NOTES:

1 INTRODUCTION

Thank you...

Thank you for purchasing an Adapt812DX Starter Package. You'll find it is a unique and useful tool for working with Motorola's 68HC812A4 microcontroller! Your questions and comments are always welcome. We provide friendly, knowledgeable technical support by telephone, fax, and e-mail to all our customers. As well, we have a comprehensive website with a resource page featuring new information, software, and links to other useful sites on the Internet. See back cover for how to contact Technological Arts.

Purpose of Adapt812DX...

Adapt812DX was designed as an evaluation and application tool for the Motorola MC68HC812A4 microcontroller. It is a fully functional, standalone implementation designed to run in Expanded Narrow Mode. Unique among evaluation boards, its modular design permits it to be easily plugged into any standard solderless breadboard, using the supplied adapters. Developing your application is easy! Simply wire up the desired application circuits in your breadboard and download your code into memory, using the convenient erase/load utility. The fast in-circuit re-programming capability is ideally suited for the frequent code changes typical during the product development phase. Once the initial design has been developed, the application circuitry can be transferred to the supplied prototyping card for a more permanent assembly, suitable for mounting in an enclosure. If multiple units are required, a printed circuit board can be designed by the user to accomodate the application circuitry.

Product Configurations

Adapt812DX is available in two memory configurations: 128K Flash and 128K SRAM, or 512K Flash and 512K SRAM.

Both configurations include a time-of-day clock/calendar chip, with battery backup for both the clock and SRAM. In addition, the ‘812A4 has 1K SRAM and 4K EEPROM. A firmware utility has been loaded into the on-chip EEPROM by Technological Arts, making it easy for you to program and erase Flash via your PC serial port, for quick and easy programming. This makes separate programming hardware unnecessary. However, the Flash chip is socketed, so that it may be programmed out-of-circuit using a Flash programmer. The utility also allows you to set the time and date.

Communications

An RS-232-compatible serial interface port (RX & TX only) is built into Adapt812DX, allowing communication with a PC, or any other device which has an RS-232 serial port. The logic-level RXD and TXD signals from the micro are also brought out to the 50-pin header, for applications such as RS-485 or MIDI. A dedicated RS-485 serial port is implemented using the second SCI on the 68HC812. This is useful for networking industrial control applications.

How is Adapt812DX different from other evaluation boards ?

Most evaluation and development systems available tend to use up system resources with resident monitors or debuggers. These are fine for learning in a classroom environment, but tend to be too expensive and bulky for embedding into a real application. Also, the prototyping area provided is often limited, and does not lend itself to re-usability. By contrast, we took a modular approach. With the Adapt12 system, all I/O lines and control signals are brought out to two standard 50-pin interface connectors. With several different connector options available, you can use the module in whatever way best suits your needs. With the solderless breadboard adapter, you can treat the module like a big chip, and plug it right into a couple of breadboard strips. Forget about soldering or wire-wrapping-- get started developing your application right away. Your prototyping space is virtually unlimited, using

5.0 SOURCES

Internet Resources

•*Technological Arts*
www.interlog.com/~techart
e-mail address: techart@interlog.com
Join our Adapt12 email list to network with other Adapt12 users and receive automatic notification of updates, new product announcements, and special promotions. Visit our SUPPORT webpage for details.

•*Motorola Freeware:* www.mcu.motsps.com/freeweb/pub/

•*Karl Lunt (SBASIC compiler):* www.seanet.com/~karllunt

•*Kevin Ross (BDM12 pod):* www.nwlinkcom/~kevinro

•*ImageCraft (ICC12 C compiler):* www.imagecraft.com

•*miniIDE (freeware HC12 development environment for W95/NT platforms, by Marius Greuel):*
http://members.tripod.com/~miniide

Publications

Motorola Fax-on-Demand: (602) 244-6609 or 800-774-1848

Motorola Semiconductor Literature Distribution Center
P.O. Box 20912, Phoenix, AZ 85036 1-800-441-2447

• CPU12 Reference Manual (CPU12RM/AD)

The clock chip provides a periodic alarm function, which is enabled by the MXFlash utility provided. However, to make the alarrm generate an interrupt to the MCU, you must insert a jumper at W11, enable IRQ (edge-sensitive), and include an appropriate IRQ interrupt service routine in your application code.

4.6 Notes on the Battery Backup Circuit

The battery backup circuit is based on the Dallas Semiconductor DS1210 “NonVolatizer” chip. This chip provides voltage monitoring and smooth power-source switching, protecting RAM from corruption during power up and power down events. One characteristic worth noting is the power-up time delay for the Chip Enable signal. According to the DS1210 data sheet, this delay can be as long as 120 ms. For this reason, make sure that your firmware does not try to access RAM for at least 120 ms after power comes on, or the results may be invalid.

Important! The battery-backup feature includes a 23mm 3V Lithium coin cell. Be careful not to place the board on a conductive surface, or the battery life may be shortened considerably.

If you choose not to use a battery, you will need to jumper Vbat to Vcc. A convenient location for this jumper is between the two diode patterns on the circuit board (D1 and D2), since these diodes are not used. Solder a small wire from the round pad of D1 to the round pad of D2.

solderless breadboards! When you’ve got a design working and your ready to make it permanent, just use the supplied prototyping card build your fully customized, compact application at low cost. Additional prototyping cards are avaialble, and application-specific cards will be coming soon.

2 USING ADAPT-812DX WITH SOLDERLESS BREADBOARDS

The standard Adapt812DX Starter Package comes with two 50-pin adapters to allow you to plug the module into one or two solderless breadboards (“protoboards”). These adapters may be used on one or both I/O connectors (H1 or H2). Use care not to short any pins on H2, since many of the signals present are data and address lines used by the expanded memory on board.

CAUTION!

Never insert or remove your module from a “live” breadboard. Make sure the power is OFF !

1) Any breadboard will do; however, you will find that the kind made with a softer, more pliable plastic (such as nylon) will be easier to use and more durable.

2) When plugging the module into your breadboard, you may find it easier to put the adapters in place first. Then plug the module into the adapters when you have finished wiring your I/O circuits. To remove the module, hold the adapter down by the ends, and gently pry up the module, using an end-to-end rocking motion.

3) Plug Adapt812DX into the middle area of your breadboard strip to allow maximum access on each end to all the signals. If possible, place an additional breadboard section in parallel on each side for easier wiring of your circuits. *(HELPFUL HINT: If you*

are using the Analog inputs, make sure to wire your analog circuits as close to these pins as possible, to keep noise levels down.)

4) Choose a convention for wiring your power distribution buses. A logical approach is to make the inside bus logic 5V, and the outside buses GROUND. Never supply external power via J1 if you are supplying 5VDC via the breadboard connector pins. However, always connect the breadboard GROUND to the module GROUND.

5) If you are using voltages other than 5V, make sure to keep these well away from the I/O pins and tie-strips, to avoid accidental shorts which may damage the module.

3 TUTORIAL

Note that this manual is not meant to provide an exhaustive study of the 68HC12 family of microcontrollers, but rather to help you get started using the Adapt812DX microcontroller board as a learning and application development tool for 68HC812A4, whether you're a beginner or an expert. If you are a beginner, you will benefit from additional material listed in the Reference section of this manual, and links provided on the Resource page of our website (see back cover for URL). You will find Motorola's 68HC11 Reference Manual an invaluable guide to the workings of the 68HC12, since most of the subsystems of the 68HC12 work the same way as their 68HC11 counterparts.

If you have internet access, make sure to subscribe to our Adapt12 User Email List (details on Resource webpage). This free service allows you to network with other Adapt12 users, and receive important notices regarding new products, bug fixes, and software upgrades.

encounters addresses in the range $c000–$ffff, it sets PPAGE to the lastpage of Flash, and subtracts $4000 from the address, to derive a destination address in the Program Window. When the board is reset in Expanded Narrow mode, the EEPROM-resident MXFlash utility moves out of the way, down to $1000, and the reset vector will be fetched from the topmost block of Flash.

4.5 Using the Real-time Clock/Calendar Chip

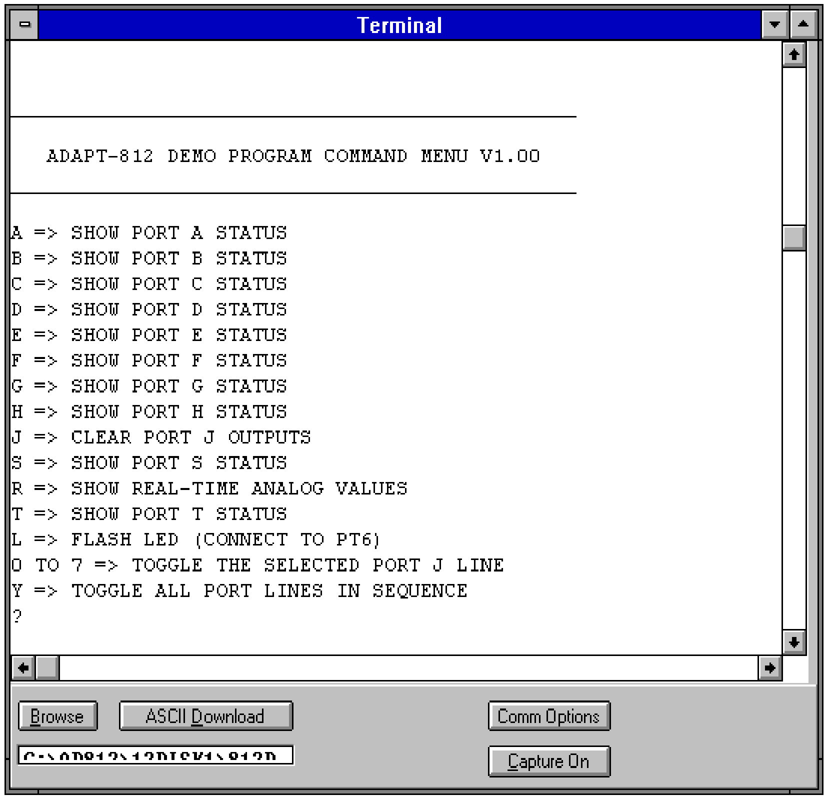
A Philips PCF8593 Real-time Clock/Calendar circuit is implemented on the board. The PCF8593 uses an industry-standard I2C 2-wire interface, and is battery-backed by a Lithium battery, which also provides backup of the SRAM. These features make Adapt812DX particularly suitable for use in data logging applications, where analog values, etc. could be time-and-date stamped, and stored in SRAM at regular intervals.

The Starter Package disk includes a complete data sheet for the PCF8593, in Adobe Acrobat format, as well as assembler source code routines for the chip. The MX1Flash utility described above offers menu selections for reading and setting the time and date from the chip, and displaying them on your terminal screen. The year function is implemented in the chip as a value from 0 to 3, in order to determine a leap year. If the year counter is zero, a 29th day is automatically added for February. Since the chip does not implement an absolute year counter, the user must implement this in software. The date-setting function provided stores the the year value entered by the user in a two-byte location at the beginning of battery-backed RAM (addresses $7000 and $7001). While all the time and date counters automatically roll over, the year value will not roll automatically. If the absolute year value is needed, it is up to the user's software to check the year counter in the clock chip periodically to see if it has "rolled", and make the necessary adjustment to the absolute year value stored in RAM. For applications requiring accurate timekeeping, consult the data sheet for calibration methods.

4.42 The MX1Flash Utility

To use MX1Flash, set your terminal program's baudrate to 9600, set Adapt812DX switches to RUN and SGL, and press RESET. You'll see the MX1Flash utility menu displayed in your terminal window. First type **e** to erase the Flash memory. Then type **p** to program an s-record into Flash. Use the ASCII file transfer function of your terminal program to send the s-record file to the board. When finished, switch to EXP and press reset. The program in Flash should now be running. Try the Flash loading procedure using **mx1demo1.s19** (**mx1demo5.s19** if you have 512K Flash**)**, located on your Adapt12 Utilities disk. Look at the source code of **mx1demo.asm** to see how the file is set up to create a banked-Flash-loadable s-record. To download a new s-record to Flash, simply switch JB1 back to SGL, press reset, and make the appopriate selection from the Flash utility menu. Note that you must perform an Erase before sending a new file. You can use the **v** command to verify the contents of Flash against an s-record file. Just use the ASCII download function again to send the s-record file to be verified against. Failure to program or verify usually indicates that your s-record file contains references to invalid ad dresses. If it *does* match, you will get a confirmation message, and the menu will be re-displayed.

The Flash programming function will handle s-records generated for either banked or non-banked memory schemes, as well as recognizing and programming non-Flash addresses. An s-record file destined to be used in non-banked mode will include s-records above the Program window (ie. $c000 and higher). These are the types of s-records generated by ICC12 and SBASIC. Before loading such a file into Flash, select the memory size of your Flash device (**1** for 128K or **5** for 512K) and Non-banked Mode (**n** from the menu). When in Non-banked Mode, the algorithm automatically sets the PPAGE value to the second-last page (hence it needs to know whether the system is 128K or 512K) when programming target addresses in the range $8000–$bfff. When it



```
ADAPT-812 DEMO PROGRAM COMMAND MENU V1.00

A => SHOW PORT A STATUS
B => SHOW PORT B STATUS
C => SHOW PORT C STATUS
D => SHOW PORT D STATUS
E => SHOW PORT E STATUS
F => SHOW PORT F STATUS
G => SHOW PORT G STATUS
H => SHOW PORT H STATUS
J => CLEAR PORT J OUTPUTS
S => SHOW PORT S STATUS
R => SHOW REAL-TIME ANALOG VALUES
T => SHOW PORT T STATUS
L => FLASH LED (CONNECT TO PT6)
0 TO 7 => TOGGLE THE SELECTED PORT J LINE
Y => TOGGLE ALL PORT LINES IN SEQUENCE
?
```

Figure 3.1 - Demo Menu shown in ICC12 Terminal Window

CAUTION!
Never insert or remove your module from a "live" breadboard. Make sure the power is OFF !

3.1 Getting Started

Adapt812DX has a demonstration program already programmed into Flash when you receive it. This is a useful program for testing your communications setup and monitoring & controlling the various I/O lines of the micro. It can also form the basis of your own application code.

You can power the module in one of two ways:

1) supply power via the external power connector; just connect a DC voltage of 8 Volts or more (maximum 15V) to the external power connector J1. Red is positive, and black is negative (ground). ***CAUTION! Make sure you have the polarity correct!***

2) or, supply regulated 5VDC via the appropriate pins on the 50-pin connector (H1). See Appendix A for the module pinout diagram. ***CAUTION! Double-check your connections before applying power!*** (Note: if you are powering the board via H1, and you're concerned about power draw on the module, you may wish to remove the voltage regulator U3 since it will add several mA to the total board's current requirements.)

To use the demo program, make sure switch SW2 is in the RUN position, and the switch at JB1 is set to EXP. Connect the supplied serial cable between Adapt812DX and a serial port on your computer. (With some PCs, you will need a 9-pin to 25-pin adapter.) Run any terminal program on your PC. Suggested DOS programs are ProCommPlus, Kermit, or Mirror. On a Windows machine, you can use Windows Terminal (W3.1), Hyperterminal (W95/98), ICC12 for Windows, or miniIDE. In your terminal program, set the baud rate to 9600, parity to NONE, # DATA BITS = 8, and #STOP BITS = 1. Press Adapt812DX RESET button (SW1). LED D1 will blink four times, indicating the demo programming is running. Press the ENTER key on your keyboard. A menu of commands will appear on your computer screen, followed by a command prompt "?" symbol. Each command is activated by a single keystroke. Typing a command not listed will cause the menu to be re-displayed. Figure 3.1 shows an example of the demo program running in ImageCraft's ICC12 IDE terminal window.

In general, typing the letter name of an I/O port in the demo program returns the state of that port. Try putting switches on some of these input port lines. Connect one side of the switch to the port pin and the other side to ground. Note that external pullup resistors are not required, since most ports have internal pull-up resistors which are enabled out of reset (refer to the 68HC812A4

and ORG the vector table at $bfc0. Also, you need all the vectors to point to locations that are not affected by the program paging system, but are accessible when EEPROM is in the top 4K of the map. This leaves a 12K block of Flash from $c000–$efff where all Interrupt Service Routines (ISRs) and startup code should be placed. Once the startup code switches on paging, subroutines in the program page window can be accessed using the CALL and RTC features of the HC12 instruction set. Listing 1 is an example of how a typical sourcecode file would be organized.

LISTING 1

```
org      $c000
;up to 12K of code goes here (up to $efff)
; This would include:
;1) startup code (where reset vector points; turn on memory expansion and
paging)
;2) main code (and some subroutines, if space allows)
;3) interrupt service routines

;paged program memory
        org      PPAGE
        fcb      0
        org      $8000

;up to 16K of subroutines for page 0 go here (up to $bfff)
.....
        org      PPAGE
        fcb      1
        org      $8000

;up to 16K of subroutines for page 1 go here (up to $bfff)
.....
        org      PPAGE
        fcb      2
        org      $8000

;up to 16K of subroutines for page 2 go here (up to $bfff)
.....
etc.
        org      PPAGE
        fcb      7                ;use $1f for a 512Kx8 Flash chip
        org      $bfc0

;vector table starts here
;IMPORTANT:
;all vector entries must point to addresses between $c000 and $efff
;(ie. the startup code and all ISRs must be located in this 12K block)
        .
        .
;end
```

into the destination address, creating the virtual address needed to map it into the linear physical address space of Flash. For in-circuit programming of Flash, however, the scheme is somewhat simpler, since no virtual addresses need be created. Since PPAGE is just a memory location, it can be written to (as many times as necessary) by means of an s1 record during the course of downloading. Of course, the assembler will have to generate the s-records sequentially during assembly. The necessary s1 record to alter the value of PPAGE can be implemented by means of the ORG and FCB directives of the assembler. Using this scheme, one can preface a block of code or subroutines with a directive which sets PPAGE to the desired page.

The MX1Flash Utility, residing in on-chip EEPROM, allows user interaction with a terminal program to perform such necessary operations as Erase Flash, Program an s-record file, and Verify the contents of Flash. In order to execute the utility out of reset, you reset the chip in single-chip mode, forcing the EEPROM into the vector space. The utility then switches to expanded mode by initializing the appropriate memory expansion and mode registers, and it is ready to load code into Flash. By careful placement of ORG and FCB directives throughout the assembly source code (to force the appropriate value into PPAGE before loading a new page of code), the s-record loader will automatically change PPAGE "on-the-fly", during s-record downloading. This supports programming of code into any combination of program pages in Flash. In fact, since the topmost page also appears in the space \$c000–\$ffff, it is possible to program the vectors via the page window– which is indeed necessary, since the on-chip EEPROM is active in the vector space from \$f000–\$ffff, preventing direct access to Flash via these addresses. There are a couple of tricks to implementing this scheme, however. In order to program the last page of Flash with the vectors and startup code, you have to use the PPAGE windowed system, in which the address range is \$8000–\$bfff. Therefore you'll set PPAGE=7 (or \$1F, for a 512K Flash)

Technical Summary for details). In the demo program, PT6 is used as a tone output for a speaker. It is also connected to LED D1, to provide a visual output. You can drive a small piezo speaker directly by hooking one end to PT6 through a 330-Ohm resistor, and the other end to ground. When you press RESET, or type "L" when the demo program is running, you will hear two beeps from the speaker (or the LED will flash twice).

In the demo program, PORTJ is set up as all outputs. Typing a digit between 0 and 7 causes the output state of the corresponding PORTJ line to be toggled (eg. typing 3 causes PJ3 to flip to a high if it was low previously, or a low if it was high previously). This allows you to activate LEDs (when driving LEDs directly from an output port, limit the current to a maximum of 10mA with 330-Ohm current limiting resistors on each LED); or drive relays, solenoids, or motors (with appropriate driver circuits). Typing J forces all PORTJ output lines low. Typing R causes the values of all 8 analog-to-digital converter (AN0-AN7) channels of to be continuously updated on the screen (near Real-time updates) The display will continue to be updated until a key is pressed. Analog channels (AN0-AN7) can read voltages between 0 and 5 Volts. Try putting a 10K-Ohm (or higher) pot across the VRL and VRH pins (pins 30 and 31), and connect the wiper to an AN input through a 1K-Ohm current limiting resistor; then change the pot setting, monitoring the AN values on the screen. Unused AN channels should be grounded to VRL through a minimum 1K-Ohm resistor. These inputs are not internally protected from electrostatic discharge (ESD) as the other input port lines are. *(HELPFUL HINT: Grounding multiple adjacent analog inputs is easy by plugging a bussed resistor SIP in your breadboard and jumpering the SIP common pin to VRL.)*

3.2 Writing Your First Program

If you are already experienced with the 68HC11 family of microcontrollers, writing 68HC12 programs will not present a challenge. In fact, you can use your existing 68HC11 assembly code and re-assemble it for the 68HC12. There are a couple of things to keep in mind when doing this. The first is assembler syntax. You may need to edit your source file to conform to the syntax and directives requirements of the HC12 assembler you are using. Keep in mind, too, that the register block default location is $0000 and the 1K internal RAM is at $0800. This means you would initialize the Stack Pointer to $0c00. Also, the HC12 bus speed is a lot higher than the HC11. This will mean changing some initialization values for control registers and revising delay constants if you are doing software timing loops. Of course, there is an expanded inteerupt vector table, handling the additional hardware functions of the HC12. As with the HC11, you will have to define at least the Reset vector in every program you write.

To explore the new instructions and addressing modes of the HC12, you should refer to the Motorola CPU12 Reference Manual, available from the Motorola Literature Center or in Acrobat format from Motorola's website.

As mentioned in the previous section, a demo program resides in your module's Flash memory when you receive it. This demo program is written in Freeware AS12 assembler syntax, and is intended to provide you with an easy way to verify your hardware setup (ie. power supply, serial connection, PC software, etc.). It also provides you with an excellent starting point for developing your own program. Rather than starting from scratch, you can make a copy of the demo source file and remove and add features, to transform it into what you need.

Many people approach programming by spending hours or even days writing a program from scratch, then assembling it and downloading it. Then they cross their fingers and reset the board, praying everything will work. About 99% of the time, their hopes are dashed, as the board does something completely different than they expected, or worse– it appears to do nothing! At that point, they either give up, or purchase expensive diagnostic equipment,

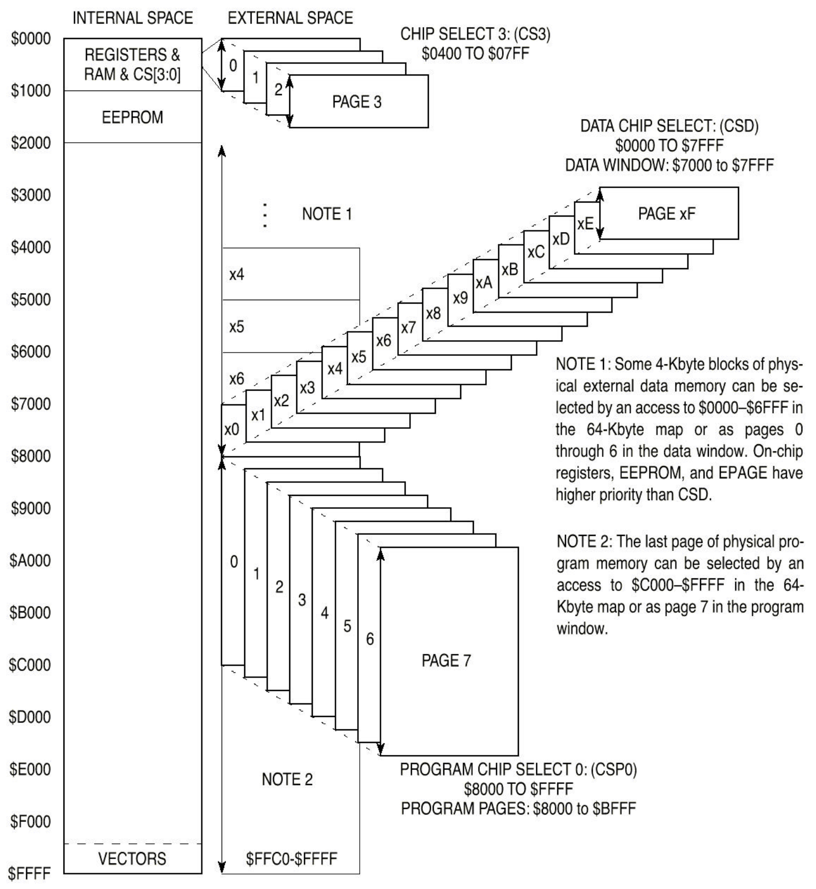

INTERNAL SPACE
EXTERNAL SPACE
$0000
$1000
$2000
$3000
$4000
$5000
$6000
$7000
$8000
$9000
$A000
$B000
$C000
$D000
$E000
$F000
$FFFF
REGISTERS & RAM & CS[3:0]
EEPROM
VECTORS
CHIP SELECT 3: (CS3) $0400 TO $07FF
0
1
2
PAGE 3
DATA CHIP SELECT: (CSD) $0000 TO $7FFF DATA WINDOW: $7000 to $7FFF
NOTE 1
x4
x5
x6
x0
x1
x2
x3
x4
x5
x6
x7
x8
x9
xA
xB
xC
xD
xE
PAGE xF
NOTE 1: Some 4-Kbyte blocks of physical external data memory can be selected by an access to $0000–$6FFF in the 64-Kbyte map or as pages 0 through 6 in the data window. On-chip registers, EEPROM, and EPAGE have higher priority than CSD.
0
1
2
3
4
5
6
PAGE 7
NOTE 2: The last page of physical program memory can be selected by an access to $C000–$FFFF in the 64-Kbyte map or as page 7 in the program window.
PROGRAM CHIP SELECT 0: (CSP0) $8000 TO $FFFF PROGRAM PAGES: $8000 to $BFFF
NOTE 2
$FFC0-$FFFF

Figure 4.1

example, if the user program were located in the first physical page of Flash, starting at $8000, the last few lines of startup code would set PPAGE=0, and jump to $8000.

A second issue is how to deal with paging and the generation of s-records in the assembler. If the Flash is to be programmed externally and subsequently plugged into the board's socket, the assembler would need to incorporate the page value tails on the paging architecture and memory expansion modes of the MCU. In this configuration, RAM is viewed by the MCU as a bank of 4K data pages, with the active page being selected by the MCU's DPAGE register. Flash memory is viewed by the MCU as a bank of 16K program pages, with the active page being selected by the MCU's PPAGE register. The example shown in Figure 4.1 is taken from Motorola's Technical Summary, and is for 64K RAM (data space) and 128K Flash (program space). A 128K RAM will consist of 0x1f 4K data pages, and a 512K RAM will have 0x3f 4K datapages. A 128K Flash will be divided into 0x7 16K program pages, as shown, while a 512K Flash will consist of 0x1f 16K program pages. The Extra Page space shown in Figure 4.1, corresponds to CS0-CS3, and is not implemented in Adapt812DX. However, the user can implement it, if desired, by decoding the signals present on connector H2.

4.41 External Memory Programming Issues

When the MCU comes out of reset in Normal Expanded Narrow mode, CSP0* is active, but program paging and A16 are not active. All address lines above A15 are pulled high, so the MCU will fetch the reset vector from the last two Flash physical address locations. Since paging is not yet on, CSP0* covers 32K of system memory (from $8000–$ffff), which is the last 32K (contiguous) block of Flash. Startup code should be put in the upper 16K half, so that when program paging is turned on, the re-definition of addresses will not disrupt program execution. (Program execution could be disrupted because, when program paging is activated, the lower 16K half [of the topmost 32K block of Flash] suddenly "jumps" to $8000–$bfff, accessible only when PPAGE=6 for a 128K Flash, or $1E, for a 512K Flash.) The startup code to be placed in the topmost 16K page would typically be the memory expansion register initializations. After that, the PPAGE register could be initialized to the desired value, and a jump to the beginning of the user program could be performed. For

such as logic analyzers and in-circuit emulators to begin the long hard road of diagnosing and correcting their software and/or hardware mistakes.

A much more sensible– and rewarding– approach is to start with something that works, and then add new features incrementally. The modular design of Adapt812DX gives you that starting point– hardware that works, and software that works. Now, if you build on that incrementally, each diagnostic step is small and manageable. And it will probably end up taking a lot less time, and costing a lot less money.

A useful debugging tool for program development is the serial communications interface (SCI). The SCI gives you a window on what's going on inside the microcontroller. Simple diagnostic messages, placed at strategic points in your evolving program, will be invaluable in debugging your software and hardware.

With the HC12, however, Motorola has added an even more powerful feature for debugging and development– the Background Debug Mode (BDM). This feature allows you to examine and modify locations and registers in your system while your program is running or suspended. It is implemented with a single-wire serial protocol, and requires a BDM interface pod to use with a serial port on your computer. Motorola makes a full-featured SDI pod, costing several hundred dollars, which is beyond the budget of most hobbyists, students, and many engineers Fortunately, there are some low-cost alternatives. Adapt912 from Technological Arts, and Motorola's 912EVB Evaluation Board both have a debugger/monitor program (D-Bug12) on-chip, which allow them to be run as a BDM interface, in POD mode. Another possibility is the BDM12, from Kevin Ross, which includes Windows95/NT-based software for easy debugging. All of these pods may be used for downloading s-records to EEPROM, and all are supported by ImageCraft's ICC12 68HC12 C Cross-complier for Windows. See Section 5 for contact information.

3.3 Downloading Your Code to Adapt812DX

Once you have assembled your code with no errors, you can download the resulting s-record file (*filename*.s19) to Flash using the on-chip firmware utility provided. Connect the supplied serial cable between connector J4 on your module and an available serial port of your PC (or Mac, Amiga, workstation, etc.). Use any terminal program, set it for the chosen serial port and 9600 baud.

Power up or reset your module with BOOT/RUN set to RUN, and the JB1 switch set to SGL. You should see a command menu in your terminal window. If your board has 512K memory devices, select **5**, and if your program does not use banked mode, select **N**. (For more on banked versus non-banked memory, see section 3.4.) To load your s-record file into Flash, first select **E** to erase the Flash. LED D1 will come on while the Flash is being erased. When it is finished, select **P** and use your terminal program's Upload or Send File function to send your s-record file. When it has finished, slide the JB1 switch to EXP and reset your module. If you wrote your code correctly, it should now be running. Always leave SW2 in the RUN position, otherwise the firmware utility may get erased from the on-chip EEPROM. (If you do erase it accidentally, however, it is easy to reload. See Section 4.1 for details.)

3.4 Expanded Mode Operation

Adapt812DX was created by merging two other Technological Arts products (Adapt812 Microcontroller card and Adapt812-MX1 Memory Expansion card) into a single compact board. As a result, references to memory expansion in documentation and filenames will sometimes be made to MX1. Adapt812DX is designed to run in Expanded Narrow Mode (8-bit data bus). For this reason, MODB is jumpered low, on JB1. The switch on JB1 allows MODA to be switched between logic 0 (single-chip mode) and logic 1 (expanded mode). Typically, you will set the switch to SGL to run the firmware Flash loader utility

MX1Flash to automatically calculate the correct address and page offset to place your code in the last two blocks of physical Flash memory. When you reset in expanded mode, your code will be located in the 32K block of Flash the 'HC812 accesses from 0x8000 to 0xffff.

4.3 Using SBASIC

As is the case with ICC12, SBASIC does not directly support banked memory on the 68HC12, so you will only have 32K Flash available for your code. The following are suggested compiler options for use with Adapt812DX. If **myprog.bas** is your SBASIC program filename, and **myprog** is the name you want the target assembly language file to be called, then type:

sbasic myprog /c8000 /v0800 /s0c00 /m6812 >myprog

After successful compilation, run as12 to create an s-record file, as follows:

as12 myprog

Then open a terminal window set for 9600 baud (no character delay). Reset the board in SGL mode, and the MX1 Flash Utility menu will appear. Type **5** if your board is a 512K version, and type **n** to set *non-banked* mode. Then type **e** to erase Flash, and choose **p** to program Flash. Select the terminal's ASCII transfer function, and download your **.s19** file. When finished, switch JB1 to EXP, and press RESET.

4.4 External Memory

As mentioned previously, external memory on Adapt812DX was designed to be used with the 68HC812A4's Data and Program memory page window system, in Normal Expanded Narrow Mode. Refer to your 68HC812A4 Technical Summary document for de-

User EEPROM. You may wish to use some EEPROM for storing calibration information or a serial number. You could erase EEPROM and use it for your application; however, you would lose both the bootloader and MX1Flash utilities, and have to re-load them when you need to make a code change in Flash. If 254 bytes is enough, use the EEPROM from $1e00 - $1efd. This block is not used by the utilities, and has been reserved for the user's applications.

4.2 Using ICC12 for Windows

Version 5 and earlier of ICC12 does not directly support banked memory on the 68HC12, so you will only have 32K Flash available for your code (ie. you will be running in Non-Banked Mode). Before compiling, set up the linker sections with 0x0800 for data (RAM), 0x8000 for text (code), and stack at 0x0c00. This will allocate the on-chip 1K RAM for both variables and stack, and the external RAM will not be used. (Note: to use external RAM, you would need to add code to your startup file to set the CSDE bit of the CSCTL0 register. Then you could locate the DATA section at 0x7000, for example. If you try to allocate the *stack* to external RAM, however, watch out! ICC12 executes a JSR instruction to run your startup code. Since the external RAM is not yet enabled, the RAM to implement your stack has not yet been enabled, so your code will never return from the startup routine.) After 'compling to executable', download the resulting s-record file using the terminal window. Open the terminal window and set communication options for 9600 baud and no character delay. Reset the board in SGL mode, and the Flash Utility menu will appear. If you have a 512K board, type **5** to set 512K mode, and type **n** to set *non-banked* mode. Then type **e** to erase Flash, and choose **p** to program Flash. Select the terminal window's ASCII download option, choose the **.s19** file you wish to download and click OK. When downloading has finished, switch JB1 to EXP, and press RESET. By selecting non-banked mode, you have told

(**mx1flash**), then erase the Flash and, finally, load an s-record file into Flash. Then you will switch to EXP, and reset the board, to have the microcontroller execute the code that is now in Flash.

There are a few points to keep in mind when writing code for an expanded mode system. The 68HC812A4 provides several internal registers which control and define the various possible configurations of expanded memory. The simplest code can ignore these registers, and consider the memory map to be limited to 64K. This will be referred to as *Non-Banked Mode*. In this case, 32K of the external RAM will be accessible from $0000 - $7fff (when enabled via CSCTL0 and CSTL1 registers), and 32K of the external Flash will be available from $8000 - $ffff (since CSP0 is enabled by default). The exceptions are those internal 68HC812A4 resources which appear in these areas– they have priority over external memory. Note also that, in non-banked mode, the physical address actually accessed in Flash and external RAM is the topmost 32K block, since the default logic level on address lines above A15 is high. If you're using a Flash Programmer to load code (instead of doing it in-circuit), you will have to specify the appropriate offset in the programmer's console.

To take full advantage of the memory capacity available on Adapt812DX, however, some of the MCU's registers must be initialized at the beginning of your code. For recommended register values, have a look at the demo program source code, found in the **adapt12/mx1** subdirectory of your Starter Package disk. The relevant files are **mx1demo.asm** for 128K boards, and **mx1demo5.asm**, for 512K boards. While the demo program is small and fits into a single page of Flash, a larger program could have subroutines located in other banks of Flash. The only change necessary would be to use the CALL instruction instead of JSR, specify the page number where the subroutine is loacted, and change the RTS to RTC (return from call) at the end of the subroutine. For further details on using expanded modes, refer to the 68HC812A4 Technical Summary data sheet (sections 6.4 and 8).

4 REFERENCE

4.1 How On-chip EEPROM Programming Works

The ADAPT-812 module uses on-chip 4K EEPROM for two purposes. A small (256-byte) bootloader has been installed in the 68HC812A4 EEPROM by Technological Arts. It can be used to load any s-record file into the rest of EEPROM, via the RS232 serial port, at 1200 baud. To use this function, reset the board with SW2 at BOOT and the JB1 switch at SGL. Now when you send an s-record file at 1200 baud, the bootloader will erase all but the protected block of EEPROM, and then program each byte of the s-record file into EEPROM. LED D1 will flash once for each s-record loaded.

How it works: This bootloader program runs whenever the chip is powered up or reset in single-chip mode, and looks at the state of PC6 (set via SW2) to decide whether to run a user program in EEPROM or to initiate downloading (BOOT) mode. If the pin is pulled low, the program loads an s-record file via the serial port, and "burns" it into EEPROM. If the pin is open (pulled high by the internal pullup resistor), control passes to the user program. This event is transparent to the user. The only limitation is that EEPROM Block3 ($fe00 to $feff) is not available to the user. During downloading, the user reset vector is automatically intercepted and stored in a pseudo-reset-vector location established by the bootloader. It is here that the bootloader looks when the chip is reset in RUN mode, passing control to the user's program based on this vector. If the vector has not yet been initialized (ie. contains $ffff), the bootloader stops in an infinite loop, flashing LED D1 to indicate an error condition.

It is virtually impossible to accidently erase the bootloader from the EEPROM, unless your program implements code to erase or program on-chip EEPROM, and it clears the Block Protect bit for the upper "boot block". If for some reason you *do* erase the bootloader, you will need to use a BDM pod to re-load it. However, if you have only erased the Flash Utility (much more easily done), the procedure for restoring it is quite simple.

4.11 Re-loading the Flash Utility

On Adapt812DX, a "user" program has already been loaded into EEPROM. It is the Flash utility mentioned previously (called **mx1flash**), which allows you to erase Flash and load s-records into it, among other things. If you have inadvertently erased **mx1flash** from EEPROM, you can easily re-load it as follows. From DOS, use one of the batchfiles included on the Starter Package disk. Use **p8s1.bat** for COM1, or **p8s2.bat** when using COM2. (If you are using another COM port, you will have to edit the batchfile to reflect this.) For example, to download **mx1flash.s19** to Adapt812DX EEPROM via COM2, at the DOS prompt, you would enter:

p8s2 <path>mx1flash

where **<path>** is the directory path to where **mx1flash.s19** is located (if it is not in the same directory as **p8s2.bat**). Then follow the on-screen instructions. If you experience problems with the DOS batchfile on a Windows95 machine, you may need to edit the batchfile to add "\dev\" to each occurence of the com port path (eg. **copy %1.s19 \dev\com2**).

Instead of using the DOS batchfile described above, you could use any terminal program to perform an ASCII transfer of **mx1flash.s19** to the board via the serial port. Just set up the terminal program for the appropriate COM port and a baud rate of 1200. Switch to BOOT, press RESET, and send the file. When you are done, change the terminal to 9600 baud, switch the board back to RUN mode, and press RESET again. The Flash Utility menu will be displayed.

Note that, when using the EEPROM bootloader, as described in this section, you should always keep the chip in single-chip mode (ie. JB1 switch set at SGL). If you reset or power-up with the JB1 switch at EXP, the 4K on-chip EEPROM will default to $1000 instead of $f000 (refer to Motorola's 812A4 Technical Summary for details), and will no longer be in the vector space of the HC12. Instead, your external Flash will be in the vector space, and the HC12 will attempt to run whatever is there.