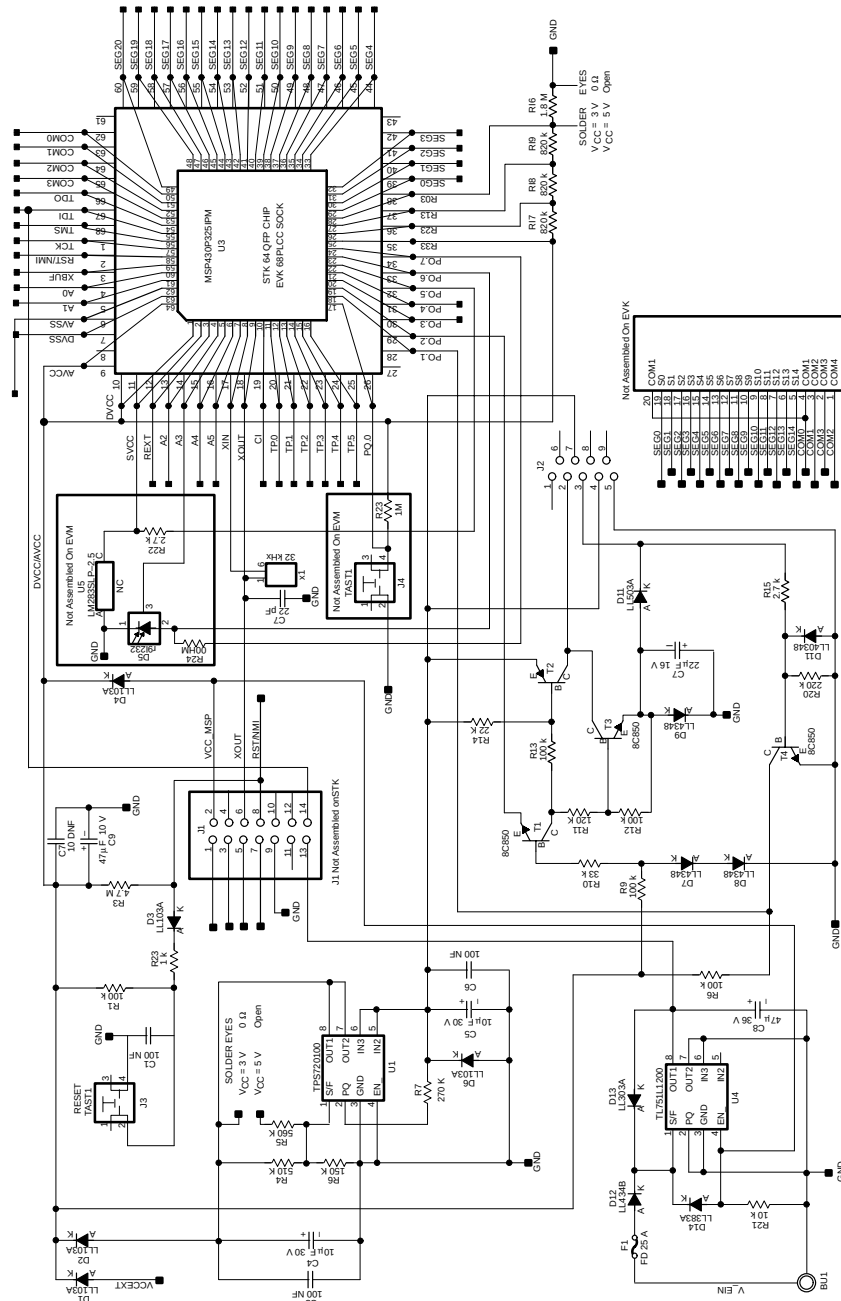
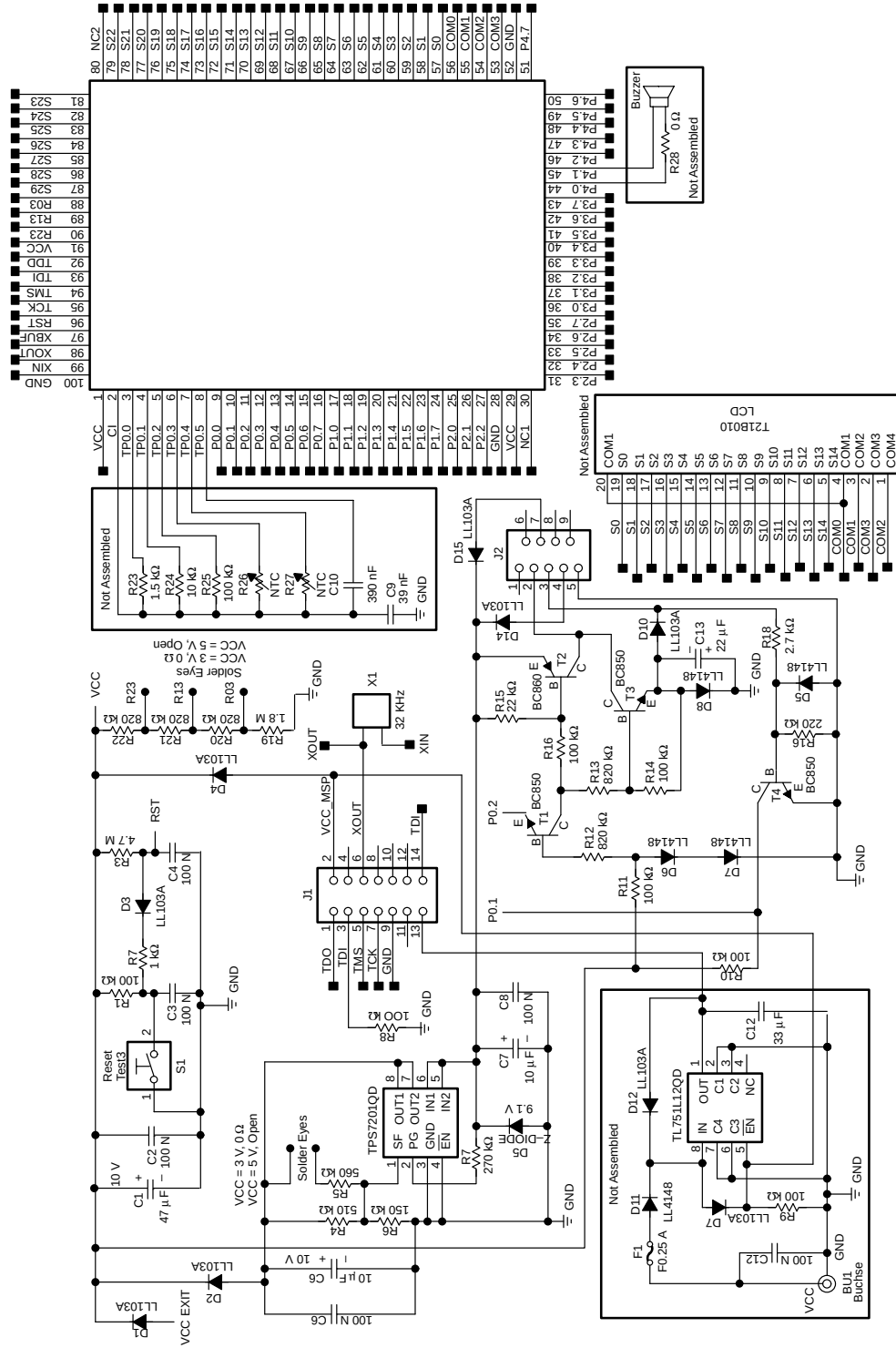


Figure 1–8. LCD Segment Digits

1.2.7 Schematic for the MSP-STK/EVK430x320



1.2.8 Schematic for the MSP-EVK430x330



1.2.9 Starting the STK Demo Program

In the HyperTerminal program, type a *d* to start the demo program. If the HyperTerminal program has been closed, double click on the Sensor Demo icon to start the demo program.



LUXMETER Demo Program:

The STK Monitor includes a demo program that shows a metering application. The application hardware consists of the light sensor, a trigger button, and a voltage reference for the analog-to-digital converter.

When the program starts, the screen displays demo and the measured light values are shown on the LCD display (measured in irradiance). The values are updated every 5 seconds. The A on the left side of the LCD indicates that the shown value is less than one second old. Any keystroke interrupts the demo program and switches back to the Monitor/HyperTerminal program. If the STK is disconnected with the demo running, the program will run until a mounted battery is discharged. To correctly exit the program push the <ESC> key.

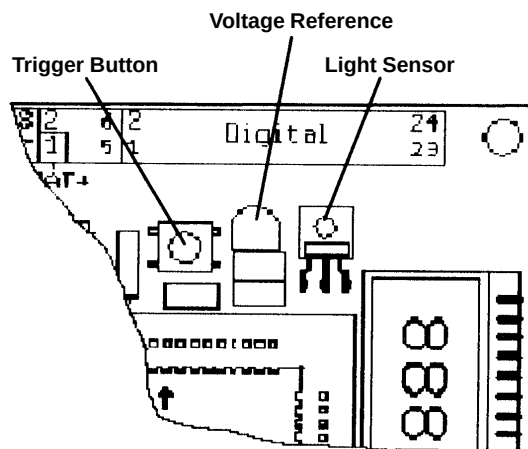


Figure 1-9. LUXMETER Demo Program

1.2.10 Executing a Program with the STK

To execute a program, write a test program or use the *gs_stk1.asm* file, which is located in the *C:\ADT430\STK\EXAMPLES* directory. Figure 1-11 is the Getting Started demo program.

NOTE: Programming Hint

During development, a program should reside in the free RAM area (240h-3FFh for the MSP430x320, and 272h-50Ch for the MSP430x330) making program modification during debug phase possible. Any code written to the device's EPROM area on the STK cannot be erased. The stack pointer for the MSP430x33x device resides at 005DEh.

How to use the command line parameters of the assembler:

The default command line parameters of the assembler (*asm430.exe*) are *-z* and *-I*. These parameters are set in the Properties of the Assembler icon after installation. For further information on command line parameters see the *Assembly Language Tools User's Guide*.

To change the parameters, select the ASM430 Assembler icon with one left mouse click. Click on the file menu in the Windows Program Manager and select the Properties command. Select the shortcut tab. Depending on the version of Windows being used, the screen will show the following dialog on the screen:



Figure 1-10. Properties

NOTE: File | Properties...

If the installation is not changed, the parameters *-z* and *-I* will be added to the target's statement. If the *Start in:* properties are adjusted to the source code directory, the path of the source file should not be entered.

```

;*****
;Getting Started 1 Demo Program                (C) TEXAS INSTRUMENTS on 2/96
;*****
SIM          .set    0                ; 1 = Simulator
                                           ; 0 = STK/EVK
RAM_orig     .set    00240h           ; Free Memory startaddress
SP_orig      .set    003DEh           ; stackpointer

;--- Control register definitions
IE1          .equ    0h
IE2          .equ    01h
IFG1         .equ    02h
IFG2         .equ    03h
ME1          .equ    04h
ME2          .equ    05h

WDTCTL       .equ    0120h
WDTHold      .equ    80h
WDT_wrkey    .equ    05A00h

GIE          .equ    08h

;*****
;Reset : Initialize processor
;*****
                .sect  "MAIN" ,RAM_orig
RESET
MOV    #SP_orig,SP                ; initialize stackpointer
MOV    #(WDTHold+WDT_wrkey), &WDTCL ; Stop Watchdog Timer

;--- Clear Special Function Registers
MOV.B  #08h, IE1                  ; ! Monitor !
CLR.B  IE2
CLR.B  IFG1
CLR.B  IFG2

                JMP    $                ; Endless Loop
File gs_stk1.asm

```

Figure 1–11. Getting Started Demo Program

Use the assembler icon in the ADT430 Program Folder to assemble the *gs_stk1.asm* file, which is located in the *C:\ADT430\STK\EXAMPLES* directory, and is shown in Figure 1–11. Ensure that the *Start in:* property of the assemble icon is set to the source directory (*C:\ADT430\STK\EXAMPLES*). Click on the ASM430 Assembler icon.



The following screen will appear (the version number, date, and copyright date may differ from that shown below):

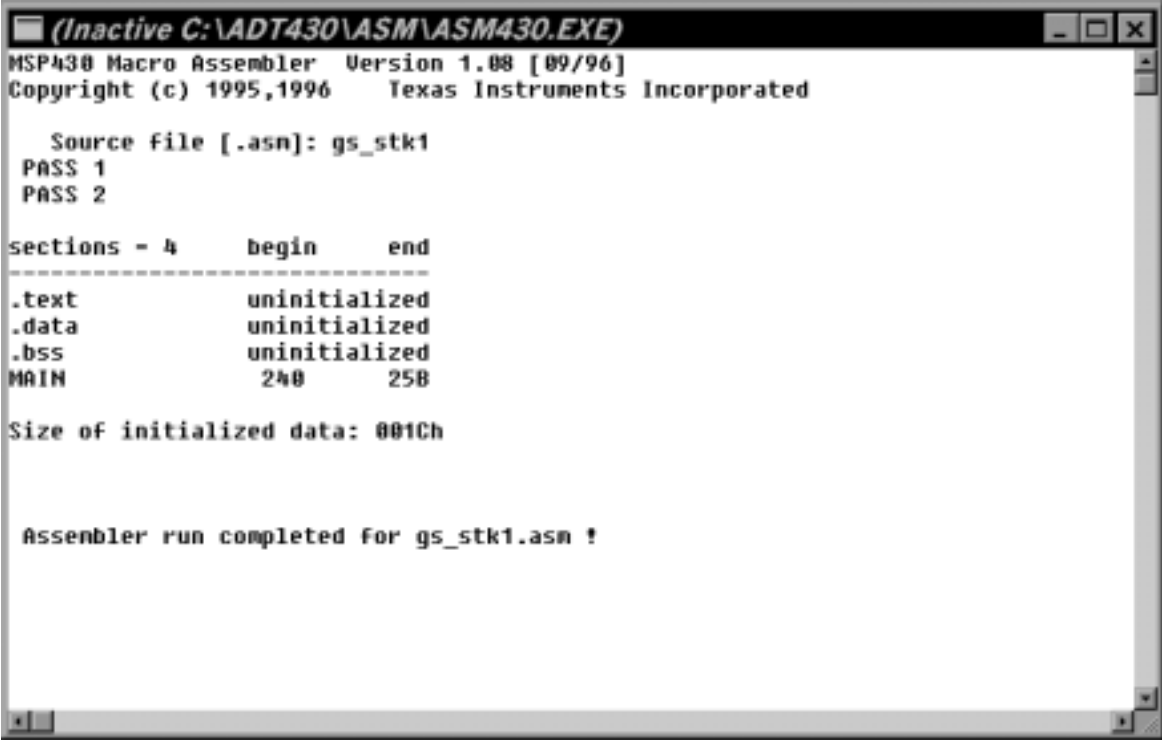


Figure 1–12. ASM430 Assembler Window

Enter the source name *gs_stk1* and press ENTER. This will cause the assembler to assemble *gs_stk1.asm* into *gs_stk1.txt*. The following display will appear:

File

Source



```
(Inactive C:\ADT430\ASM\ASM430.EXE)
MSP430 Macro Assembler Version 1.08 [09/96]
Copyright (c) 1995,1996 Texas Instruments Incorporated

Source file [.asm]: gs_stk1
PASS 1
PASS 2

sections - 4      begin      end
-----
.text             uninitialized
.data             uninitialized
.bss              uninitialized
MAIN              240        258

Size of initialized data: 001Ch

Assembler run completed for gs_stk1.asm !
```

Figure 1–13. *gs_stk1.asm* Window Display

The assembler output file is named *gs_stk1.txt*. This file is created only if the assembler is invoked with the option -z. To verify that the *gs_stk1.txt* file was created, check the *ADT430\STK\EXAMPLES* directory.

An additional listing of *gs_stk1.lst* is created when the option -l is used. To review the *gs_stk.lst* output, double-click on the file name and view using an appropriate viewer. Figure 1–14 is a listing of *gs_stk1.asm*:

```

1;*****
2;Getting Started 1 Demo Program                      (C) TEXAS INSTRUMENTS on 2/96
3;*****
4
5;*** Set this variable to '1' for the use on the Simulator***
6      00          SIM          .set    0          ; 1 = Simulator
7                                     ; 0 = STK/EVK
8      0240        RAM_orig     .set    00240h      ; Free Memory startadress
9      03de        SP_orig      .set    003DEh      ; stackpointer
10
11                                     ;--- Control register definitions
12
13      00          IE1          .equ     0h
14      01          IE2          .equ     01h
15      02          IFG1         .equ     02h
16      03          IFG2         .equ     03h
17      04          ME1          .equ     04h
18      05          ME2          .equ     05h
19
20      0120        WDTCTL       .equ     0120h
21      80          WDTHold      .equ     80h
22      5a00        WDT_wrkey    .equ     05A00h
23
24      08          GIE          .equ     08h
25
26;*****
27;                                     ; Reset : Initialize processor
28;*****
29 0240                .sect "MAIN" ,RAM_orig
30 0240                RESET
31 0240 403103de        MOV     #SP_orig,SP          ; initialize stackpointer
32 0244 40b25a800120    MOV     # (WDTHold+WDT_wrkey) , &WDTCTL
33                                     ; Stop Watchdog Timer
34
35                                     ;--- Clear      Special Function Registers
36 024a 42f20000        MOV.B  #08h,IE1            ; ! Monitor !
37 024e 43c20001        CLR.B  IE2
38 0252 43c20002        CLR.B  IFG1
39 0256 43c20003        CLR.B  IFG2
40 025a +3fff          JMP     $                    ; Endless Loop
    
```

No Errors, No Warnings
 gs_stk1.lst

Figure 1-14. gs_stk1.asm

Next, the object file *gs_stk1.txt* will be downloaded to RAM of the MSP430-STK/EVK.

The PC downloads the object file *gs_stk1.txt* to RAM of the MSP430_STK/EVK. The PC uses the HyperTerminal to communicate with the Monitor Program in the EVK/STK EPROM or ROM. Windows95 (NT) provides a terminal emulator icon under *Start/Programs/Accessories*.

Double-clicking with the mouse on the STK430 terminal icon also starts the HyperTerminal. The STK430 icon is located under the ADT430 program group.

Serial communication is set up for the STK/EVK as needed. When the serial interface of the STK/EVK is connected to the COM2 port of the PC, the LCD display on the STK/EVK will show MSP430, and the Monitor in the MSP430 device will send the following string:

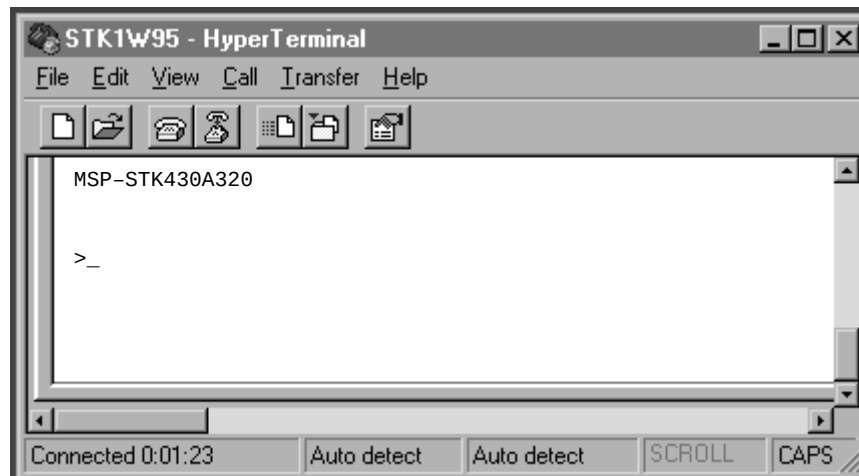


Figure 1–15. Terminal Window

Open the pulldown menu Transfer in the HyperTerminal to load the program using the RS232 into the STK/EVK, as shown in Figure 1–16. Select the Send Text File menu item.

NOTE:

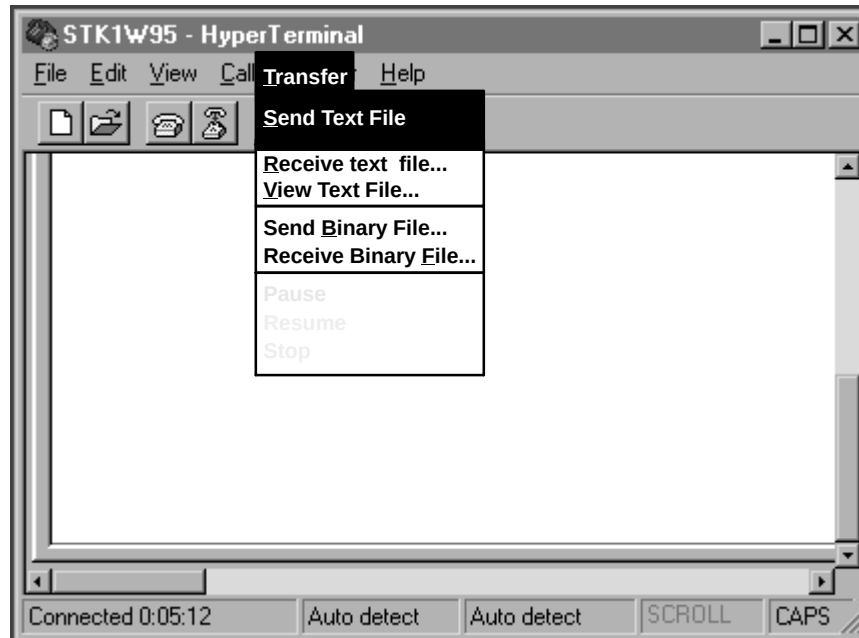


Figure 1–16. Terminal Transfer

NOTE: The help menu can be displayed by typing an *h*.

Select the assembler output file *gs_stk1.txt* in the examples directory from the Send Text File . . dialog box. Figure 1–17 shows the HyperTerminal screen displaying the following data after loading *gs_stk1.txt* into the STK/EVK.

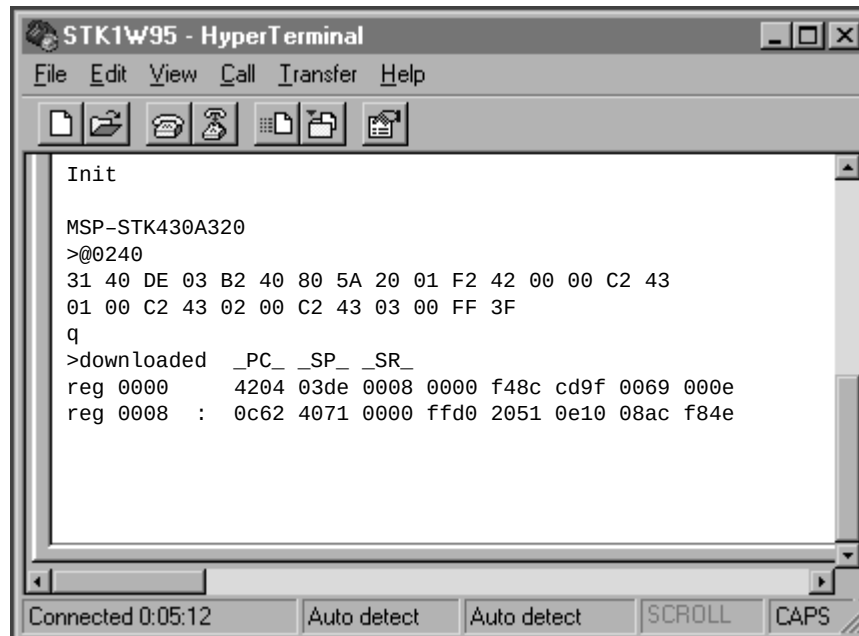


Figure 1–17. HyperTerminal Display

In the HyperTerminal, type `r0` <ENTER> to modify the program counter of the user program. The program displays the current content of `r0`. Next type `0240` <ENTER> to set the start address as defined in `gs_stk1.asm`. Exit the modify register mode by pressing the <ESC> key.

Each time the SPACE bar is pressed, the program executes one single step and the program displays the registers contents. Typing a `g` followed by <ENTER> causes the program to continue from the current address stored in the program counter. Pressing the <ESC> key stops the program.

NOTE: Single Step

The single step command should only be executed if the program counter points to an instruction in the RAM. Otherwise this command acts like a go command.

```

>@0240
31 40 DE 03 B2 40 80 5A 20 01 F2 42 00 00 C2 43
01 00 C2 43 02 00 C2 43 03 00 FF 3F
q
>download  _PC_ _SP_ _SR_
reg 0000  4204 03de 0008 0000 f48c cd85 2ed4 10de
reg 0008  0c42 4068 0000 ffd0 2051 0e10 08ac b8
>r0
=reg 0000  4204 0240
=reg 0001  03de !
  reg 000:  0008
>step
>executed  _PC_ _SP_ _SR_
reg 0000  0244 03de 0008 0000 f48c cd85 2ed4 10de
reg 0008  0c42 4068 0000 ffd0 2051 0e10 08ac b842
>go..
>user break _PC_ _SP_ _SR_
reg 0000  025a 03de 0008 0000 f48c cd85 2ed4 10de
reg 0008  0c42 4068 0000 ffd0 2051 0e10 08ac b842
>

```

Figure 1–18. Terminal | execute `gs_stk1.txt`

Another way to start this program is to write the start address (in this instance `0240h`) into memory location `03FEh`. Each time the `u` user command is executed or the reset button is pressed, the start address (in this instance `0240h`) is written to the user program counter. Switch to the word mode by entering a `w`, and then enter `m3fe` and <ENTER>. Now the start address can be entered (in this instance `0240`), and press <ENTER> and <ESC> to exit. Pressing `u` and then `g` starts the program.

NOTE: Use of the examples in MSP430 Simulation Environment

If the examples are used in the MSP430 Simulation Environment, the SIM constant (located at the top of the assembler code) must be set to 1.

1.2.11 How to Use Breakpoints

Continue the Getting Started exercise by assembling and downloading the file *gs_stk2.asm* in the examples directory as described previously.

This program includes a section for the reset vector, so the program counter is initialized after the download and each time a user performs a reset.

```
*****  
; Interrupt vectors  
*****  
  
    .sect      "Int_Vect", USER_END-1  
    .word      RESET      ; POR, ext. Reset, Watchdog  
    .end
```

1. Start the program by typing *g* <ENTER>. The display indicates 0, then 1 on the LCD.
2. Press the <ESC> key to stop the program.
3. Type *u* <ENTER> to execute a user reset.
4. Set the breakpoint by typing *s*, and then *0264* (address) <ENTER>. This breakpoint stops the program prior to changing the display.
5. Type *s* <ENTER> to stop the program. It stops prior to changing the display. Step through the program by pressing the space bar showing how the display changes.
6. Clear the breakpoint by typing *c*, and then *0264* (address) <ENTER>.

Figure 1–19 shows these commands as they appear on the display.

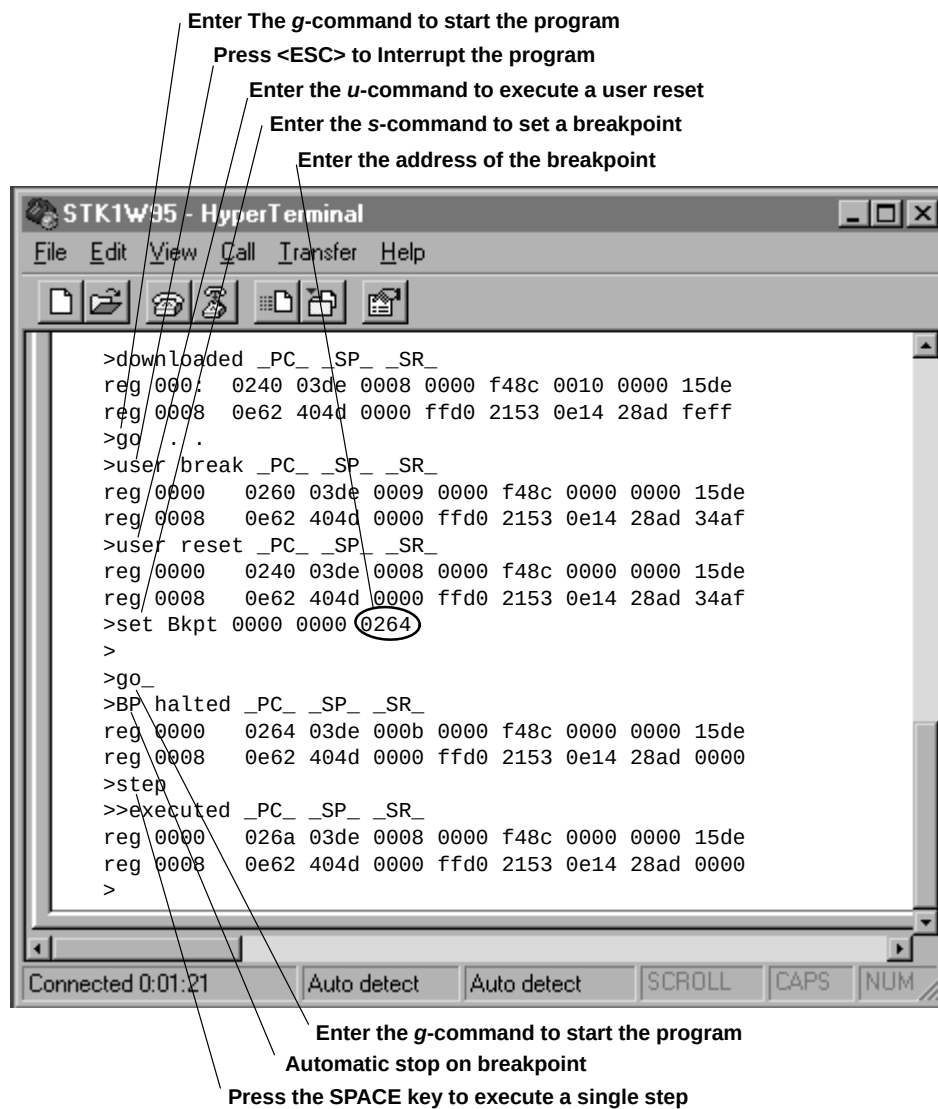


Figure 1–19. Terminal Breakpoint

1.2.12 Accessing the Port on the MSP430x320 STK/EVK

Assemble and download the file *gs_stk3.asm* in the examples directory, as described previously.

Press the trigger button on the demo to start the program. Press the trigger button to toggle the display between 0 and 1.

The following instructions are necessary:

```

;--- Init Port
        BIC.B    #01h,P0DIR    ; Set P0.0 as input
        BIC.B    #01h,P0IE    ; Disable P0.0 interrupt

;--- Test of Port

```

```

$ML1      BIT.B      #01h,P0IN      ; Test P0.0
          JNZ        $ML1           ; Do nothing if P0.0 low

```

1.2.13 How to Use an Interrupt Routine

Assemble and download the file *gs_stk4.asm* in the examples directory, as described previously.

1. When an interrupt occurs, the program sends the contents of the program counter and status register to the stack.
2. Next, the program branches to the starting address of the interrupt routine.
3. The interrupt routine normally ends with the RETI instruction.
4. The RETI instruction loads the data, which was saved to the stack at the beginning of the interrupt routine, to the status register and program counter.
5. The program continues from the point of interruption.

Using interrupt routines allows the MSP430 to use low power modes. Selecting a low power mode causes the program to stop at the current position, and an activity that causes an interrupt automatically clears the low power mode. The interrupt continues program execution at the starting address of the corresponding interrupt routine. The RETI loads the saved program data to the status register and program counter. Loading the status register and program counter clears out the low power mode bits, and the program continues with the next instruction. Figure 1–20 is the Basic Timer Interrupt routine.

```

          BIS        #CPUOFF,SR      ; set CPUoff Bit
;*****
; Basic Timer Interrupt routine
;*****
Int_BT    ; Basic Timer 128 Hz (7.8 ms)
          DEC.B      lcd_timer       ; decrement SW lcd-timer
          JNZ        Int_BT_end      ; !0 : no action
          BIC        #CPUOFF,0(SP)   ; Clear CPUoff Bit
          MOV.B      #lcd+ival,lcd_timer ; = 0 : load again
Int_BT_end
          RETI

```

To use the interrupts, the interrupt vector table must be set up as follows:

```

;*****
; Interrupt vectors
;*****
          .sect "int_Vect", USER_END-31
          .word      RESET           ; no source
          .word      Int_BT          ; Basic Timer
          .word      RESET           ; no source
          .word      RESET           ; no source
          .word      RESET           ; no source
          .word      RESET           ; no source
          .word      RESET           ; no source
          .word      RESET           ; no source
          .word      RESET           ; no source
          .word      RESET           ; no source
          .word      RESET           ; no source
          .word      RESET           ; no source
          .word      RESET           ; NMI, Osc. fault
          .word      RESET           ; POR, ext. Reset, Watchdog
          .end

```

Figure 1–20. Basic Timer Interrupt Routine

Set a breakpoint at address 025Eh, and single step through the program. A longer reaction time can be expected for any step beyond address 025Eh. At this time, the CPU is off and waiting for the Basic Timer interrupt.

NOTE: Only two breakpoints may be set at any one time.

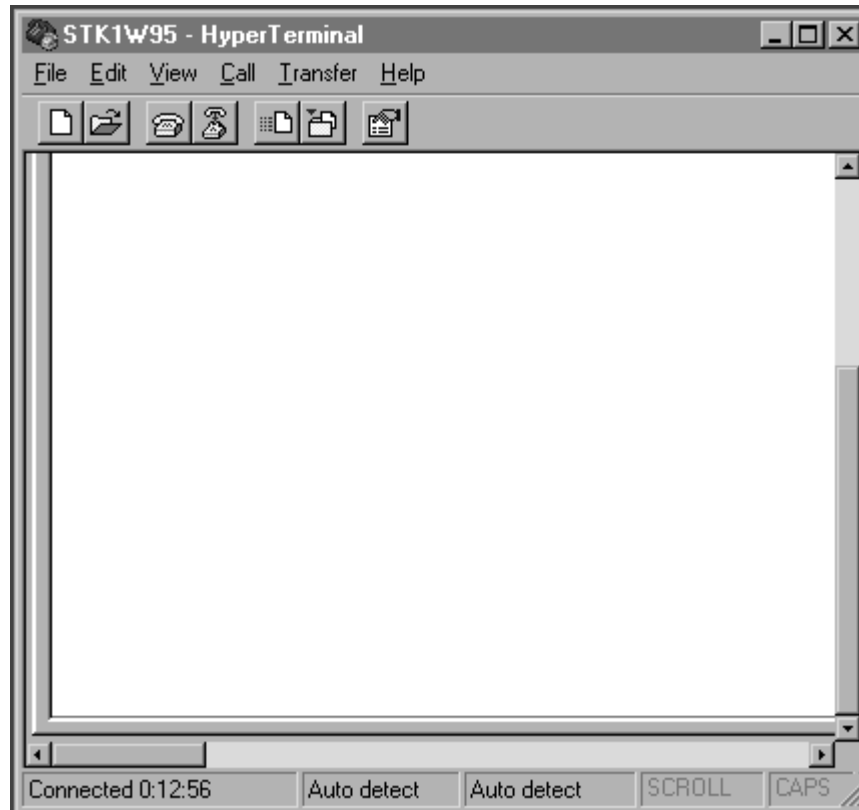


Figure 1–21. Terminal | Interrupt

1.3 Loading a Program Into the EPROM Via the Terminal

After debugging, the program is ready to load. The program is loaded to the EPROM for the EVK, or ROM for the STK. The STK requires a programming voltage. The EVK requires a jumper between pins 13 and 14 of the programming connector. Figure 1–22 show the locations for the programming voltage and the connector jumper.

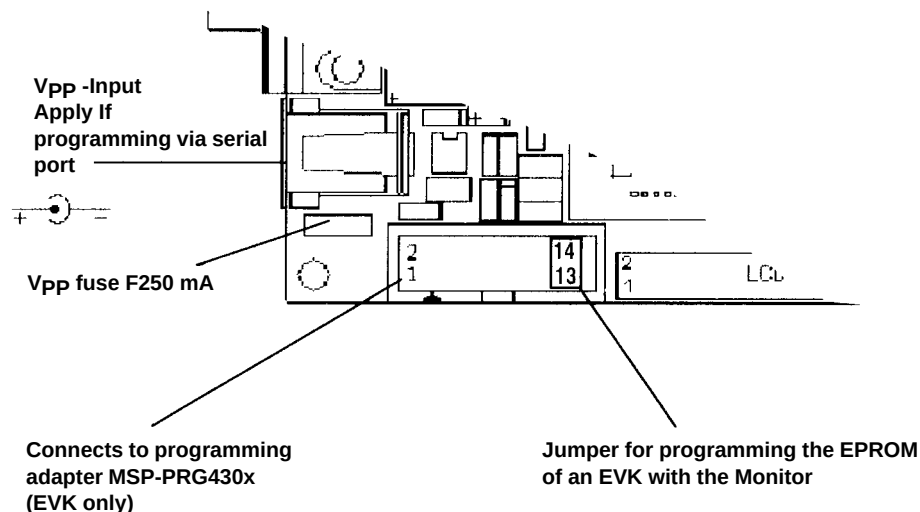


Figure 1–22. Programming Voltage and Jumper Location

To load a program into the MSP430 EPROM/ROM using the HyperTerminal, the following points must be considered:

- The Monitor code must already be programmed into the EPROM/ROM (only the EVK is supplied with the code for the monitor program *mon_x.txt*).
- The STK/EVK 9-pin D-SUB-connector must be connected to the serial interface (the default is COM2) of the PC.
- The supply voltages of the STK/EVK must be applied by connecting the STK/EVK to the serial interface of the PC and by connecting the programming supply voltage.

NOTE: Programming the EVK using the Serial Port

Attach a jumper between PIN 13 and PIN 14 on the Programming Adapter connector (see *Hardware Installation* in Chapter 9).

If the assembled file *gs_stk5.asm* is downloaded, the program routines for the LCD, and the interrupt routine of the previous example will be written into the EPROM/ROM (see sections in the code).


```

;*****
; Section in EPROM
;*****
.sect "PrepLCD", EPROM_orig
;--- Prepare LCD and Basic timer
PrepLCD MOV.B    #-1h,LCDM          ;LCD : Analog generator on Low
                                           ; impedance 4 Mux active
                                           ; all outputs are Seg
MOV.B    #057h,BTCTL              ; Basic Timer : SSEL=0 DIV=0 Reset=1
                                           ; ACLK
                                           ; 32768/256 = 128 Hz
                                           ; (7.8 ms debounce time)
                                           ; LCD frame frequency @ 4 Mux: 64 Hz
BIS.B    #BTME,ME2                ; Enable basic timer module
BIC.B    #040h,BTCTL              ; Basic Timer reset disabled
BIS.B    #BTIE,IE2               ; enable basic timer interrupt
MOV.B    #lcd_ival,lcd_timer      ; load SW lcd timer

CALL     #show_cir                ; clear LCD

EINT                                           ; enable interrupts

;--- clear LCD
.sect "show_clr:",EPROM_orig+50h
show_clr MOV     #15, r5              ; clear display memory
show_clr1 MOV.b   #0, LCD1-1(r5)
DEC      r5
JNZ      show_clr1
RET

;*****
; Basic Timer Interrupt routine
;*****
.sect "Int_BT" ,EPROM_orig+100h
Int_BT DEC.B     lcd_timer            ; Basic Timer 128 Hz (7.8 ms)
                                           ; decrement SW lcd-timer
JNZ      Int_BT_end                ; !0 : no action
BIC      #CPUOFF,0 (SP)            ; Clear CPUoff Bit
MOV.B    #lcd_ival,lcd_timer        ; =0 : load again
Int_BT_end RETI

```

Figure 1–23. EPROM LCD and Interrupt Routine

Now these routines can be used in any other program. This section does not have to be downloaded again, even if the STK/EVK is disconnected from the power supply.

To load a program using the HyperTerminal, double click the terminal icon in the MSP-STK/EVK430 program group in the Windows Program Manager. The HyperTerminal window appears as follows in Figure 1–24.

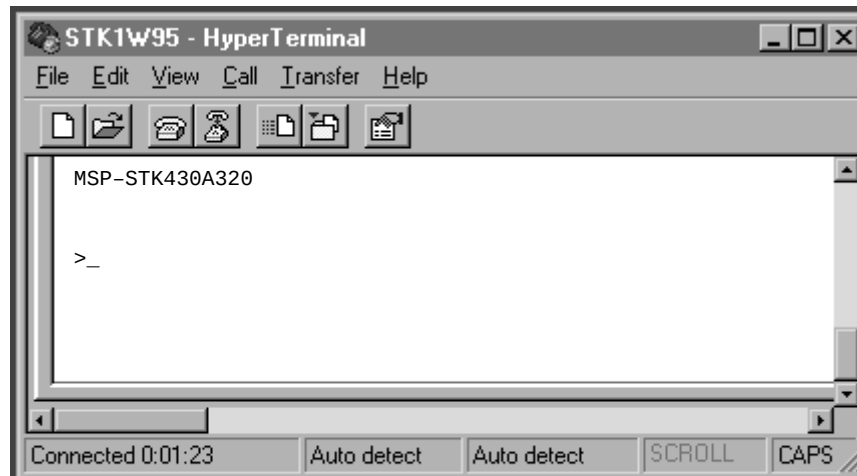


Figure 1-24. HyperTerminal Window

To load a program into the MSP430 using the HyperTerminal, activate the Transfers/Send Text File. . . command as shown in Figure 1-25.

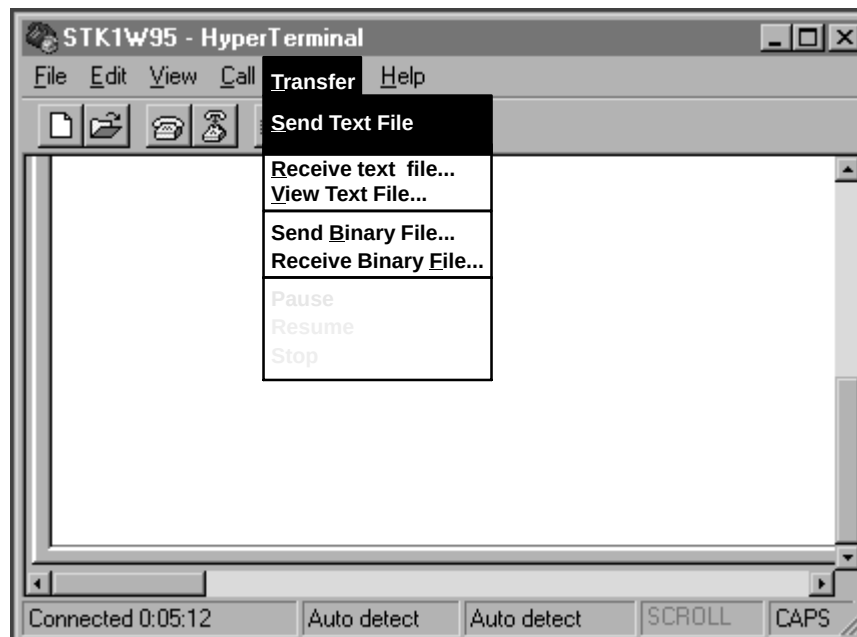


Figure 1-25. Transfers \ Send Text File

Figure 1-26 shows the Send Text File dialog box.

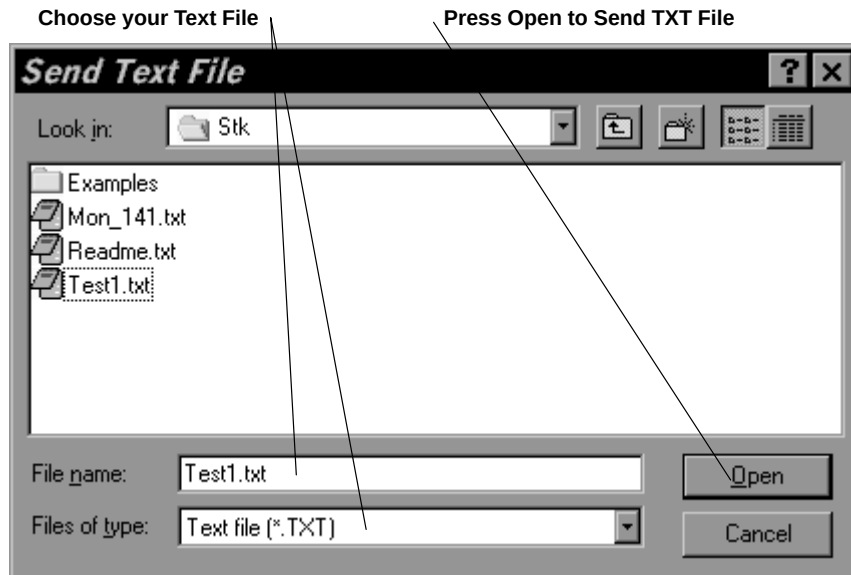


Figure 1–26. Send Text File Dialog Box

Figure 1–27 represents a burn fail screen. If an error occurs during the programming of the EPROM, the screen shows the message *burn failed at XXXX*. Where XXXX is the address where the programming cycle failed the first time.

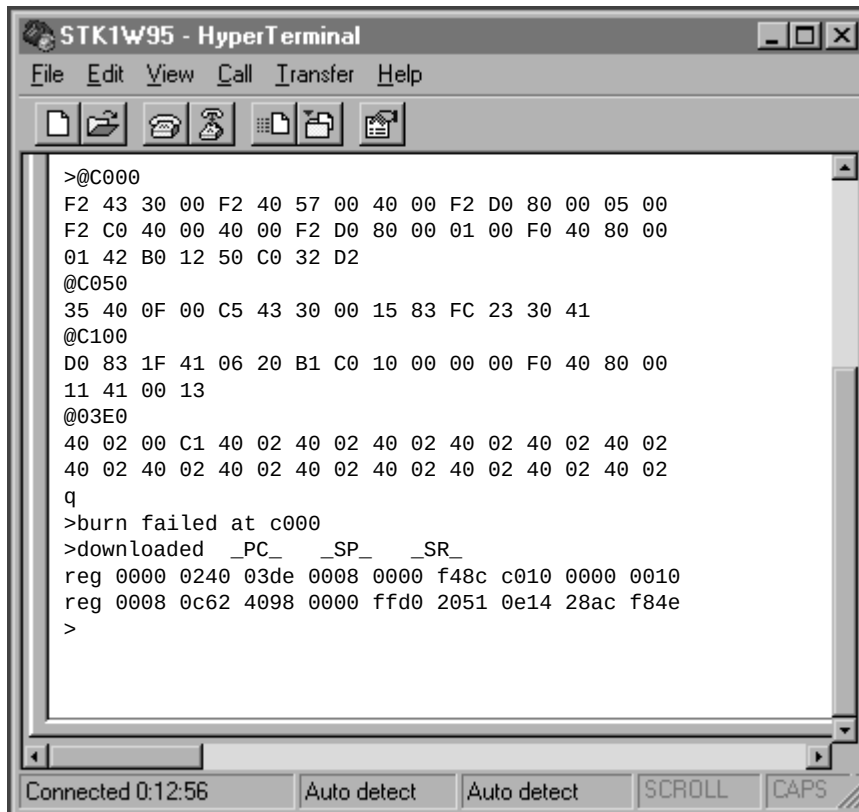


Figure 1–27. Burn Failed Message

In the *gs_stk6.asm* example, in the examples directory, only the addresses of the routines in the EPROM are labeled. This supports easy use of program sections in the EPROM.

```

;*****
;
; Addresses in EPROM
;*****

PrepLCD .equ      EPROM_orig      ; EPROM Address of PrepLCD
;--- clear LCD
show_clr .equ      EPROM_orig+50h ; EPROM Address of show_clr
; Basic Timer Interrupt routine
Int_BT .equ        EPROM_orig+100h ; Basic Timer 128 Hz (7.8 ms)

```

2 Monitor Commands

This chapter describes the syntax conventions and the available commands of the Monitor Program.

2.1 Syntax Conventions

- The numbers in brackets [] are optional.
- x is a hexadecimal address.
- n is the hexadecimal number of bytes to show.
- i is the hexadecimal register number.
- Only the *r*, *m*, and *e* commands expect <ENTER> or <ESC> commands.
- The <ESC> key provides a keyboard interrupt for all command inputs except *m*.
- The half-duplex UART prevents keyboard interrupts while entering a large number of bytes to show the results of an action.
- The registers are named in hexadecimal format from R0 to RF.
- All addresses or memory/register contents must be entered in hexadecimal mode. Entering zeros in the MSB is optional.
- Errors in address or memory/register contents in the terminal input must be corrected by re-entering all four hexadecimal digits. The leading (MSB) zeros are optional.
- The Arrow, Insert, Delete, or Backspace keys can not be used to change an incorrect terminal input.

2.2 Memory Organization

To use some commands, it is necessary to know the memory locations and their functions (see Figure 2–1). In memory locations 000h to 0FFh, only the byte mode is possible. In memory locations 100h to 1FFh, only the word mode is possible. Memory locations 010h to 0FFh are reserved for 8-bit peripheral modules, and the locations between 100h and 1FFh are reserved for 16-bit peripheral modules.

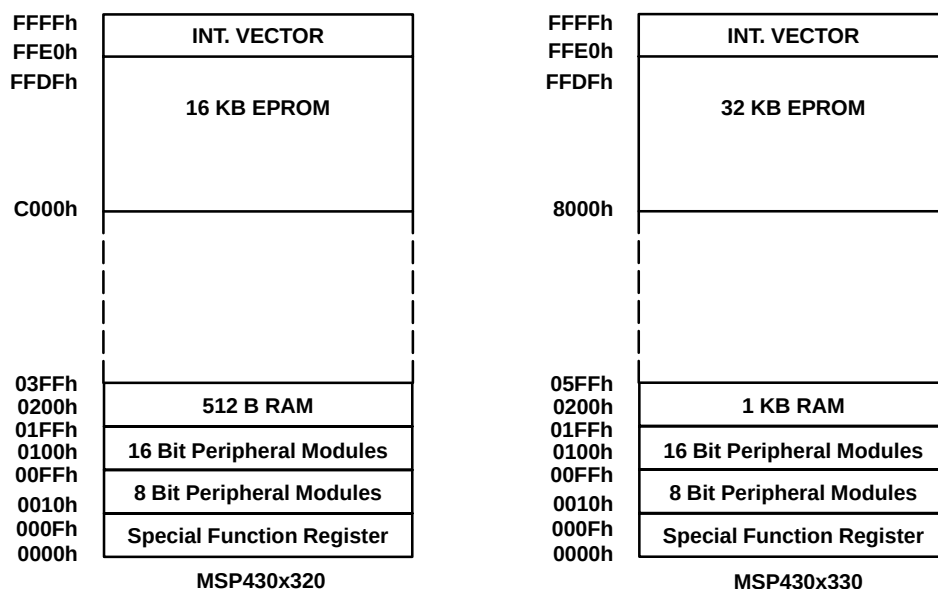


Figure 2–1. Memory Organization

2.3 Commands

- h** The *h* command displays the Help Command screen shown in Figure 2–2 with the available commands.

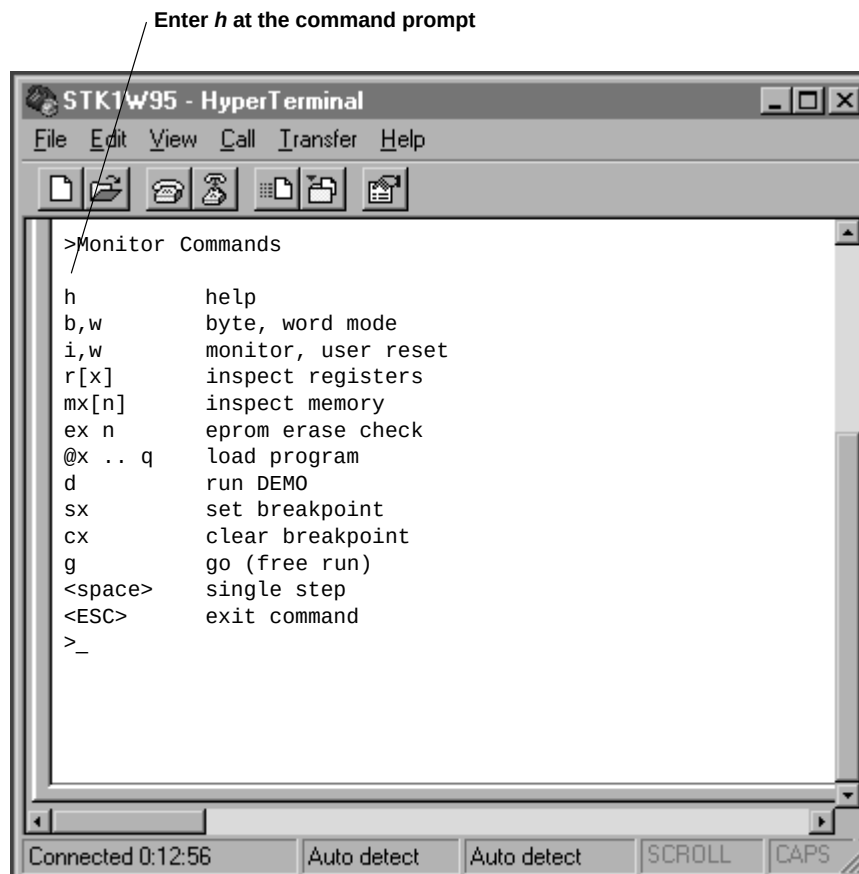


Figure 2–2. Help Command

b,w The **b** and **w** commands shown in Figure 2–3 switch to byte or word indication mode. The two indication modes are only important for the memory inspect command **m**. The Monitor completes the entered command (byte or word).

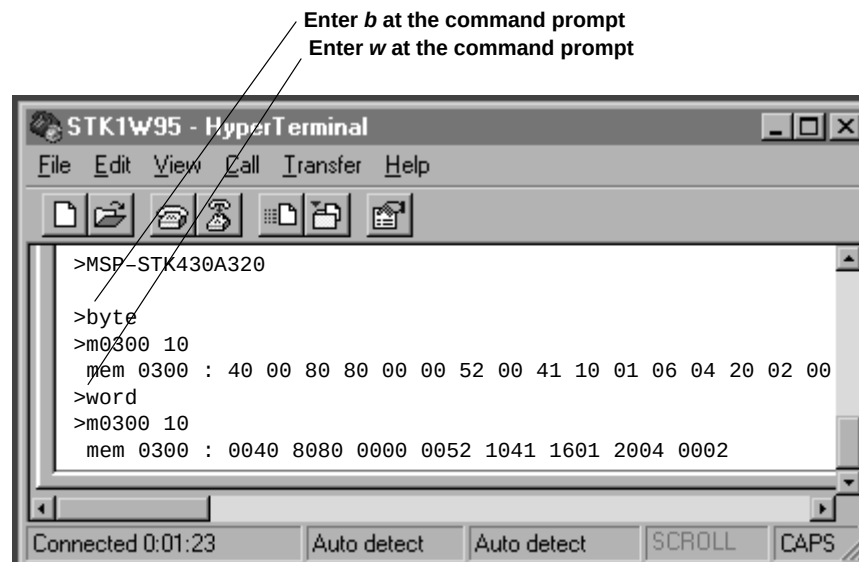


Figure 2–3. Byte, Word Commands

- i* The *i* command, shown in Figure 2–4, initializes the entire monitor program. This command performs a software reset of the STK/EVK, but is only possible if the monitor is still running. If the contents of memory location 3DEh are AA55h, the *i* command will start the user application. Turn off the hardware supply voltage to return control to the monitor. This clears the bit-pattern AA55h.

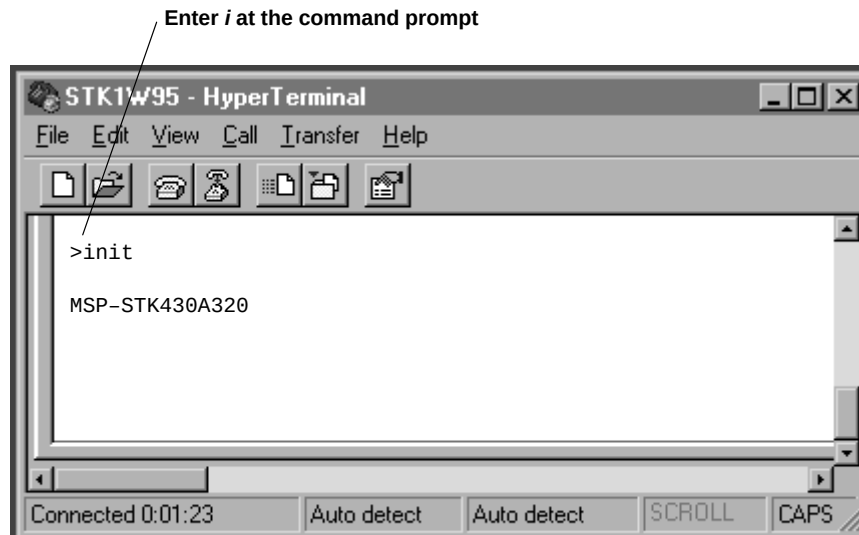


Figure 2–4. Initializing the Terminal Program Command

- u** The *u* command, shown in Figure 2–5, sets the user PC to the start vector of the loaded user program. This performs a user reset of the application. In this example, the start vector of the user application, located at address 03FEh, is 0300h.

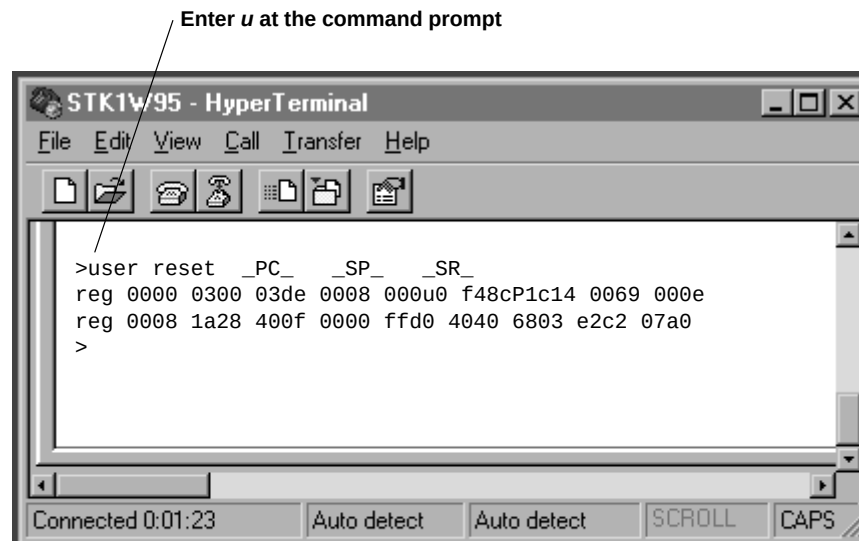


Figure 2–5. User Reset Command

- r[i]** The r <ENTER> command, shown in Figure 2–6, without a specific register number, shows all 16 CPU registers R0 to R15. With this syntax, a modification of the register contents is not possible.

Enter r then <ENTER> at the command prompt

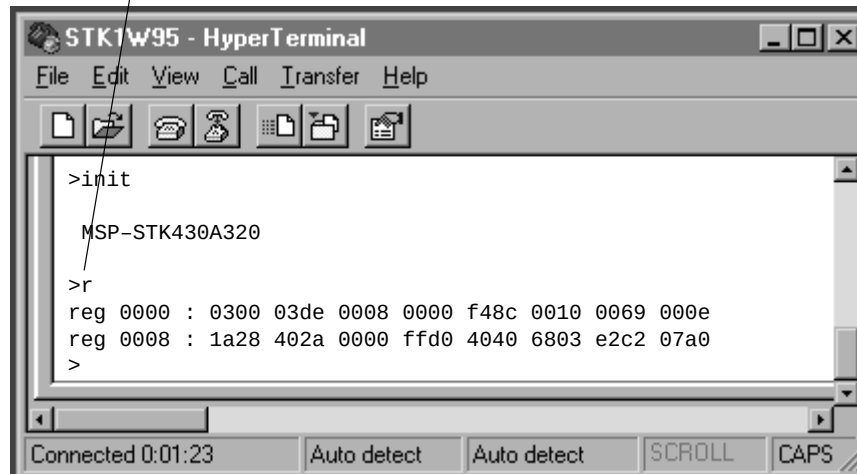


Figure 2–6. Register Command

Entering the command with the hexadecimal register number *r[i]* <ENTER> as shown in Figure 2–7, results in displaying only the contents of the dedicated register *i*.

Type *r5* <ENTER> at the command prompt

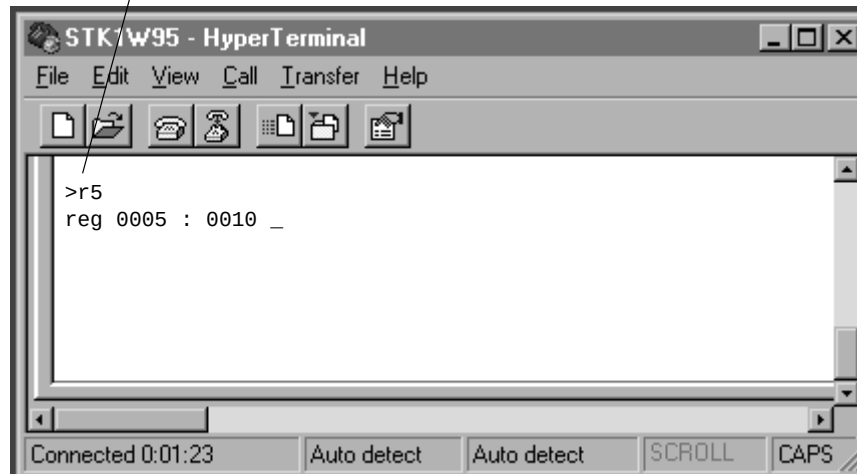


Figure 2–7. Register Specified Command

NOTE: Modification of the Registers R1, R3, and R4

The Monitor Program does not allow the modification of registers R1, R3, and R4. R1 is the stack pointer and can not be modified because of the internal program structure of the monitor. R3 is the constant generator register and therefore cannot be modified. R4 is internally used by the monitor, therefore, modification of R4 by the user application code will overwrite the correct function in the monitor program and cause improper operation.

Figure 2–8 shows the procedure for changing data in a register. Write new data into register *r[i]* by pressing <ENTER> and the program displays the next register.

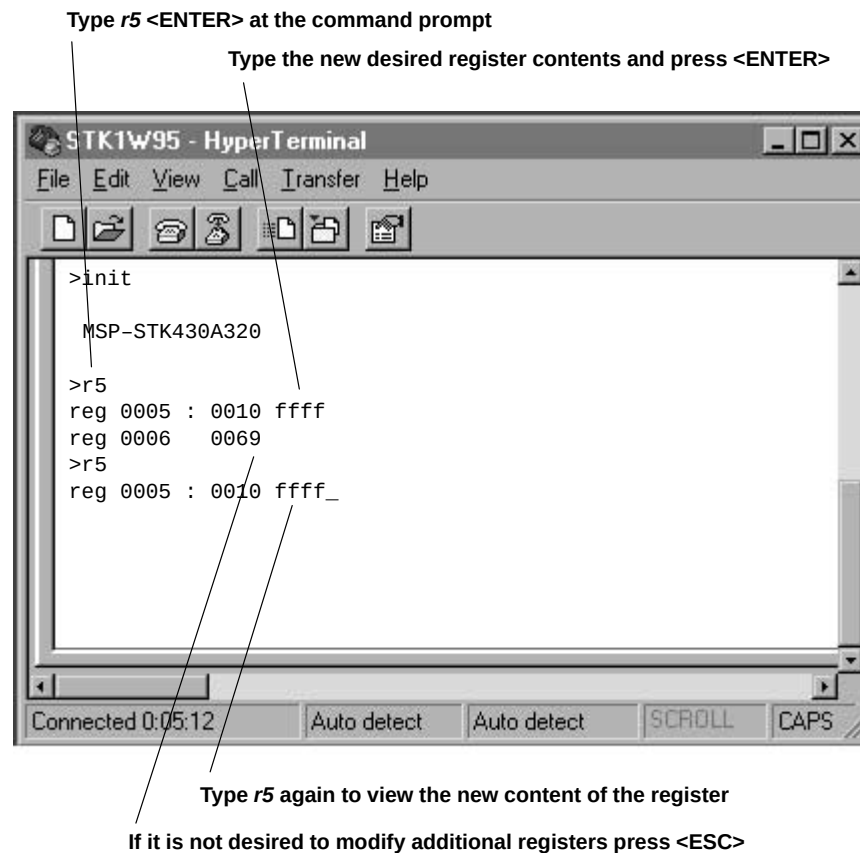


Figure 2–8. Modify One Register

Figure 2–9 shows the procedure to modify an additional register. Press <ENTER>, and the program displays the next register.

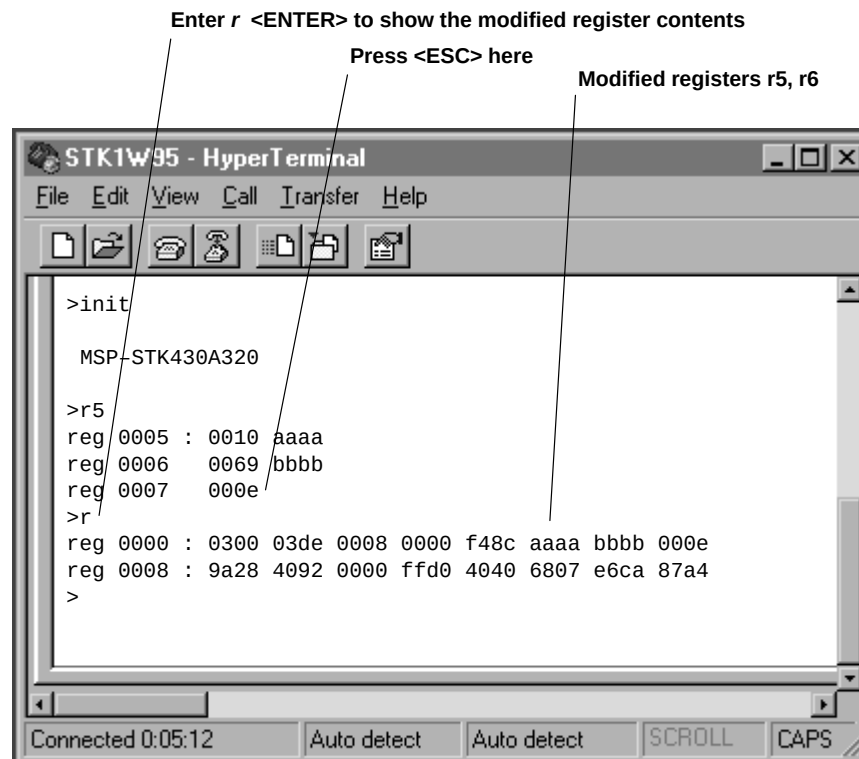


Figure 2–9. Modify Additional Registers

Figure 2–10 shows the proper way to change incorrect entries. Using the Arrow, Delete or Insert keys as inputs causes unpredictable behavior. After entering an incorrect register content, the entire input should be entered again. Another method is to press <ESC> and to enter the r[i] command again.

NOTE:

The ENTER key must never be pressed after entering an incorrect input.

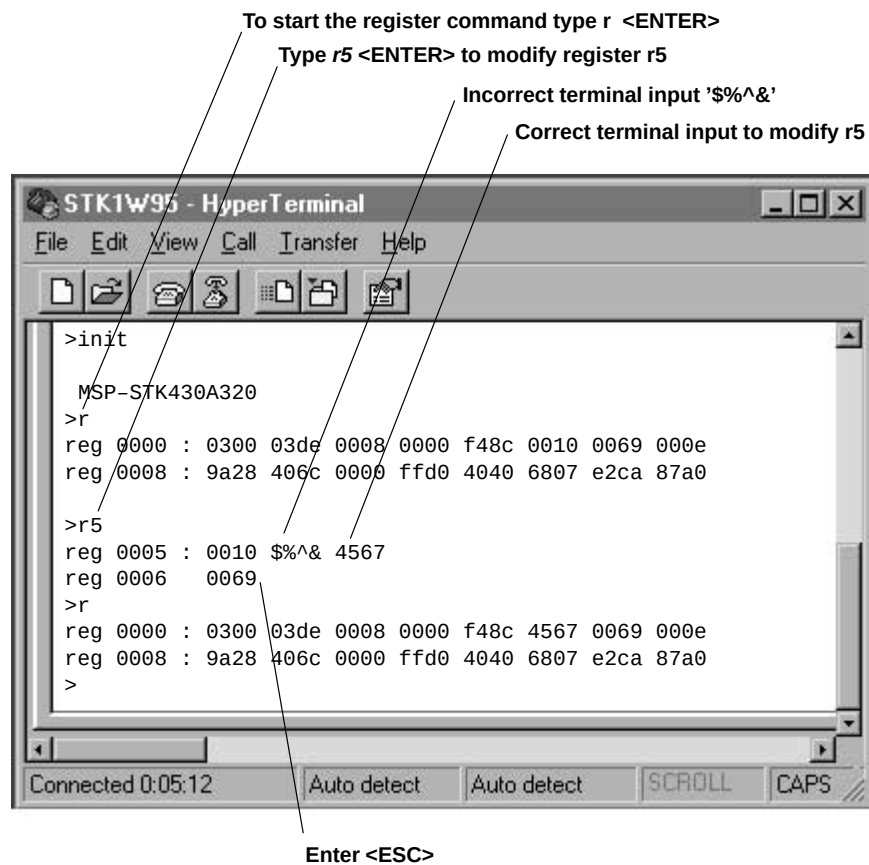


Figure 2–10. Revise Memory Modification

mx[n] The *m* command allows the user to inspect (read/modify/write) memory locations. Use the *m* command in conjunction with the *b* or *w* commands. The *b* or *w* command displays the memory as shown in Figure 2–11.

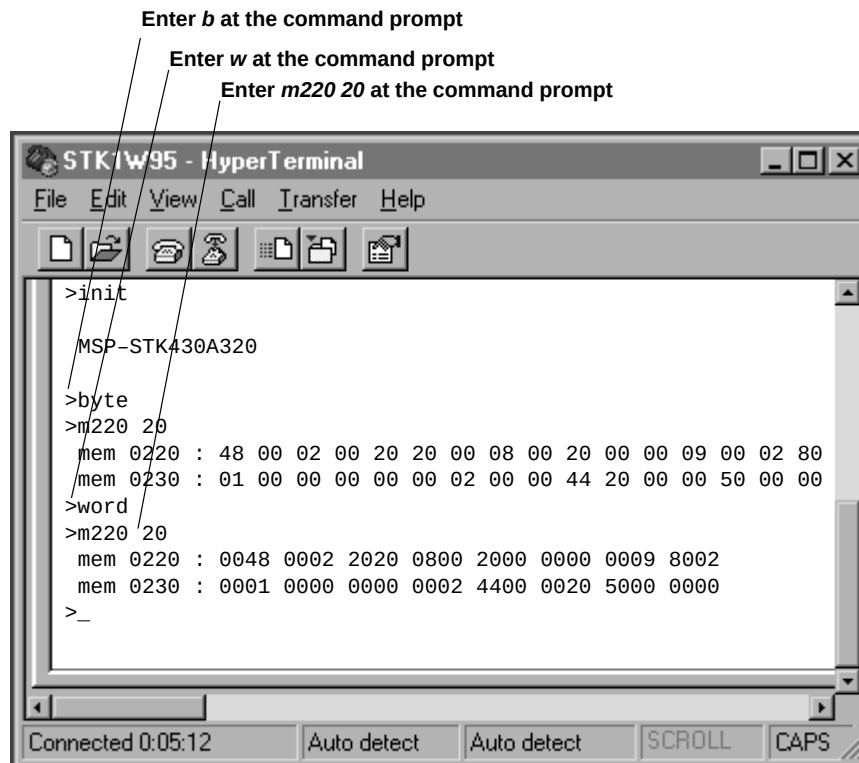


Figure 2–11. Memory Byte, Word Command

The address *x* shown above as 220 must be entered to define the memory location. The number *n*, shown above as 20, is optional. It defines the number of memory bytes that are to be displayed. Entering the *m* command without *n* allows the memory contents at address *x* to be displayed or modified.

NOTE: Interrupt.

Viewing large memory areas will take some time because the output function to display memory contents cannot be interrupted while the UART is operating in the half duplex mode.

Figure 2–12 shows how to modify memory contents. Modify the memory by typing in new data and pressing <ENTER>. The program displays the next memory location. If no modification is necessary press <ENTER>. To exit the memory command (*mx*) press the <ESC> key. Pressing <ENTER> toggles through memory locations, displaying each new location after <ENTER>.

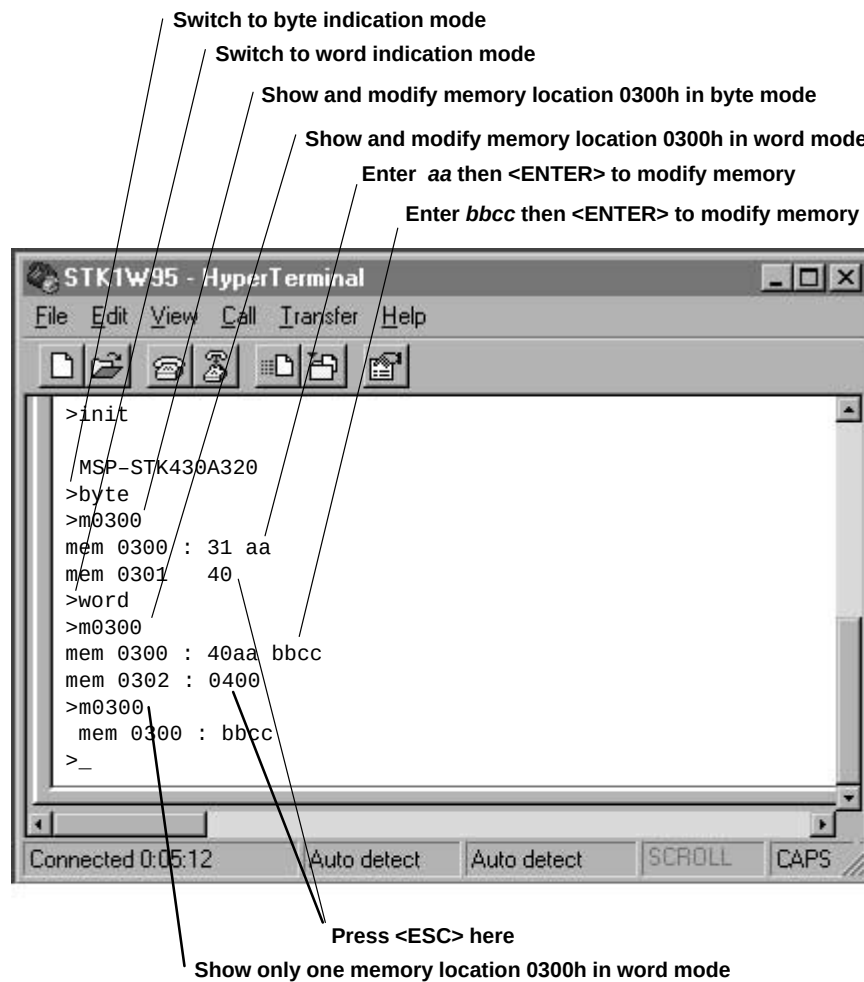


Figure 2–12. Memory Modification

To revise an incorrect terminal input, perform the following functions as shown in Figure 2-13. Using the Arrow, Delete or Insert keys as inputs causes unpredictable behavior. After entering an incorrect register contents, the entire input should be entered again. Another method is to press <ESC> and enter the *m* command again. Never type <ENTER> after entering an incorrect input.

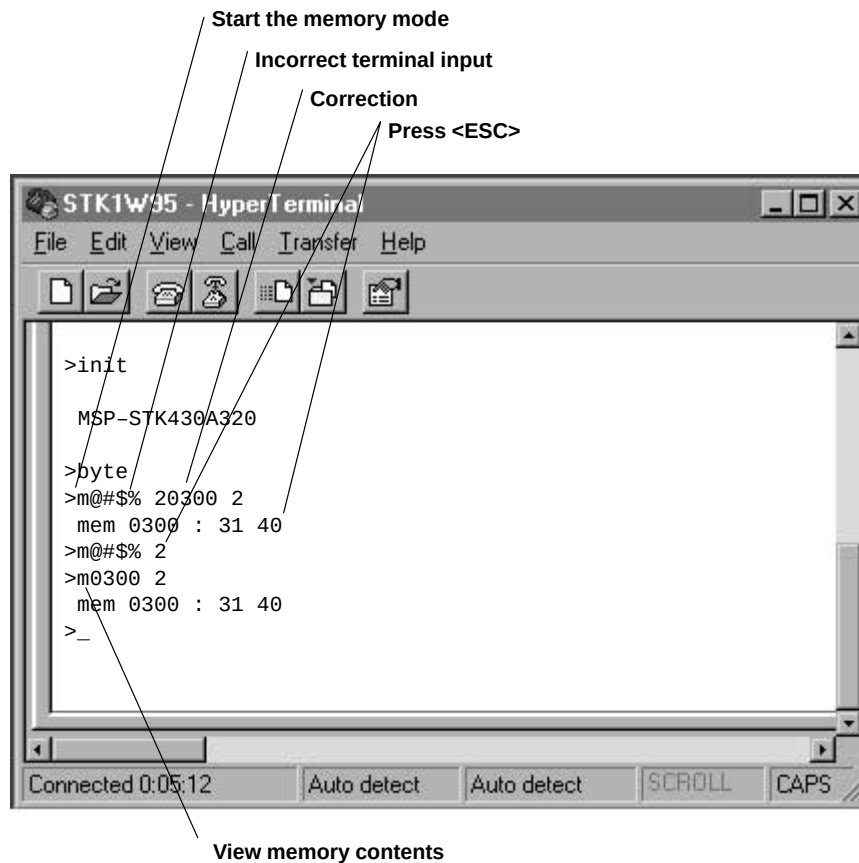


Figure 2-13. Revised Memory Modification

@x The @x command loads program/data section(s), byte by byte, into the RAM/EPROM. A program algorithm detects the correct download section (in the RAM or EPROM) and runs automatically. The transfer can be made by using a keyboard or by file transfer (pull-down menu Transfers/Send Text File in Hyperterminal program). If download is performed manually, it can be terminated with a *q* keystroke. Normally the data associated with the load command is a content of linker or assembler-generated program/data files.

Figure 2–14 is an example of the format of such a program/data file:

```

>@0220
C2 43 12 00 3B 40 B4 02 92 12 D4 FF F2 40 FF 00
32 00 B0 12 90 D8 B0 12 48 D8 B0 12 C8 D7 B0 12
72 D7 B0 12 0E D7 B0 12 56 02 B0 12 72 D7 B0 12
1C D8 30 40 52 02 06 12 F2 F0 FB 00 03 00 B2 40
1D 00 12 01 B2 40 0B 09 14 01 56 42 03 00 66 F2
FC 27 16 42 18 01 36 90 00 10 05 38 3B 40 40 DA
92 12 D4 FF 04 3C 3B 40 4C DA 92 12 D4 FF 36 41
31 40 00 03 B2 40 80 5A 20 01 30 40 20 02 06 12
52 12 12 00 52 12 11 00 F2 40 19 00 12 00 F2 42
11 00 B2 40 FF 00 12 01 16 42 10 01 F2 41 11 00
F2 41 12 00 76 F0 EB 00 B0 12 7A D7
@03FE
20 02
q
>downloaded _PC_ _SP_ _SR_
reg 0000 : 0220 0400 0008 0000 f8b8 0010 0000 0003
reg 0008 : 0000 2837 0000 f0c2 fffe fffe fffe fbfe

```

Figure 2–14. Transfer Data Command

The @x command sets the memory-modification address to the address where program/data files are stored. All code lines following the @x command represent data in byte format. These bytes will be copied sequentially to the specified section address. The *q* command terminates the section input and returns the control back to the Monitor Program.

NOTE: Programming an EPROM

While programming the EPROM, be sure that the file being used has the correct section addresses and contains the correct program/data. Programming an EPROM location once will prevent a second programming at this location in the STK. For more information see the *Hardware Installation* section in this manual.

ex n The EPROM erase check (ex n) command verifies that the EPROM is empty. The EPROM is empty if the contents of all EPROM-memory locations are FFh. The *x* represents the starting address to verify that the EPROM has been erased. The *n* portion of this command will show the number of bytes that have been erased. The Monitor ceases checking the EPROM if a memory location other than FFh is detected and displays that memory location as shown in the Figure 2–15.

Enter *mC040 20* <ENTER> to start memory mode (to show 20h)

Check if there is memory space at C040h for 20h Words

Memory location C040h is not erased and is displayed

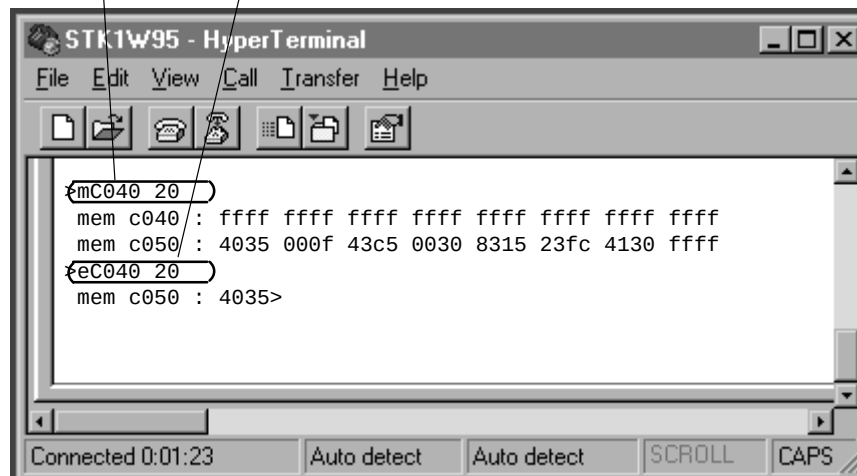


Figure 2–15. EPROM Erase Check Command

- s** The s command allows a breakpoint to be set. The program supports the capability of having two breakpoints. Breakpoints may be set only if the program is in RAM. Typing an s shows the breakpoints that are currently set. Figure 2–16 shows how to enter the location of the breakpoint.

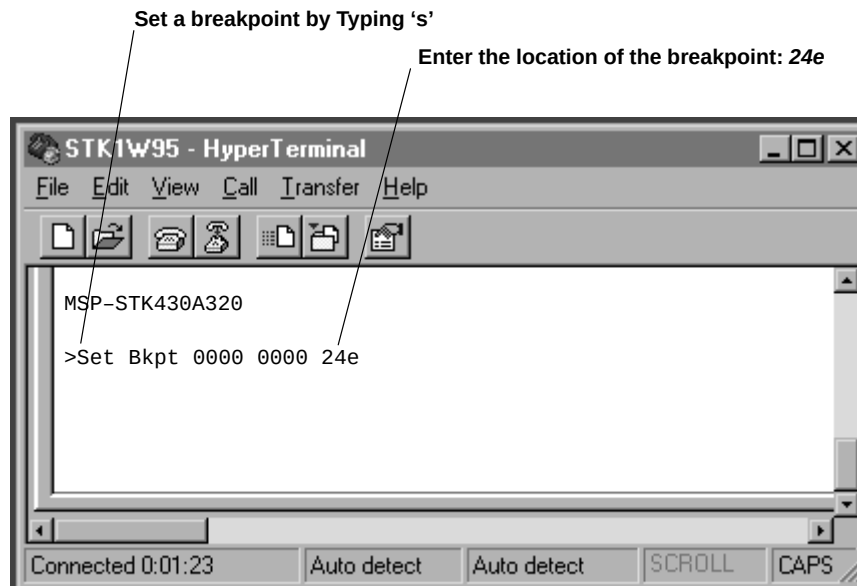


Figure 2–16. Location of a Breakpoint Command

Both breakpoint addresses are displayed. A breakpoint address that contains all zeros indicates that no breakpoint is set. Enter the address to set a breakpoint. Pressing the <RETURN> key activates the breakpoint at that address.

The address of the two breakpoints should not be identical. Only two breakpoints are supported, a third breakpoint cannot be set. In order to set another breakpoint, one breakpoint has to be cleared first with the `c` command. Figure 2–17 shows the breakpoint after entry.

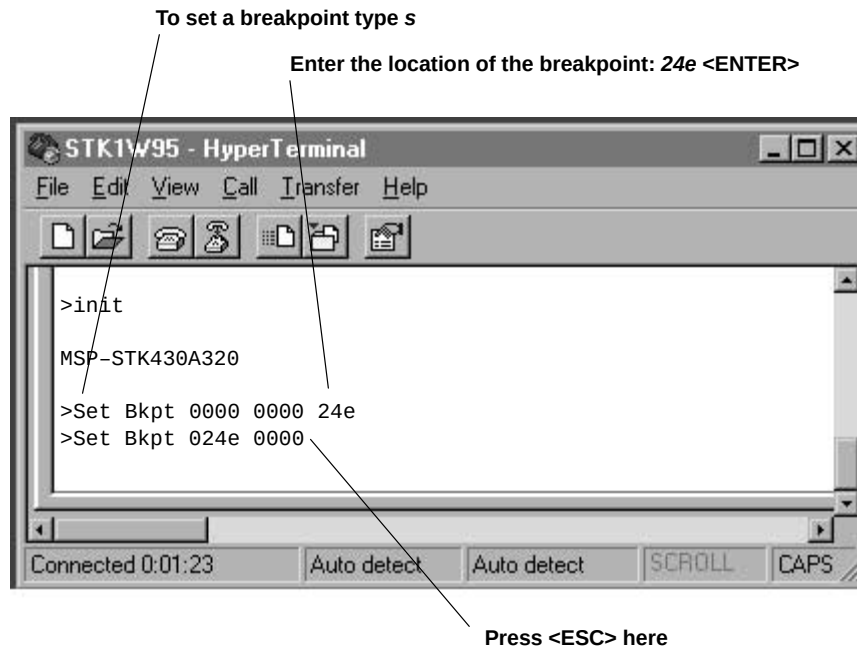


Figure 2–17. Set a Breakpoint Command

- c The c command is used to clear a breakpoint. Typing c shows the set breakpoints, as shown in Figure 2–18.

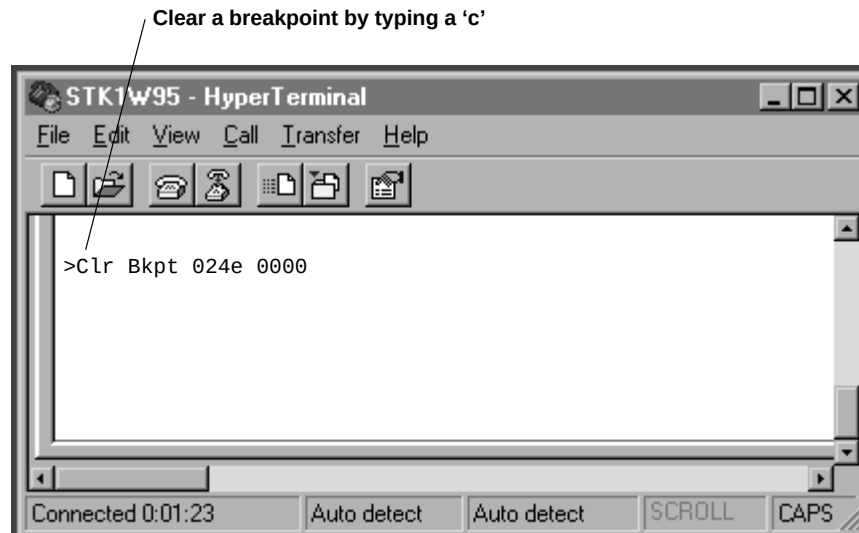


Figure 2–18. Clearing a Breakpoint Command

To clear one of the breakpoints (other than zero), it is necessary to enter the associated address. Typing the <ENTER> key after entering the address clears the breakpoint. See Figure 2–19.

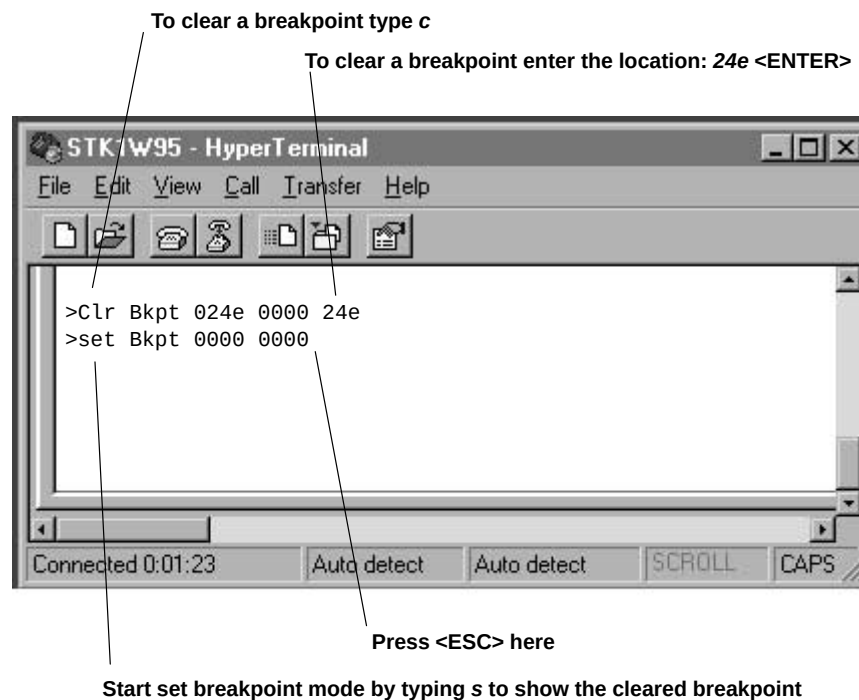
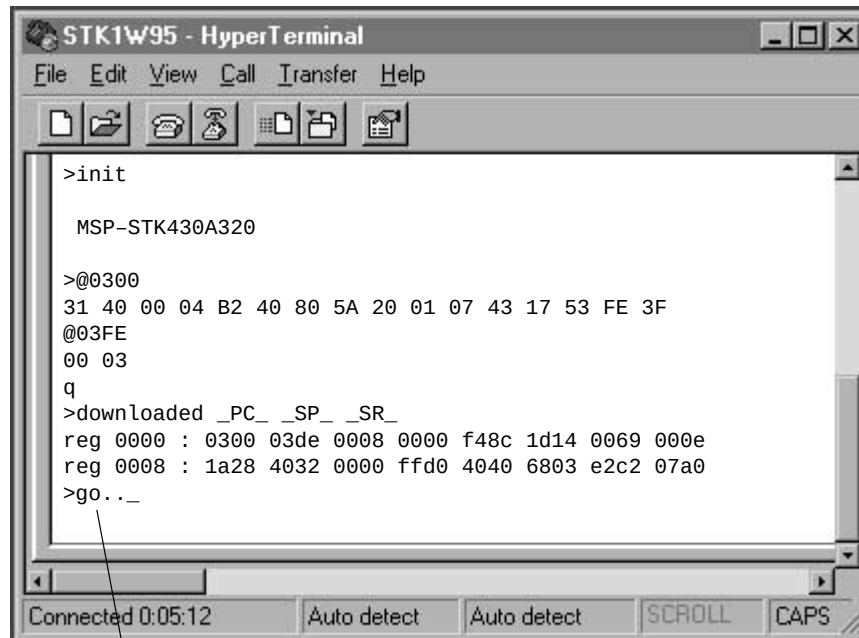


Figure 2–19. Clearing a Breakpoint Location

- g** The *g* command starts/restarts the user application that is loaded into RAM. The system retrieves the start vector of a user's application from memory location (3FEh on the 320 STK/EVK, 5FEh on the 330 EVK) if a new program is loaded, or after executing a PUC command or a user reset. Otherwise, the program execution continues with the actual PC (R0). Return to the Monitor occurs after pressing any key on the keyboard. The *g* command functions as stated if the user application is not running in an interrupt routine (GIE=0 if interrupt nesting is not allowed), and the user application has not disabled the GIE bit.



```
STK1W95 - HyperTerminal
File Edit View Call Transfer Help

>init

MSP-STK430A320

>@0300
31 40 00 04 B2 40 80 5A 20 01 07 43 17 53 FE 3F
@03FE
00 03
q
>downloaded _PC_ _SP_ _SR_
reg 0000 : 0300 03de 0008 0000 f48c 1d14 0069 000e
reg 0008 : 1a28 4032 0000 ffd0 4040 6803 e2c2 07a0
>go.._

Connected 0:05:12 Auto detect Auto detect SCROLL CAPS
```

Enter *g* to start the user application

Figure 2-20. Starting the Application Command

3 Monitor Restrictions

3.1 Register R4

The Monitor Program uses the R4 register internally to return data from the user application to the Hyperterminal. Modifications to the value stored in R4 will result in unexpected behavior of the Monitor Program. The Monitor command *r*, to change the register contents, is not supported for R4. The register R4 can be modified by the user application code. Consequently, the user should exercise caution when using this register.

It is possible to interrupt the user application by pressing the ESC key on the keyboard. This works only if the GIE bit has not been cleared. One exception is possible in spite of a cleared GIE flag. If the user application runs on a breakpoint, program execution will branch to the Monitor as the program runs.

Problems may occur if the user application uses most of its time in interrupt routines. While the application is servicing an interrupt, a new interrupt can only occur if the GIE bit has been set. The GIE bit is set only if the first interrupt routine explicitly sets it within the routine itself, thereby allowing for nested interrupts.

Starting the user application with the Monitor causes the Monitor Program to set the GIE bit. If the GIE-bit was reset before in the user application, it will remain set. This ensures the availability of the RS232 communication but influences the user's software application. The particular interrupts can be enabled with their associated interrupt flags.

3.2 The Instruction CALL R4

In the user application, it is possible to return to the Monitor with the instruction CALL R4. The monitor should be started prior to this so that the contents in R4 are valid and to initialize the Monitor.

A single step should not be executed over the CALL R4 instruction because this will cause unpredictable behavior of the Monitor. The following code is an example that uses the instruction CALL R4.

NOTE:

Execute the single step command only if the program counter points to an instruction in the RAM.

```
WDTCTL      .equ      0120h
WDTHold     .equ      80h
WDT_wrkey   .equ      05a00h

RESET:      .text      0240h
            MOV        #03DEh,SP
            MOV        # (WDTHold+WDT_wrkey),&WDTCTL      ; stop Watchdog
                                                    ; Timer

WAIT:      MOV        #0h,R7
            INC        R7
            CALL       R4

            .sect      "Int_Vect" ,03FEh
            .word      RESET
            .end
```

Figure 3–1. CALL R4 Instruction Code

3.3 Peripheral Hardware/Registers

Do not modify the following peripheral registers and bits because UART operation uses these register and bits.

Table 3–1. Peripheral Registers and Bits

REGISTER	ADDRESS	BITS
TCDAT	44h	All
TCPLD	43h	All
TCCTL	42h	All
IFG1	02h	3
IE1	00h	3
P0IES	14h	1, 2
P0DIR	12h	1, 2
P0IFG	13h	2
P0IE	15h	2

3.4 RAM Locations for the Monitor

The Monitor Program uses eighteen bytes of RAM, from address 200h to 212h, within the RAM area of the MPS430x32x (which ranges from 200h to 3FFh for the MSPx32x family, and 200h to 5FFh for the MSPx33x family). The user application should not use this memory area.

The Monitor Program needs no stack while running the user application. If the user application returns to the Monitor Program using a breakpoint, single step, or keyboard interrupt, an additional stack size of 32h bytes is needed for the Monitor Program. This Monitor Program stack is always set up on the top of the user stack. When returning to the user application, the Monitor Program clears the entire user stack. If the user program is inactive and the Monitor Program is running, 50 bytes are put onto the stack. The stack pointer shown in Figure 3–3 reflects the application situation, not the actual Monitor stack pointer. In these 50 bytes, all register data valid in the application program is saved to be restored when the user application is reactivated. Returning to the user application, the 50 bytes used are freed, and the stack pointer is pointing to the user application.

In most cases, it is efficient to initialize the user stack which is allowed to grow (downwards) until it reaches the address 270h (26Fh–32h=23Eh). Thus, the resulting size of the maximum user stack is 16Dh (3DCh–272h=16Ah) on the MSP430x325 family and 360h (5DCh–272h = 36Ah) on the MSP430x33x family.

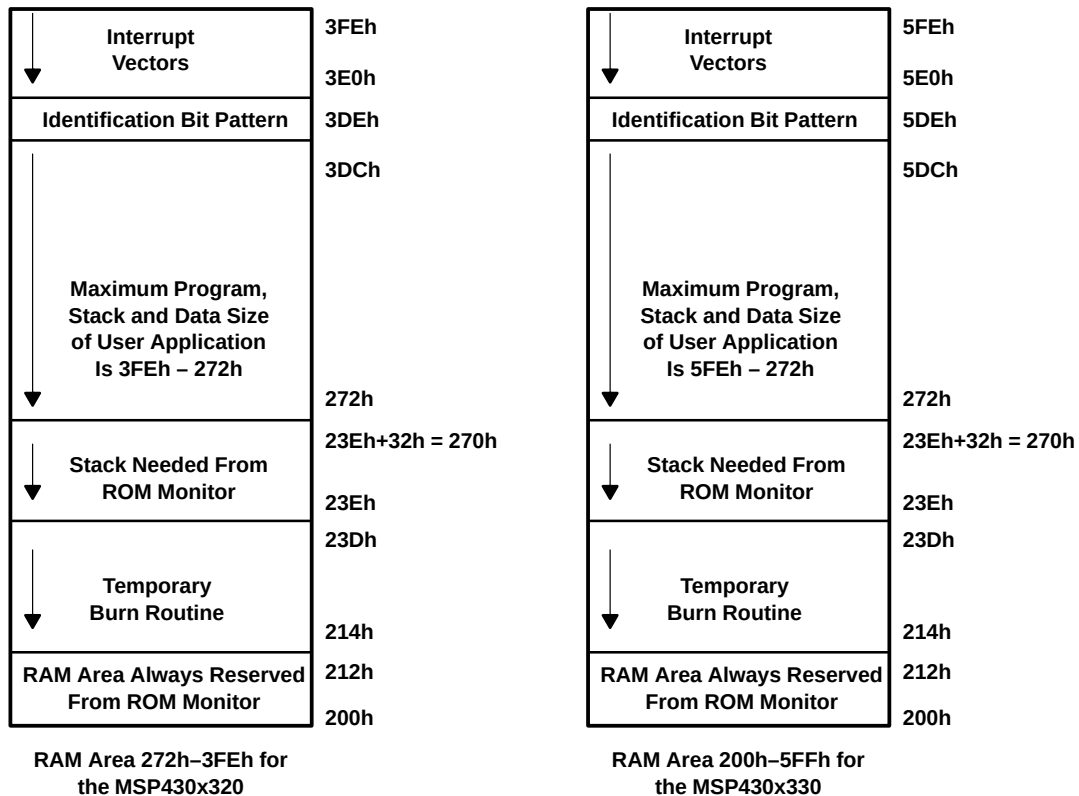
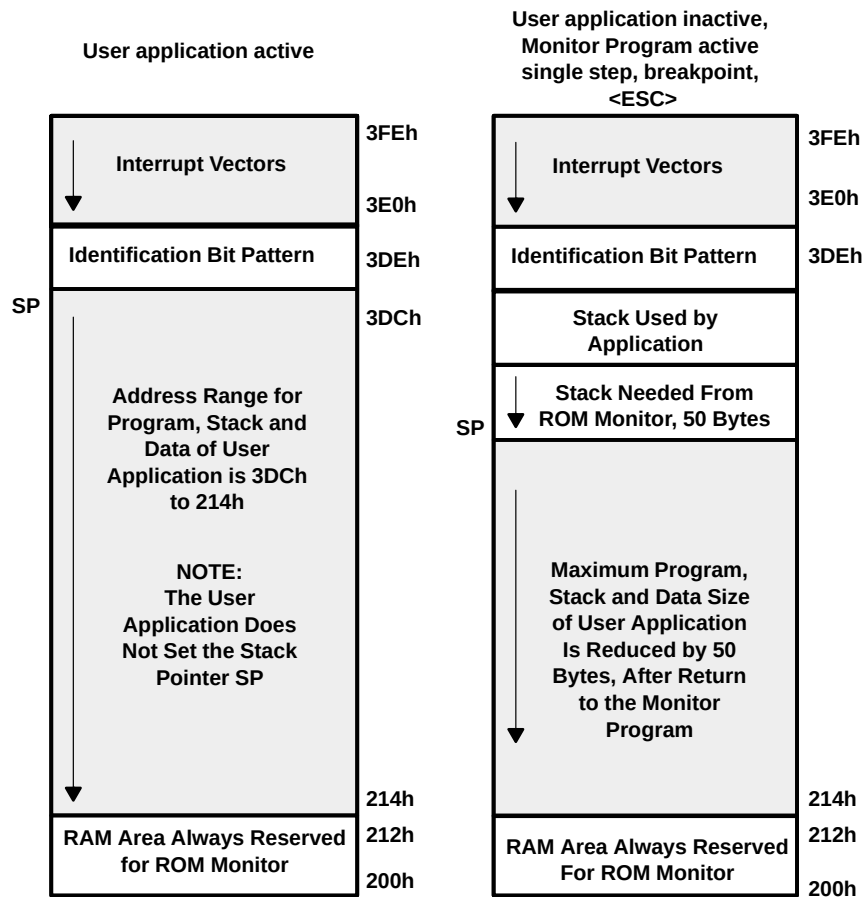


Figure 3–2. RAM Area 272h to 3FEh

The user application data is located within the RAM area from 272h to 3FEh (5FEh for 33x). The locations used statically (200h to 214h) and dynamically (stack) by the Monitor Program limit the size of the available RAM. Due to limited size, two memory configurations are recommended as shown in Figure 3–3.



NOTE:

The identification bit pattern determines if the monitor code or a user's code (program) is executed after a power-up or hardware reset (PUC). See Section 5.2. The temporary burn routine addresses are allocated for the EPROM burn routine, which must run in the RAM. When data is written into the EPROM (program memory), code in the EPROM cannot be executed. Therefore, the burn program has to be in RAM (loaded from the EPROM) so it can be executed.

Figure 3–3. RAM Area 200h to 3FFh for the MSP430x32x Family

In most cases it is efficient not to initialize the user-stack, but to use the stack pointer set by the monitor.

3.5 Writing Data Into the EPROM

When data is written into the EPROM address range, software code is written into RAM locations 214h to 23Dh. All addresses 1000h or higher are assumed to be EPROM. After the write operation is completed, the code in the RAM is no longer needed. The code is written into RAM when a write into the EPROM address range is performed. Ensure that the data in these locations are not needed when the EPROM write code is temporarily loaded into those locations.

A write into EPROM can be done when an application program is executed, or when the monitor is active. For example, when the *mx[n]* command is executed:

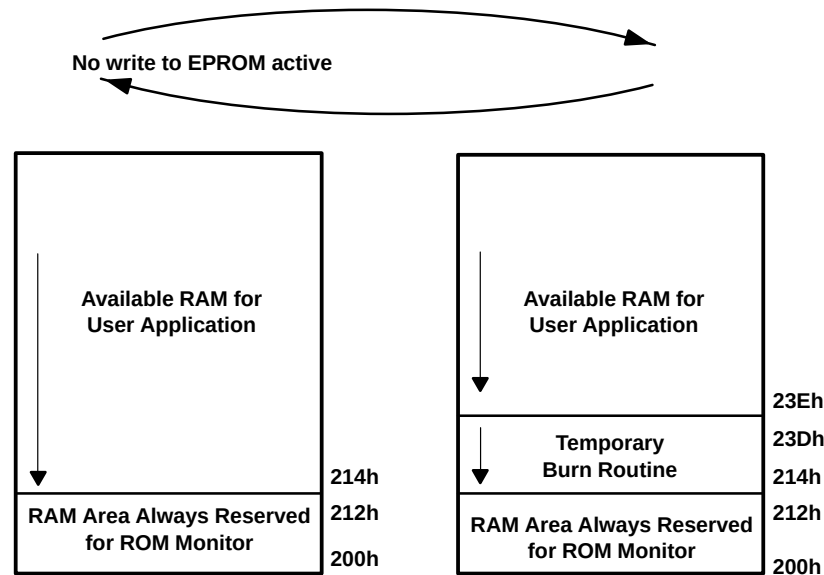


Figure 3–4. Temporary Burn Routine in the MSP430x32x RAM Area

User application data for the MSP430X33x family should be located within the RAM area (from 200h to 5FFh). The locations used statically (200h to 214h), and dynamically (stack) by the Monitor Program limit the size of the available RAM.

NOTE:

The *Identification Bit Pattern* defines if the monitor code or a user's code (program) is executed after a power-up or hardware reset (PUC). See Section 5.2. The *Temporary Burn Routine* user's addresses are allocated for the EPROM burn routine which must run in the RAM. When data is written into the EPROM (program memory), code in the EPROM can not be executed. Therefore, the burn program has to be in the RAM (loaded from the EPROM so it can be executed).

4 Treatment of Interrupts

This chapter describes the special treatment of interrupts in the Monitor environment.

4.1 Use of Interrupts in the Monitor Environment

The interrupt structure of the MSP430 is fully supported by the terminal program with one exception, the NMI interrupt has the same interrupt vector as the RESET interrupt.

There are no restrictions on the interrupt flags, but certain restrictions apply to the interrupt vectors. It is impossible to program the interrupt vectors located in the address range FFECh to FFEh, because they are preprogrammed in the EPROM area and can not be modified. The Monitor Program has the flexibility to allow a second set of interrupt vectors in the RAM address range 3E0h to 3FEh (5E0h–5FEh for the MSP430X33x family) as shown in Figure 4–1. The Monitor Program branches program execution if an interrupt occurs to the associated interrupt vector located in the second interrupt vector address range. The instruction used is an absolute BR command (i.e., BR &0FFEAh for the ADC interrupt). Therefore, during the use of the Monitor Program, the interrupt vectors are moved by an amount of FC00h below their normal location.

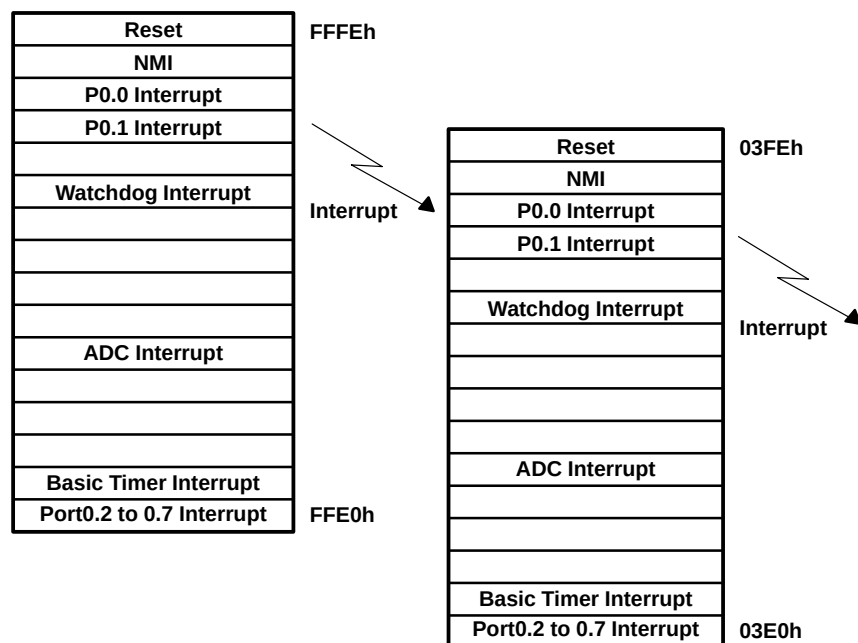


Figure 4–1. Monitor Interrupts for the MSP430x32x Family

Figure 4–2 shows the handling of a P0.0 interrupt for the MSP430X32x family. For the MSP430X33x, replace address 03FAh with 05FAh.

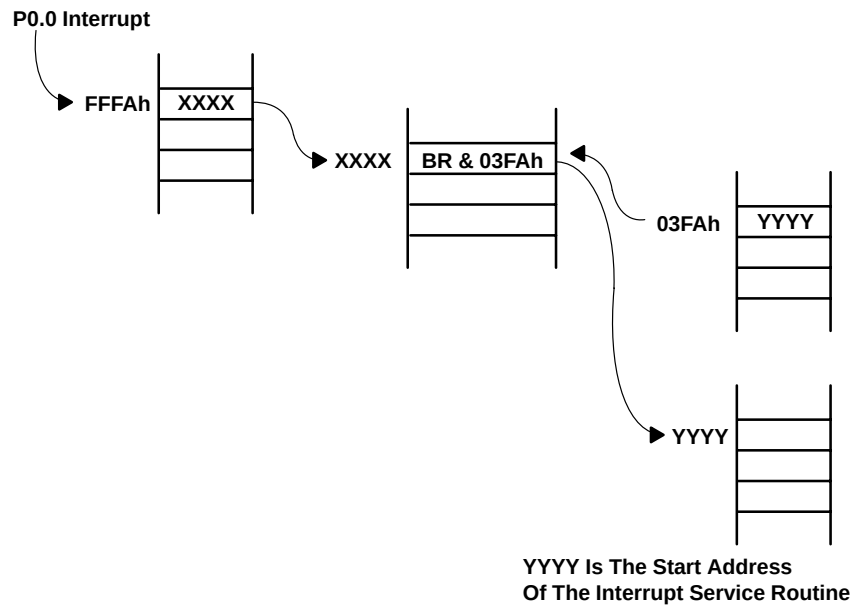


Figure 4–2. P0.0 Interrupt Example

The number of cycles an interrupt is additionally delayed in the Monitor Program depends on the type of interrupt received.

Table 4–1. Type of Interrupt

TYPE OF INTERRUPT	DELAY (NUMBER OF CYCLES)
RESET	14
IN_P01	10
All other interrupts	3

NOTE: Status Register Setting Exception After Interrupts

After entering the interrupt service routine of a Reset or a P0.1 interrupt, the zero-bit Z and the carry-bit C in the status register SR are not reset as expected.

The interrupt vectors and the associated interrupt vector addresses of the MSP430E325 device are shown in Table 4–2 (or Table 4–3 for the MSP430E33x).

Table 4–2. Interrupt Vectors for the MSP430x32x Family

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-up external reset watchdog	RSTI [†] , WDI [†]	Reset	3FEh	15, highest
NMI Oscillator fault	NMIIFG [†] OFIFG [†]	Nonmaskable Nonmaskable	3FCh	14
Dedicated I/O	P0.0IFG	Maskable	3FAh	13
Dedicated I/O	P0.1IFG	Maskable	3F8h	12
		Maskable	3F6h	11
Watchdog timer	WDTIFG	Maskable	3F4h	10
		Maskable	3F2h	9
		Maskable	3F0h	8
		Maskable	3EEh	7
		Maskable	3ECh	6
ADC	ADCIFG	Maskable	3EAh	5
Timer port	‡	Maskable	3E8h	4
		Maskable	3E6h	3
		Maskable	3E4h	2
Basic timer	BTIFG	Maskable	3E2h	1
I/O Port 0	P0.27IFG [†]	Maskable	3E0h	0, lowest

[†] Multiple source flags

‡ Timer Port interrupt flags are located in the module

Table 4–3. Interrupt Vectors for the MSP430x33x Family

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-up external reset watchdog	RSTI [†] , WDI [†]	Reset	3FEh	15, highest
NMI Oscillator fault	NMIIFG [†] OFIFG [†]	Nonmaskable Nonmaskable	3FCh	14
Dedicated I/O	P0.0IFG.0	Maskable	3FAh	13
Dedicated I/O	P0.1IFG.1	Maskable	3F8h	12
		Maskable	3F6h	11
Watchdog Timer	WDTIFG	Maskable	3F4h	10
Timer A	CCIFG0 ^{††}	Maskable	3F2h	9
Timer A	TIFG ^{††}	Maskable	3F0h	8
UART Receive	URXIFG	Maskable	3EEh	7
UART Transmit	UTXIFG	Maskable	3ECh	6
	ADCIFG	Maskable	3EAh	5
Timer Port	‡	Maskable	3E8h	4
I/O Port P2	P2IFG.07 [†]	Maskable	3E6h	3
I/O Port P1	P1IFG.07 [†]	Maskable	3E4h	2
Basic Timer	BTIFG	Maskable	3E2h	1
I/O Port 0	P0.27IFG [†]	Maskable	3E0h	0, lowest

[†] Multiple source flags

‡ Timer Port interrupt flags are located in the module

5 Half Duplex Monitor Software UART

The Monitor Program provides several functions for handling serial data communications using the RS-232 interface. The user can call these functions by using the associated vectors in the terminal program. Use the absolute address mode to call these functions. For example, a possible syntax for preparing the half duplex software UART to receive characters is:

```
CALL    &0FFD6h ;in address FFD6h the vector of RX_Prep is stored
```

another possible syntax is:

```
RX_Prep .equ    0FFD6h
CALL    &RX_Prep ;in address FFD6h the vector of RX_Prep is stored
```

The STK/EVK software UART cannot be used for binary transfers for the following two reasons:

- The protocol has only seven data-bits.
- A zero cannot be received because this is the detection scheme for no-character-received.

The vectors related to these functions are stored in the following locations (see Table 5–1):

Table 5–1. Function/Vector

FUNCTION NAME	VECTOR ADDRESS	FUNCTION PURPOSE
TX_Word	0FFD0h	Transmit 1 space and a four digit hex number in R11
TX_Char [†]	0FFD2h	Transmit 1 char in TXData (20Eh)
TX_Table [†]	0FFD4h	Transmit table (string address should be in R11)
RX_Prep [†]	0FFD6h	Prepare halfduplex software UART for receive
TX_Prep [†]	0FFD8h	Prepare halfduplex software UART for transmit
INT_RXTX	0FFDAh	Interrupt service routine for receive and transmit
Ret_Mon	0FFDCh	Return to the Monitor with a <i>br Ret_Mon</i> statement

[†] Calling TX_Char, TX_Table, TX_Prep, or RX_Prep enables the GIE flag in the status register (SR). The GIE bit remains set even if it was disabled before calling these routines.

5.1 Transmission Parameters of the Software UART

The transmission parameters of the MSP-STK/EVK430x320 and EVK430x330 are:

- 2400 Baud
- 1 start-bit
- 7 data-bits
- 1 parity-bit (even)
- 1 stop-bit

The RTS and DTR pins of the serial port must be set to high to provide the supply voltage for the STK/EVK if no battery is assembled. This is done automatically in the Windows HyperTerminal program. If other communication software is used, the specified pin levels must be met.

5.2 Identification of Bit Pattern AA55h

The interrupt service routine INT_RXTX is used for the receive and transmit function. If another INT_RXTX service routine is being used, the identification bit pattern AA55h must be stored in memory location 3DEh (5DEh for the MSP430x33x family). Otherwise, the INT_RXTX service routine will never branch program execution to the vector located in the user interrupt vector table. This pattern remains in memory until it is changed by the program or the power is switched off.

The flow chart in Figure 5–1 illustrates the entry point of the INT_RXTX interrupt service routine:

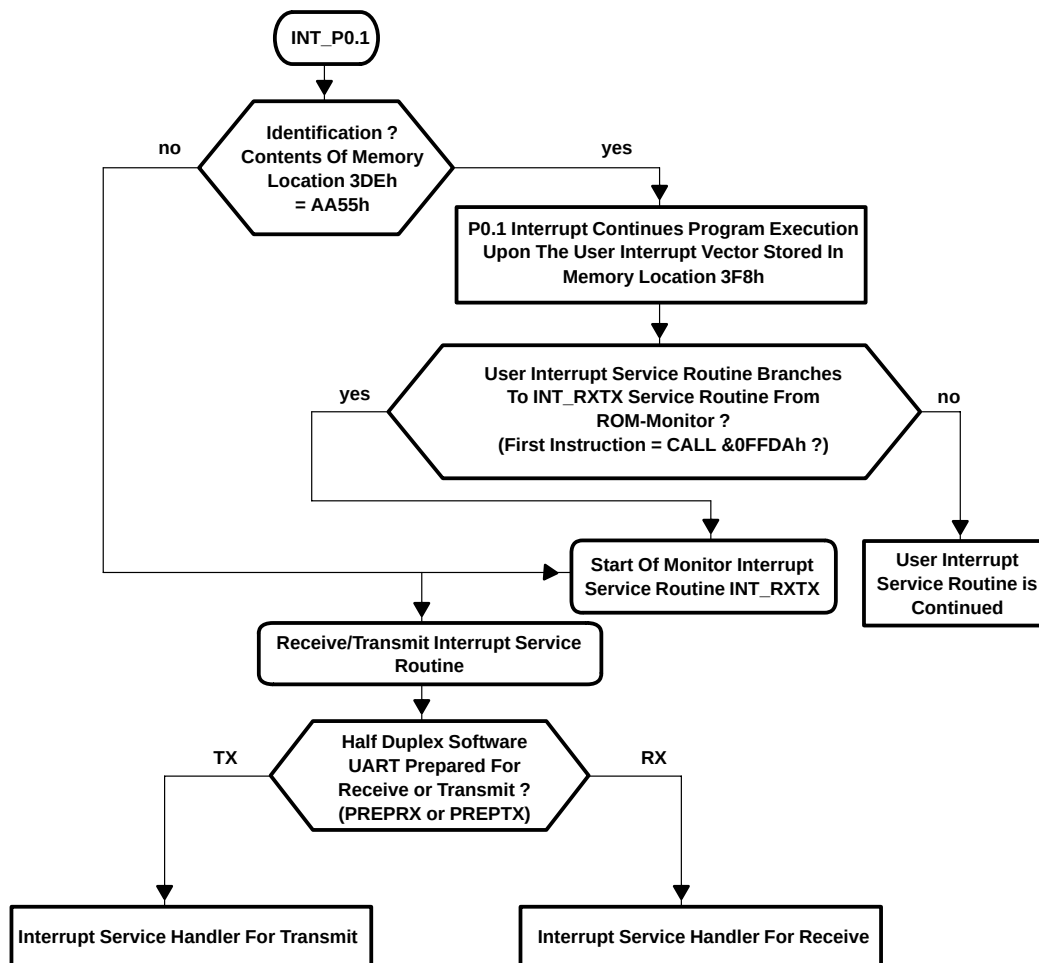


Figure 5–1. Identification of Bit Pattern AA55h for the MSP430X32x Family

NOTE: *Init* Command

Do not use the Monitor *init* command out of the terminal emulator while the bit pattern AA55h is stored in memory location 3DEh on the MSP430X32x, or 5DEh on the MSP430X33x family. Otherwise, the user application with the reset vector stored at 3FEh (5FEh for the MSP430X33x family) will be started.

To use the INT_RXTX interrupt service routine, first load the P0.1 interrupt-vector, which is responsible for the timer interrupt used in the software UART. The P0.1 interrupt vector is stored in the user interrupt vector table at address 3F8h.

The following code is an example of the user interrupt vector table and the associated branch to the INT_RXTX routine.

```
UART:    br &0FFDAh
RESET:   br &0FFFEh
        .sect "Int_Vect",03E0h ; Use 05E0h for the MSP430X33x family
        .word RESET           ; Port0, bit 2 to bit 7
        .word RESET           ; Basic Timer
        .word RESET           ; no source
        .word RESET           ; no source
        .word RESET           ; no source
        .word RESET           ; EOC from ADC
        .word RESET           ; no source
        .word RESET           ; no source
        .word RESET           ; no source
        .word RESET           ; Watchdog/Timer, Timer mode
        .word RESET           ; no source
        .word UART            ; Address of UART handler
        .word RESET           ; P0.0
        .word NMI             ; NMI, Osc. fault
        .word RESET           ; POR, ext. Reset, Watchdog
        end
```

In this example, all other interrupts will continue program execution at the reset vector of the terminal program stored at address FFFEh.

NOTE: Identification Pattern AA55h

The identification pattern must be programmed as the last word of the complete download. To assure this, the section containing only the identification, is the last section in the source file.

5.3 Special Treatment of <ESC> in the Software UART

The software UART treats a received ESC character in two different ways:

- The software UART receives the ESC character and stores it at address 210h. The condition, therefore, is that bit 0 in memory location 200h is reset.
- The software UART receives the ESC and returns back to the Hyperterminal. The condition is that bit 0 in memory location 200h be set.

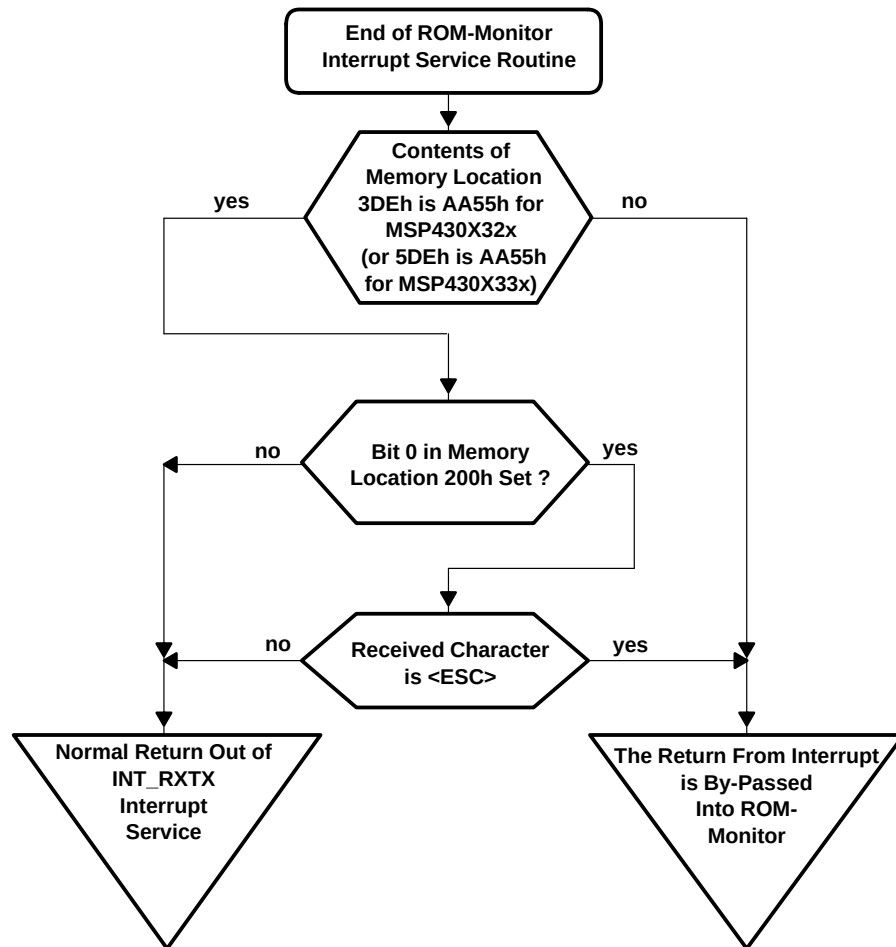


Figure 5–2. Special Treatment of ESC

```
BIC    #01h,&200h    ; reset bit 0, because no special treatment of <ESC> is
                        ; wanted
```

```
BIS    #01h,&200h    ; reset bit 0, because special treatment of <ESC> is
                        ; wanted
```

NOTE: Memory Location 200h

Only bit 0 in memory location 200h may be modified. A modification of the higher bits may result in an unpredictable behavior of the terminal program.

5.4 Transmitting One Character

To transmit one character, copy *AA55h* to memory location 3DEh, or memory location 5DEh for the MSP430X33x family. Store the character to transmit in RAM-location 020Eh.

```
TX:      MOV      #0AA55h,&03DEh    ; &05DEh for MSP430X33x
        MOV.B    #'a',&TX_Data      ; put char to TX_Data
```

Call the function *TX_Char* to transmit the character stored at location 20Eh.

```
        call     &0FFD2h            ; call of TX_Char
```

The function *TX_Char* includes the call of *TX_prep* implicit which prepares the half-duplex software UART to receive. Therefore, it is not necessary to call *TX_Prep* before calling *TX_Char*.

The following code in Figure 5–3 is an example of transmission of the single character s.

```

WDTCTL      .equ 0120h
WDTHold     .equ 80h
WDT_wrkey   .equ 05a00h
TXCHAR      .equ 0FFD2h
TXTABLE     .equ 0FFD4h
PREPRX      .equ 0FFD6h
PREPTX      .equ 0FFD8h
INT_RXTX    .equ 0FFDAh
TXDATA      .equ 020Eh
RXBUF       .equ 0210h

                .text 0240h
RESET:        MOV     #03DEh,SP           ; use #05DEh on the MSP430X33x
                MOV     #(WDTHold+WDT_wrkey),&WDTCTL ; stop Watchdog Timer
                EINT                    ; enable interrupt
TX:           MOV     #0AA55h,&03DEh      ; prepare software UART for the use
                                                ; in user application
                                                ; (05DEh FOR MSP430X33x)
                MOV.B   #'a',&TXDATA      ; put char to TXDATA
;            BIC     #01h,&200h           ; use only if no special
                                                ; treatment of ESC is wanted
                CALL    &TXCHAR           ; call transmit sub-routine in
                                                ; monitor
                CALL    &PREPRX           ; prepare software UART for receive
                                                ; to get back to monitor with ESC
                MOV     #00h,&03DEh        ; prepare software UART only for
                                                ; the use in the Hyperterminal
                                                ; (05DEh FOR MSP430X33x)

ENDL:        JMP     ENDL
UART:        BR      &INT_RXTX
                .sect "Int_Vect",03E0h    ; (use 05E0h on the MSP430X33x)
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .word  UART               ; UART Routine
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .word  RESET              ; Reset
                .end

```

Figure 5–3. Transmitting the s Character

In Figure 5–3, the endless loop at the end of the program can be interrupted by pressing <ESC> in the HyperTerminal. If no special treatment of <ESC> is required in the software UART, bit 0 in RAM location 200h must be cleared. This bit is the ESC-active-flag and allows itself to return back to the Hyperterminal when pressing the ESC key.

Conditions for implementation:

- Correct setting of the P0.1 interrupt vector in the user interrupt vector table
- The first statement of the user interrupt handler is an absolute branch to the INT_RXTX interrupt service routine.
- GIE is enabled.
- The halfduplex software UART is set up to receive.

NOTE: Clear the Bit Pattern AA55h

Do not try to load a program while the bit pattern AA55h is stored at address 3DEh on the MSP430x32x, or 5DEh on the MSP430x33x. To load a new program, clear the bit pattern AA55h at address 3DEh on the MSP430x32x, or 5DEh on the MSP430x33x, or switch off the STK/EVK for a short time to clear the RAM.

5.5 Transmitting a String

The first step to transmit a string is to move the bit pattern AA55h to memory location 3DEh on the MSP430x32x, or 5DEh on the MSP430x33x. The address of the string must be stored in register R11.

The following program transmits the TEST String:

```

WDTCTL      .equ 0120h
WDTHold     .equ 80h
WDT_wrkey   .equ 05a00h
TXCHAR      .equ 0FFD2h
TXTABLE     .equ 0FFD4h
PREPRX      .equ 0FFD6h
PREPTX      .equ 0FFD8h
INT_RXTX    .equ 0FFDAh
TXDATA      .equ 020Eh
RXBUF       .equ 0210h

                .data 0300h
STRING:       .string "TEST"
                .byte 0h
                .text 0240h
RESET:        MOV     #03DEh,SP           ; #05DEh on the MSP430X33x
                MOV     #(WDTHold+WDT_wrkey),&WDTCTL ; stop Watchdog Timer
                EINT                    ; enable interrupt
TX:           MOV     #0AA55h,&03DEh      ; prepare software UART for the
                ; use in user application
                ; (&05DEh on the MSP430X33x
                MOV     #STRING,R11       ; Test: TX_table
;             BIC     #01h,&200h          ; use only if no special
                ; treatment of ESC is wanted
                CALL    &TXTABLE          ; call transmit sub-routine in
                ; monitor
                ; (&05DEh on the MSP430X33x
                CALL    &PREPRX           ; prepare software UART for

```

```

; receive to get back to monitor
; with ESC
MOV      #00h,&03DEh      ; prepare software UART only for
; use in the ROM-Monitor
; (&05DEh on the MSP430X33

ENDL      JMP      ENDL
UART      BR       &INT_RTX
          .sect "Int_Vect",03E0h ; 05E0h on the MSP430X33x
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word UART             ; UART Routine
          .word RESET           ; Reset
          .word RESET           ; Reset
          .word RESET           ; Reset
          .end

```

5.6 Receiving a Character

A character received using the RS232 interface is written from the INT_RTX interrupt routine into memory location 210h. After performing a reset, this memory location contains zeros. Each byte (not equal to zero) received is stored in memory location 210h.

The content of memory location 210h is checked to determine whether or not a byte has been received. After reading the byte, this location must be cleared. The next-received character will overwrite memory location 210h with a new byte.

The following code is an example of receiving a character and transmitting the same character back.

```

WDCTL      .equ 0120h
WDTHold    .equ 80h
WDT_wrkey  .equ 05a00h
TXCHAR     .equ 0FFD2h
TXTABLE    .equ 0FFD4h
PREPRX     .equ 0FFD6h
PREPTX     .equ 0FFD8h
INT_RTX    .equ 0FFDAh
TXDATA     .equ 020Eh
RXBUF      .equ 0210h
          .text 0240h

RESET      MOV      #03DEh,SP      ; #05DEh on the MSP430X33x
          MOV      #(WDTHold+WDT_wrkey),&WDCTL ; stop Watchdog Timer
          EINT      ; enable interrupt
RX          MOV      #0AA55h,&03DEh ; select user interrupt vector
          ; table (&05DEh on MSP430X33x)
          CALL     &PREPRX      ; prepare software UART for

```

```

; receive to get back to monitor
; with ESC
WAIT      TST.B      &RXBUF      ; char. in rxbuf ?
          JEQ        WAIT        ; no, then wait
TX        MOV.B      &RXBUF,&TXDATA ; put char to TXDATA
          CLR.B      &RXBUF
;          BIC        #01h,&200h    ; use only if no special
;                                     ; treatment of ESC is wanted
          CALL       &TXCHAR      ; call transmit sub-routine in
;                                     ; monitor
          CALL       &PREPRX      ; prepare software UART for
;                                     ; receive to get back to monitor
;                                     ; with ESC

          JMP        WAIT
UART      BR         &INT_RXTX
          .sect "Int_Vect",03E0h    ; 050Eh on the MSP430X33x
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word UART               ; UART Routine
          .word RESET              ; Reset
          .word RESET              ; Reset
          .word RESET              ; Reset
          .end

```

The next example receives a string with a maximum of 100 (100=64h) characters, stores them, and then transmits them after the <ENTER> key is pressed.

```

WDTCTL      .equ 0120h
WDTHold     .equ 80h
WDT_wrkey   .equ 05a00h
TXCHAR      .equ 0FFD2h
TXTABLE     .equ 0FFD4h
PREPRX      .equ 0FFD6h
PREPTX      .equ 0FFD8h
INT_RXTX    .equ 0FFDAh
TXDATA      .equ 020Eh
RXBUF       .equ 0210h
cr          .equ 0dh
          .data 0300h
STRING:     .space 0064h          ; reserves receive buffer
          .text 00240H
RESET:      MOV       #03DEh,SP    ; #05DEh on the MSP430X33x
          MOV        #(WDTHold+WDT_wrkey),&WDTCTL ; stop Watchdog Timer

```

```

RX:      EINT                ; enable interrupt
WAIT:    CALL      &PREPRX
MOV      #0AA55h,&03DEh    ; &05DEh on the MSP430X33x
MOV      #STRING,R7        ; load string address
MOV.B    #RXBUF,R6         ; char. in rxbuf ?
CMP.B    #0h,r6            ;
JEQ      WAIT              ; no, then wait
CLR.B    &RXBUF
CMP.B    #cr,R6            ; end of line
                                ; received?

JEQ      TX
MOV.B    R6,0(R7)          ; put char. into string
INC      R7
CMP      #064h,R7          ; all 100 characters are
                                ; received ?

JL      WAIT
TX:      MOV      #STRING,R11 ; load string address
MOV      #0h,1(R7)         ; end of text character
;      BIC      #01h,&200h    ; use only if no special
                                ; treatment of ESC is wanted

CALL     &TXTABLE
CALL     &PREPRX
MOV      #00h,&03DEh        ; &05DEh on the MSP430X33x
ENDL;    JMP      ENDL
UART:    BR       &INT_RXTX

.sect "Int_Vect",03E0h      ; &05E0h on the MSP430X33x
.word RESET                ; Reset
.word RESET                ; Reset
.word RESET                ; Reset
.word RESET                ; Reset
.word RESET                ; Reset
.word RESET                ; Reset
.word RESET                ; Reset
.word RESET                ; Reset
.word RESET                ; Reset
.word RESET                ; Reset
.word RESET                ; Reset
.word RESET                ; Reset
.word RESET                ; Reset
.word RESET                ; Reset
.word UART                 ; UART Routine
.word RESET                ; Reset
.word RESET                ; Reset
.word RESET                ; Reset
.end

```

6 Using Interrupt Vectors in the EPROM

This chapter describes how to use interrupt vectors stored in the EPROM. The interrupt vectors are stored in EPROM so they are not lost when power is removed.

6.1 The Identification Bit Pattern After a Reset

The reset start-up sequence checks the contents of memory location E9DEh to see if its contents are equal to AA55h. If the memory location contents equal AA55h, the EPROM interrupt vector table with addresses ranging from E9E0h to E9FEh is copied to the interrupt table in the RAM locations 3E0h to 3FEh on the MSP430x32x, or 5E0h to 5FEh on the MSP430x33x family, and the key for the software UART is set to AA55h to switch the monitor off. If the contents of memory location E9DEh are equal to 0000h, the next set of interrupt vectors, located at 22h below the previous location, is tested like the first set previously described. If the valid identification is set to AA55h after each power up or hardware reset, the user program starts and the EPROM monitor is switched off. The only way back to the monitor is an indirect branch to *Ret_Mon* (0FFDCh). The user program code implements a branch, as well as the condition for the branch.

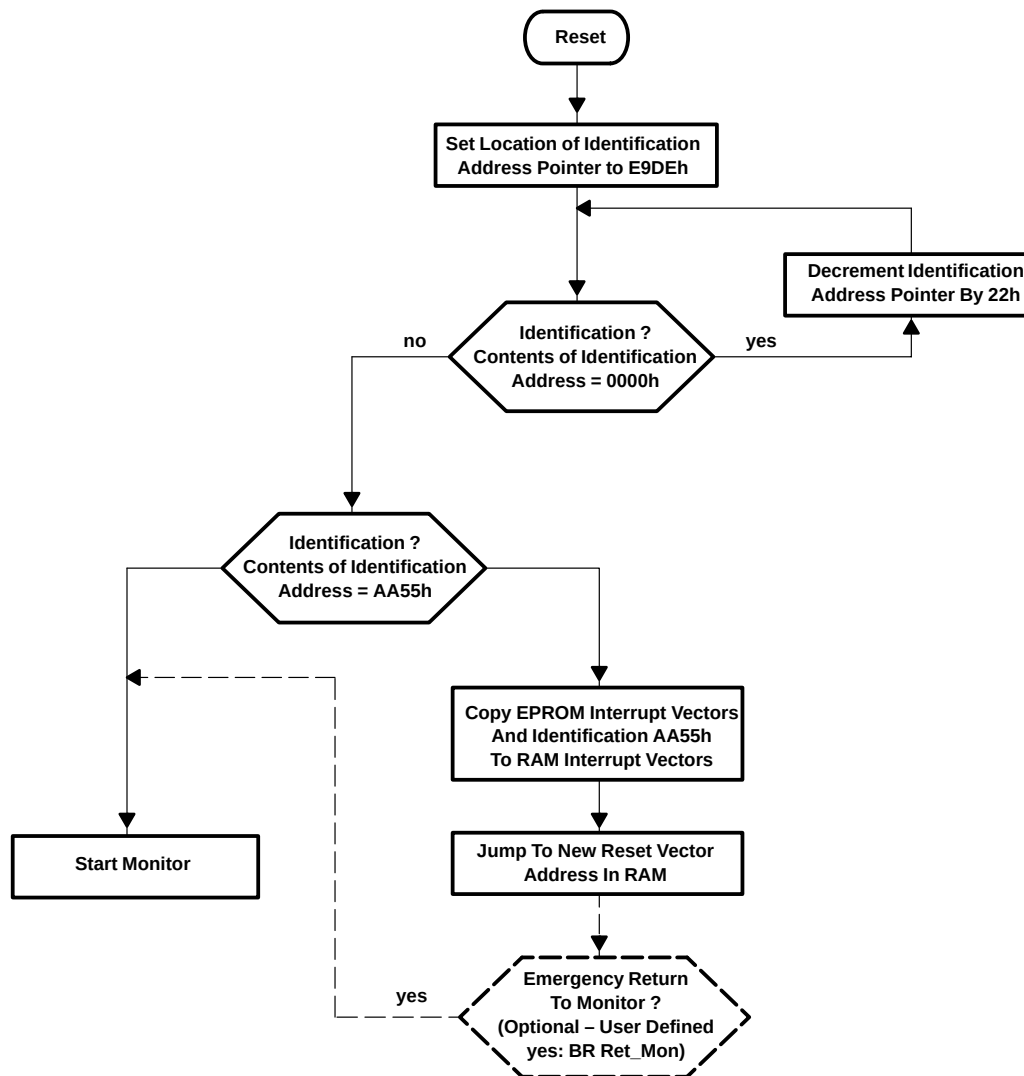


Figure 6–1. Identifying AA55h After Reset

NOTE:

It is important that the emergency return to the Monitor routine works properly and has been tested in RAM before burning it into the EPROM. If the actual identification address contains address AA55h, the Monitor will NEVER start again. Be sure to program the key as the last step in the evaluation of a program, after all errors have been fixed.

The following example demonstrates how to implement and test the emergency come-back to the monitor routine:

1. Set the variable DVLP to 0 after the routine has been successfully tested in RAM. The variable TRIAL indicates the number of interrupt tables that have been burned. The last necessary input is the start address BEGIN.
2. Insert the address where the program is stored. To find a free section in EPROM, use the *m* or the *e* command.
3. The terminal is switched off and the program enters an endless loop after burning the program and executing a reset, or after starting it with the *go* command.
4. To come back to the monitor, press and hold down the *demo* button and press the button.

```
;*****
; Demo for Emergency Burn-over of already written ID = AA55h
;*****
DVK0 .set      1          ; Development = 1, Final = 0
TRIAL .set      1          ; Progressing trial number )start=1)
BEGIN .set      0C000h     ; Startaddress of (new) code in EPROM

;--- definition of testpin

pin .set      01h          ; testpin is P0.0
P0DIR .set     12h         ; Port 0 direction control register
P0IN .set      10h         ; Port 0 input register

;--- ddfine working sections

    .if DVLP =1
    .text 00240h           ; code in RAM during development
    .else
    .text BEGIN           ; code in EPROM at final run
    .endif

;--- test routine is waiting for low at testpin

start
    bic.b      #pin, P0DIR ; testpin is input
    bit.b      #pin, P0IN  ; test testpin
    jnz        user_prg    ; jump to user program if testpin = 1

    clr        003DEh      ; clear ID in RAM (005DEh on MSP430X33x)
    br         0FFDCh      ; branch indirect to Monitor

;--- insert here the start up of your user program

user_prg
    jmp        user_prg    ; Dummy endless loop

;--- define reset vector in RAM for development

    .sect      "RAM_RES", 03FEh ; 05FEh on MSP430X33x
```

```

        .word      start

;--- additionally define reset vector in EPROM if final version

        .if DVLP = 0
            .sect "EPRM_RES", 0E9FEh-((TRIAL-1)*22h)
            .word start

;    write identification to EPROM if final version. This MUST be the LAST
    section !

            .sect "IDENT", 0E9DEh-((TRIAL-1)*22h)
            .word 0AA55h
        .endif

```

NOTE: Identification Pattern AA55h

Program the identification pattern as the last word of the download. To assure this, the section containing the identification should be the last section in the source file.

The following code is an example of the EPROM user interrupt vector table and the associated key for its activation.

```

        .sect "Int_Vect", 0E90Eh-((TRIAL-1)*22h)
        .word POIFG.27      ; I/O Port 0
        .word BTIFG         ; Basic Timer
        .word RESET        ;
        .word RESET        ;
        .word RESET        ; (TimerB)
        .word ADCIFG       ; ADC, Timer/Port
        .word RESET        ; Timer/Port
        .word RESET        ; (SCI)
        .word RESET        ; (TimerA)
        .word RESET        ; (TimerA)
        .word WDTIFG       ; Watchdog timer
        .word RESET        ; (SPI)
        .word POIFG.1      ; Dedicated I/O
        .word POIFG.0      ; Dedicated I/O
        .word OFIFG        ; OSC. fault
        .word WDTIFG       ; Power-up, ext. Reset, Watchdog

        .sect "IDENT", 0E9DEh-((TRIAL-1)*22h)
        .word 0AA55h
        .end

```


7 Memory Configurations for MSP430 Devices

The MSP430 is well suited for the development cycle. The Monitor Program provides the commands s and c to set and clear breakpoints, and SPACE to perform a single step execution in the RAM area for these devices.

Monitor	FFFFh	Monitor	FFFFh	Monitor	FFFFh
	EA00h		EA00h		EA00h
One-Time Programmable EPROM	C000h	EPROM	C000h	EPROM	8000h
RAM	03FFh	RAM	03FFh	RAM	05FFh
	0200h		0200h		0200h
16 Bit Peripheral Modules	0100h	16 Bit Peripheral Modules	0100h	16 Bit Peripheral Modules	0100h
8 Bit Peripheral Modules	0000h	8 Bit Peripheral Modules	0000h	8 Bit Peripheral Modules	0000h
MSP430P325 in MSP-STK430x320		MSP430E325 in MSP-EVK430x320		MSP430E337 in MSP-EVK430x330	

Figure 7-1. Memory Map of the STK/EVK

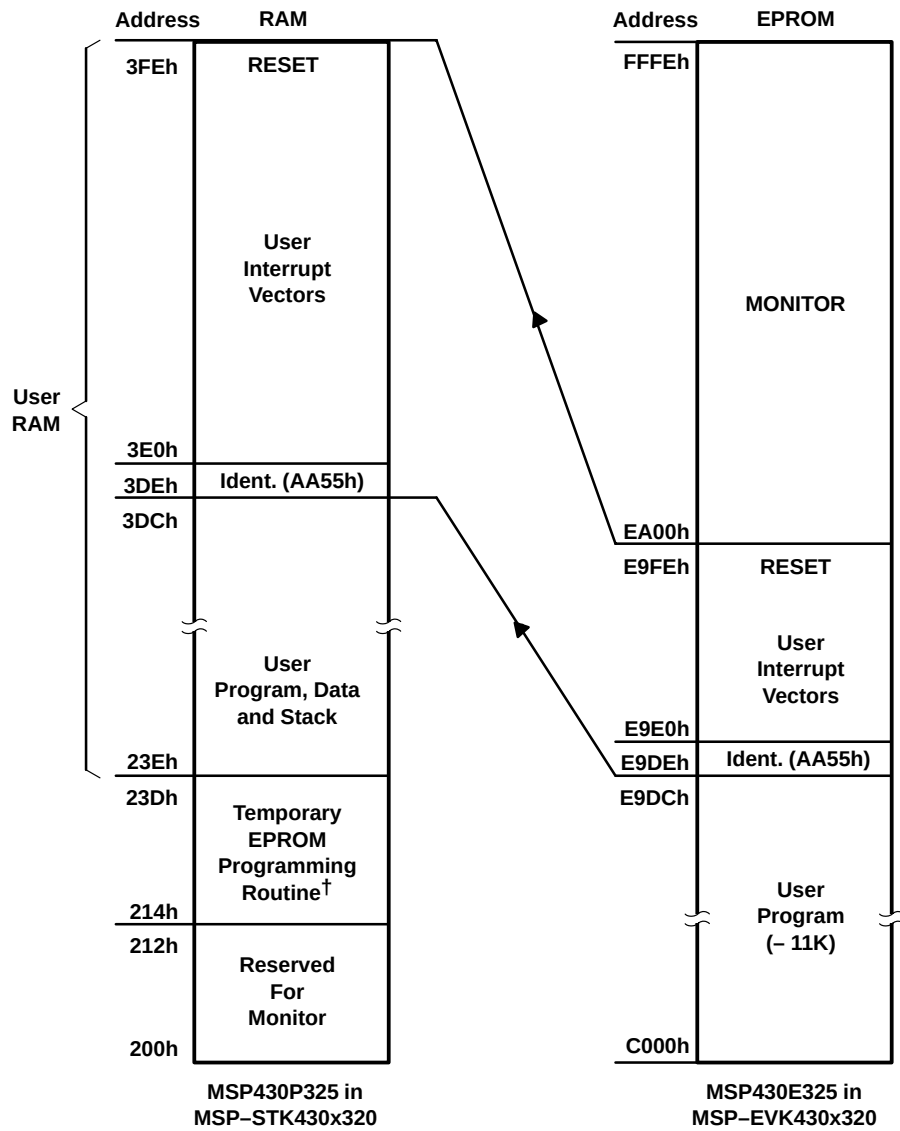


Figure 7-2. Memory Map of the STK/EVK430x32x

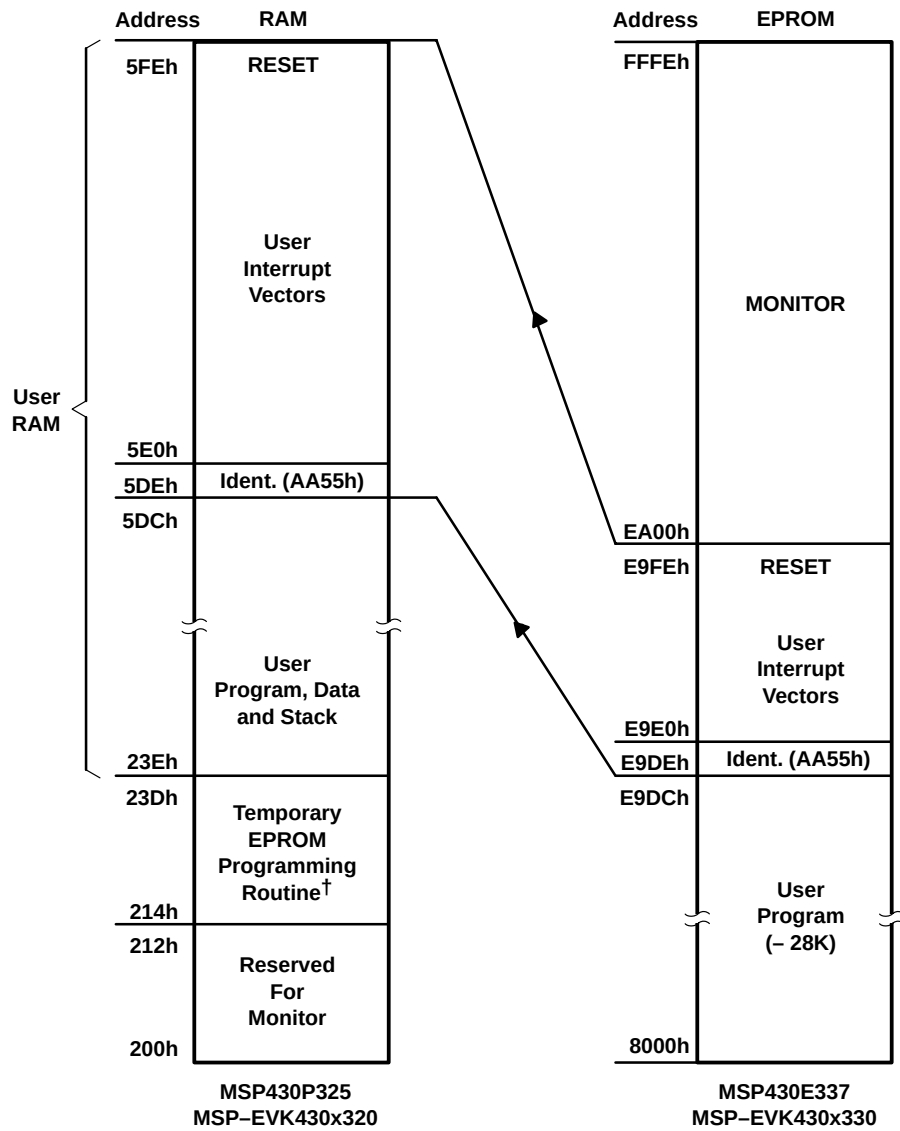


Figure 7–3. Memory Map of the STK/EVK430x33x

Appendix A

Difference Between STK and EVK

	STK	EVK
Initialization banner	MSP-STK430x320	MSP-EVK430x320/MSP-EVK430x330
Device	One mounted OTP device MSP430P325IPM	Two windowed unbearable devices PMS430E325FZ or PMS430E337HFD
Monitor	Programmed	Programmed in only one device. After erasing the device, the monitor program (mon_140.txt for the MSP-EVK430x320, and mon_160.txt for the MSP-EVK430x330) in the STK directory must be programmed again. See Programming Adapter Manual.
LCD	Assembled	Not assembled
Sensor demo	Hardware assembled	Hardware not included

