Order this document
by AN1753/D

Motorola Semiconductor Application Note

AN1753

Implementing a FLASH Memory System in an MC68HC711E9 Design

By Joe Haas
Applications Engineering
TSG Body Electronics and Strategic Industrial Division
Austin, Texas

Introduction

FLASH technology offers several advantages to a M68HC11 microcontroller design. Field updates, lower-power consumption, and increased memory densities are but a few of the potential benefits of incorporating external FLASH memory as a firmware/data media. However, several drawbacks can provide significant obstacles to implementation on a M68HC11 system.

Most of the difficulties with a FLASH memory implementation derive from the fact that FLASH requires an algorithm to program data. Many FLASH devices cannot perform read operations during a programming cycle, so the algorithm must be maintained in a memory device physically separated from the FLASH to be programmed. Thus, an additional memory device is needed to hold the programming algorithm. Typically, this device will occupy the upper slot in the M68HC11 memory map to maintain the interrupt vectors. This necessitates the need for a jump table to route these static interrupt vectors which increases interrupt response latency. Any interrupts that are desired for the programming algorithm also must be subjected to an arbitration

MOTOROLA

mechanism which adds both latency and documentation overhead for firmware development and maintenence.

Bootstrap mode of the M68HC11 Family could be used to upload a FLASH programming algorithm, but this also has several drawbacks. For example:

- The MCU RAM is limited in size, so it cannot hold an algorithm of any complexity. External RAM may be available, depending on the design.
- If external RAM is non-volitile, its use as a temporary program space could corrupt previously stored data.
- Bootstrap mode requires a special data sequence which is not compatible with manual download from a terminal and thus requires the use of a special boot loader running on the host system.
- Depending on the MCU crystal frequency, the bootstrap serial communication parameters can specify an odd serial data rate which isn't supported by a standard boot loader.

For most applications, any one of these difficulties, much less all of them, could easily eliminate the possiblity of implementing a FLASH device.

This application note describes a single board computer (SBC) design which uses a FLASH device as its main program/data storage media. The emphasis is on the hardware and firmware techniques used to implement the FLASH programming system as well as its impact on firmware development. Also, an example of a retrofit design is included to illustrate how these techniques are modified to convert an existing EPROM-based design to FLASH.

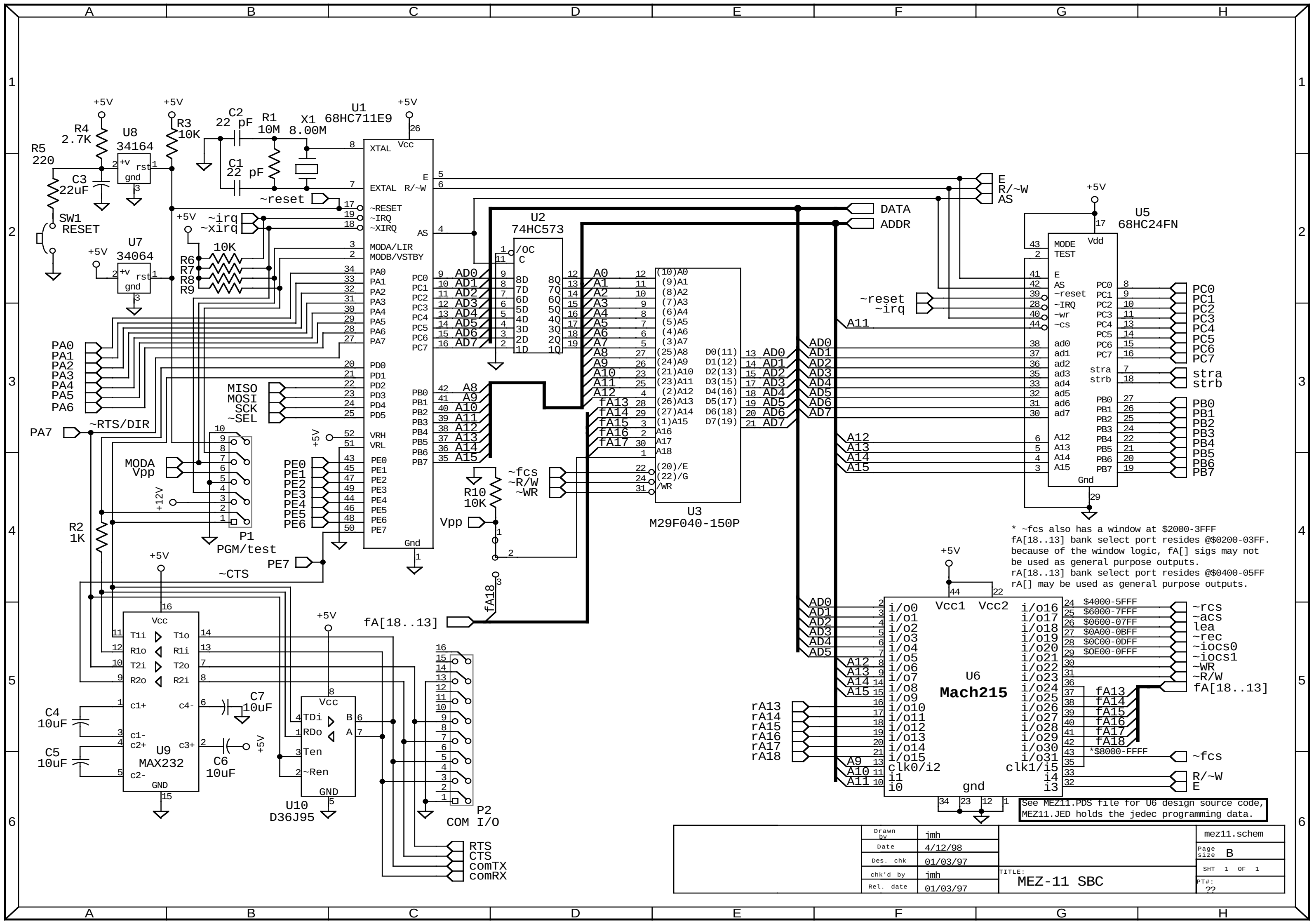
System Requirements

While many aspects of the SBC design presented here can be varied to fit the needs of a particular application, there are some features that must be present to support the FLASH system as described here. Typically, FLASH data originates from a host system (for instance, a PC) which necessitates the need for a programming host connector. The host port connector must bring out the MODA/MODB pins, as well as the SCI (serial communications interface) pins (PD0/PD1), ~XIRQ (optional), and RESET. If a FLASH device is used that requires an external V_{PP} source, this signal also is required on the host port. This requirement depends on the system design. The V_{PP} source can be placed on the target system, but will represent a fixed cost in the final design that will erode per-unit cost margins. Moving this source off target onto a host programming interface can reduce overall costs if the number of target systems is significantly higher than the number of programming systems. Finally, a small segment (four to six bytes) of MCU EEPROM must be reserved for use by the programming firmware.

Providing the ~XIRQ signal to the host system is optional. Doing so will allow in-circuit factory programming of the 68HC711E9 EPROM. The same host system can be used for both FLASH and MCU EPROM programming operations which support a factory environment where virgin MCU and FLASH devices may be placed on the target circuit board for in-circuit programming as part of the test procedure. In this case, special considerations should be given to the ~XIRQ circuit, if it is used in the target design, as this signal is subjected to voltage levels in excess of 5 volts during MCU EPROM programming operations.

Host Interface Description

Figure 1 shows the schematic for the SBC system used in this example. The host port connector is implemented with a 10-pin dual row connector (P1) which connects to a programming interface card. Most of the signal connections are straightforward except for the RXD (PD0) signal. Here, a resistor is used to passively switch the RXD signal when the programming interface is connected. This allows the target system to share the serial port between the FLASH programming host and another target resource.



* ~fcs also has a window at $2000-3FFF
fA[18..13] bank select port resides @$0200-03FF.
because of the window logic, fA[] sigs may not
be used as general purpose outputs.
rA[18..13] bank select port resides @$0400-05FF
rA[] may be used as general purpose outputs.

See MEZ11.PDS file for U6 design source code,
MEZ11.JED holds the jedec programming data.

Drawn by	jmh
Date	4/12/98
Des. chk	01/03/97
chk'd by	jmh
Rel. date	01/03/97
TITLE:	MEZ-11 SBC
	mez11.schem
Page size	B
SHT	1 OF 1
PT#:	??

Figure 1. SBC Schematic

With the programming module removed, the RXD signal passes from the target RS-232 receiver (U9) to the MCU (U1) via resistor R2. The only consideration is to keep the resistor value low enough that it will not interact with the input capacitance of the MCU to form a low pass filter while providing enough impedance that driver contention is eliminated.

To eliminate filter effects, the following must be satisfied:

$$r < 1/(2\pi 3fc)$$

Where r is the value of R2 in ohms, f is the maximum desired baud rate (bits per second), and c is the total capacitance of PD0 in farads (input and stray capacitance). 3f is used here to account for the third order harmonics which can be a significant component of a square wave signal. At 115 Kbaud, this value calculates to approximately 80 kΩ – in practice, values of 1 K to 10 K are recomended.

The only difficulty with this mechanism involves the nature of the target resource; it must be able to tolerate the serial traffic that is present during programming without producing unknown or undesirable results.

NOTE: *The FLASH programming baud rate usually is not related to the target resource baud rate. This can compound the difficulty due to the fact that a baud rate mismatch can cause known data streams at one baud rate to appear as pseudo-random data streams at another. For situations where this is a problem, a more involved switching mechanism may be required.*

As mentioned earlier, a specialized programming interface card can offer a convenient and cost-effective host interface. The programming interface card used in this design is shown in **Figure 2** and features an RS-232 transceiver, SCI loop-back switch, V_{PP} voltage converter, and reset controller. The reset controller is implemented with an MC68HC705K1 microcontroller which contains a simple program that interprets the control switches and manipulates the target MCU signals appropriately.

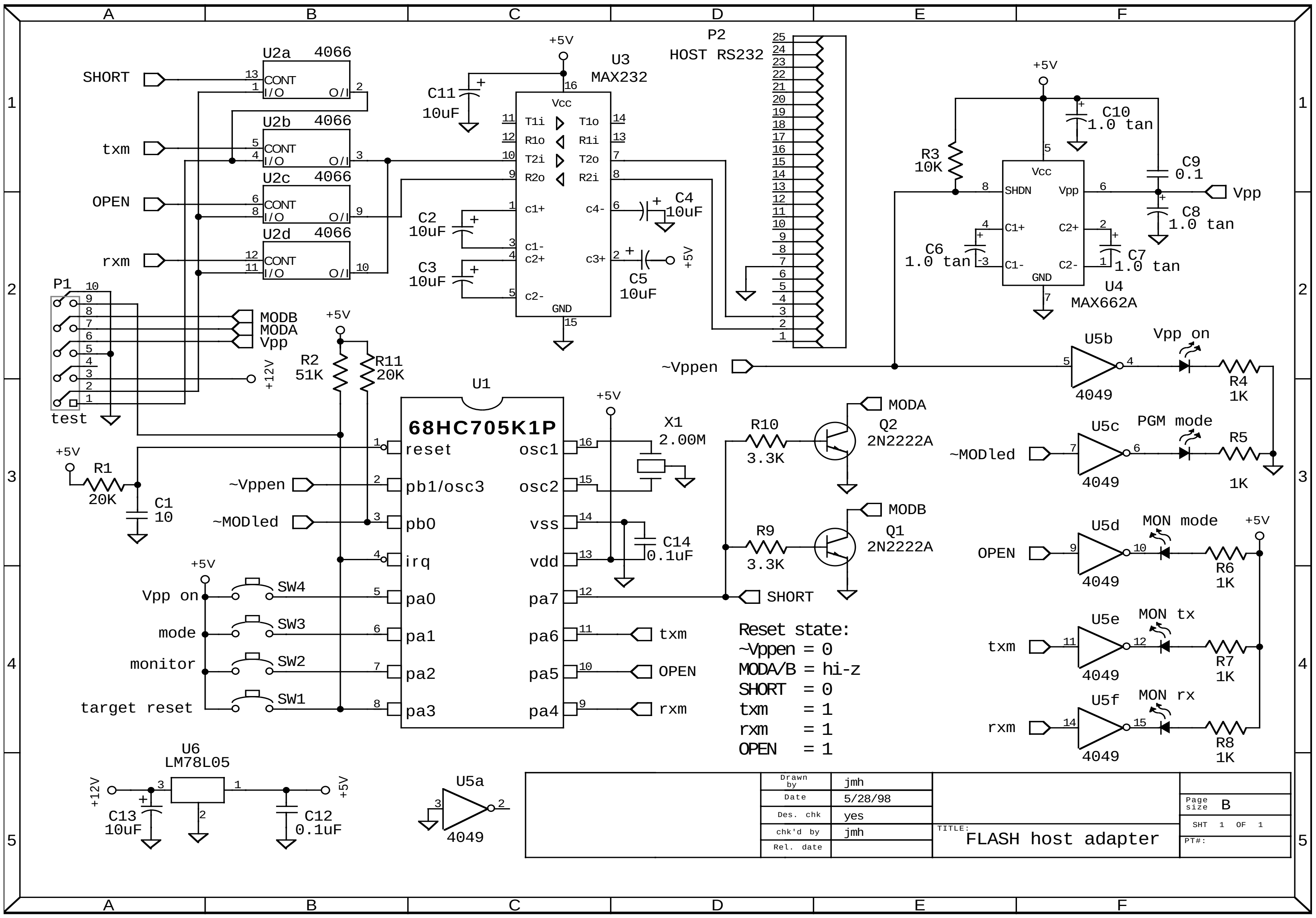


Figure 2. Host Programming Adapter Schematic

The following describes the button functions:

- **Target RESET (SW1)** — Connected to the target reset I/O (input/output) signal. When activated in boot mode, the target PD0 and PD1 signals are briefly "shorted" to loop-back the MCU SCI activity. In normal mode, the target system is simply reset (no loop-back).
- **MONITOR (SW2)** — When in the normal mode, this function toggles the host RXD source between the target PD0 and PD1 signals. A third state connects the host RXD/TXD to the target PD0/PD1 (respectively). This function is not active in bootstrap mode.
- **MODE (SW3)** — This toggles the target system between normal mode and boot mode. The MODA/MODB control output is high-Z for normal mode and V_{OL} for bootstrap mode.
- **V_{PP} ON (SW4)** — In bootstrap mode, allows the V_{PP} source to be toggled on and off. This function is not active in normal mode.

The monitor function uses the module's loop-back switch to selectively route the target RXD and TXD signals (MCU PD0/1) to the host serial port when the target is in normal mode. This allows the host to easily trace serial data from either side of the target system's SCI interface which can be useful as a system level debugging tool.

Program Listings — **Host Interface Reset Controller — Listing 1** shows the reset controller assembly source code. The code structure is relatively straightforward, consisting of the initialization, main loop, and ~IRQ service routine. Initialization begins at the ENTRY label and consists of setting the port A and port B DDRs (data direction register) and initializing the port outputs. The main loop begins at the TOP label which continuously polls the push button inputs. If a button push is detected, the firmware loops until it is released. Switch edges are debounced by a 20-ms timer to help prevent false signals due to switch noise. After switch release is detected, the firmware processes the selected button's function.

The interrupt routine at IRQI traps falling edges of the reset button. Since the reset button is also connected to the target reset signal, resets originating with the target system will be trapped also. This way, the reset sequence is handled properly, no matter what the reset source.

System Memory Map

Figure 3 illustrates the memory map of the SBC that is being demonstrated in this application note. In addition to the normal MCU resources (on-chip RAM, I/O registers, and EEPROM), there are a FLASH memory device, two external RAMs, and some external registers. The FLASH and RAM slots can contain devices of up to 4 Mbit (512 K x 8). Since this far exceeds the 64-K addressing space of the M68HC11, a bank switching scheme is used to page through the large density devices using I/O signals to resolve the upper physical address signals.

While the RAM device(s) simply use direct replacement to resolve the upper address signals, the paging scheme for the FLASH device is a bit more complicated.

First, we desire a fixed portion of the FLASH memory to be visible in the upper portion of the MCU memory map ($8000 through $FFFF). This will hold the MCU interrupt vectors as well as up to 32 Kbytes of kernal ROM, which is intended to hold the core of the SBC operating system. Intuitively, this window should reside at the top of the FLASH ROM physical address space where the banked address signals A[18:15] are [1111]. This forces any access to MCU addresses in the range of $8000 to $FFFF to access FLASH memory at physical addresses $78000 through $7FFFF regardless of the current bank setting.

In addition to the kernal window at $8000–$FFFF, a separate 8-Kbyte window is used to access the remainder of the FLASH device at MCU addresses $2000 through $3FFF (referred to as "mirroring"). Addresses in this range will access the full physical device map from $00000 to $7FFFF, depending on the combination of A[12:0] and the bank selects. The A[18:13] vector for the mirror window is determined only by the value of the bank register.

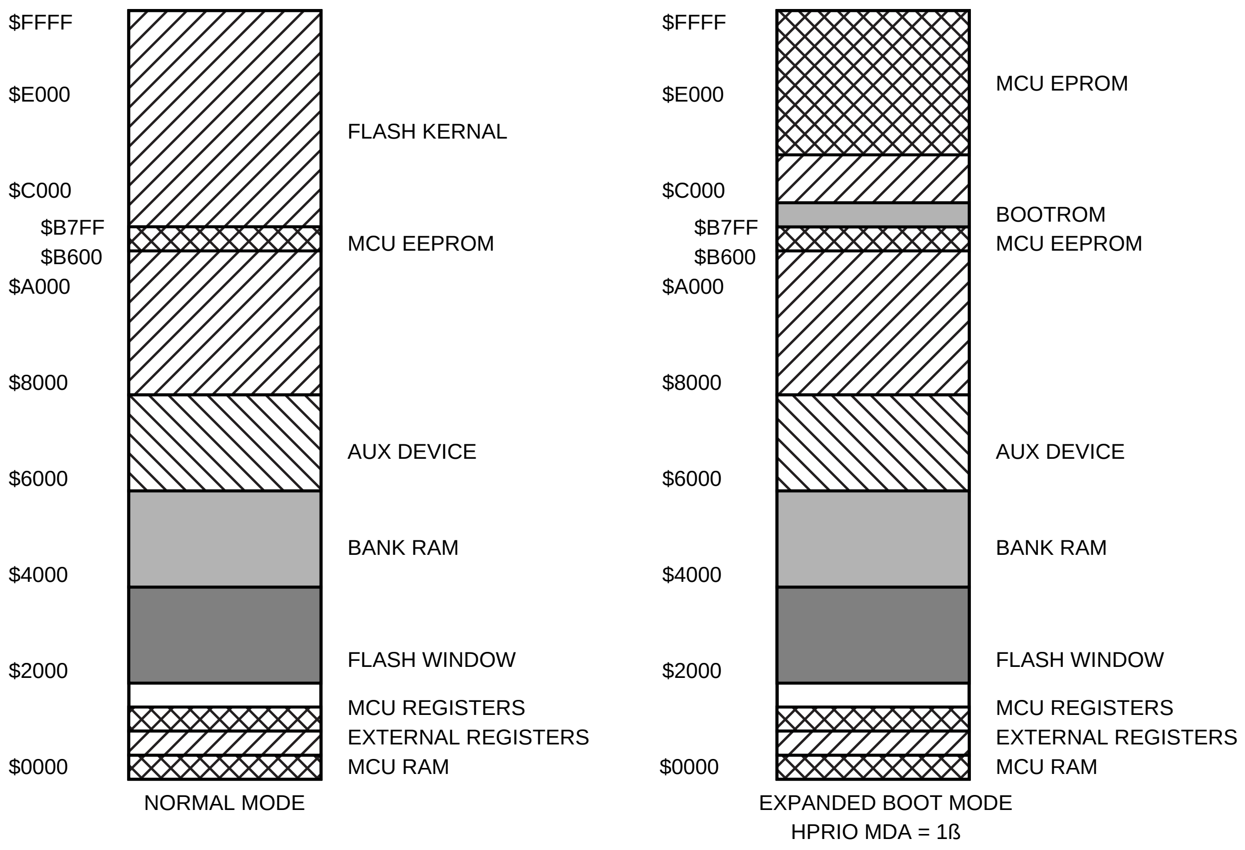


Figure 3. Target System Memory Map

A logic circuit is used to combine the MCU address signals and the bank select signals into a physical address vector that can be connected to the FLASH device. This logic circuit needs to force the physical address[18:15] to [1111] for MCU $8000–$FFFF and pass the bank selects to the physical address [18:13] for accesses at MCU $2000–$3FFF. Since the kernal window takes the upper half of the memory map, the A15 signal can be used to force the physical address source. Since the resulting physical address is connected exclusively to the FLASH device, only A15 is needed to switch between the two FLASH windows.

Since the logic required to implement this project is relatively simple, a behavioral level schematic was sketched using off-the-shelf logic devices. The schematic's sum-of-products representations were then extracted manually as shown in **Table 1**.

Table 1. SBC Memory System Logic Equations

fA[18:15]	= ~(~lat[5:2] * ~A15)
	= lat[5:2] + A15
fA[14:13]	= ~(~(lat[1:0] * ~A15) * ~(A[14:13] * A15))
	= ((lat[1:0] * ~A15) + (A[14:13] * A15))
~fcs	= ~(~(~(~A15 * ~A14 * A13) * ~A15) * E)
	= ~(A[15] * E + ~A[15] * ~A[14] * A[13] * E)
~rcs	= ~(E * ~A15 * A14 * ~A13)
~acs	= ~(E * ~A15 * A14 * A13)
lef	= ~A15 * ~A14 * ~A13 * ~A12 * ~A10 * A9 * E * ~R/W
~ref	= ~(~A15 * ~A14 * ~A13 * ~A12 * ~A10 * A9 * E * R/W)
ler	= ~A15 * ~A14 * ~A13 * ~A12 * A10 * ~A9 * E * ~R/W
~rer	= ~(~A15 * ~A14 * ~A13 * ~A12 * A10 * ~A9 * E * R/W)

Table 1 forms the basis of the HDL (hardware definition language) file used to generate the JEDEC (Joint Electron Device Engineering Council) fuse map for an FPGA (field programmable gate array). (The complete HDL design file appears in **Program Listings — FPGA HDL Listing — Listing 2**.) The fA[18:13] vector represents the physical FLASH address. (A[12:0] are obtained directly from the MCU.) The use of an FPGA is not required, but it offers many advantages over discrete logic implementation. As can be seen in **Table 1**, several logic devices would be required to implement the logic functions shown. This greatly increases the component count and board space required to implement the design, problems which can be eliminated by using the appropriate FPGA.

Programmer Core

With the key aspects of the target design frozen, consider now the techniques needed to use the FLASH programming system. The key to the FLASH system described here lies in the use of some subtleties of the 68HC711's bootstrap mode.

The key bootstrap mode features of interest are:

- When reset is in bootstrap mode, the MCU EPROM is forced on temporarily.
- While waiting for a download to begin, a serial break or ASCII NULL character received at the SCI will force the MCU instruction fetch to jump to EEPROM at $B600 and begin executing code there.
- When reset in boot mode, the MCU sends a break to the SCI.

For this application, the FLASH programming firmware is placed in the MCU EPROM, which is off in normal mode, allowing the programming firmware to be "hidden" from the target application. To the application firmware developer, the system looks like a 68HC711 in expanded mode, with an extended array of memory resources. It is only necessary that the developer be concerned with these two FLASH system requirements:

- The EEPROM locations at $B600–$B605 are reserved for FLASH (holds jump to MCU EPROM and three FLASH personality data bytes).
- FLASH uses MCU RAM during programming mode.

The latter is important only if the system must use battery maintenance of MCU RAM for non-volatile storage. Since there are sufficient resources for ample external NVRAM, this should not be an issue in most designs.

FLASH Operation

The first step in initiating the FLASH program mode is to connect the programming interface and place the system in programming mode. This will reset the target MCU in bootstrap mode and set the loop-back switch to the MCU. When reset in bootstrap, the M68HC11 Family will issue a break to the SCI subsystem. By looping back the serial break that originates from the 68HC711E9, the MCU is forced to jump to $B600 (the start of EEPROM). A pre-placed JMP instruction at $B600 (programmed along with MCU EPROM) allows the system automatically to execute MCU EPROM code with no host intervention. A timer on the reset controller switches the loop-back to feed-through so that commands and data may be exchanged between the host and target systems after the reset sequence is complete.

This topology provides a scenario for getting the MCU to execute ROM code out of a bootstrap reset. Now the ROM code must manipulate some additional resources to get the system into a mode which will allow FLASH programming to occur. The ROM code must:

- Run the FLASH command processor
- Turn on MCU EPROM
- Turn on expanded mode (set MDA bit in HPRIO)

While in bootstrap mode, the expanded mode cannot be enabled until the MCU EPROM is turned on, which requires a subsequent MCU reset to take effect. This step could be performed automatically by the MCU EPROM code but is probably best left as a host command function. To illustrate why, consider that the MCU EPROM must be turned off at the completion of the FLASH programming session. A programming session can consist of several data upload operations, depending on how many devices are to be programmed and how their data is segmented. This can make it difficult for the target system to determine when the host is actually finished. The best scenario is for the host to explicitly command the target to turn off EPROM, with the same being true for EPROM on.

The full reset sequence for a programming session would then proceed this way:

- Reset in bootstrap mode
- Host commands EPROM on

- Reset again (still in bootstrap mode)
- Perform programming operations
- Host commands EPROM off
- Reset in normal mode

FLASH Algorithm

The bulk of the FLASH algorithm was derived from a simple command line-based EPROM programmer written for the M68HC11 (reference 1). This code uses a command and parameter parsing system to perform device selection, programming operations, status, and download and upload protocols, etc. The original device selection topology was kept such that each device in the memory map appears as a separate selectable device to the algorithm.

The command lines are gathered by the SCI in interrupt mode, which buffers incoming characters until a <cr> ($0D) is received. A system flag is then set which signals the command parser to examine the buffer for valid command/parameter combinations.

<XON>/<XOFF> handshaking is also supported to control ASCII upload/download transfers. Handshaking is important because the programming sequence can take up to several milliseconds and the buffer must be frozen in that time to ensure data integrity until all the data has been transferred to the target device.

The heart of the data transfer system was adapted from the BUFFALO S-record transfer code. This code has been modified to support buffered data, as well as the xmodem transfer protocol. The xmodem transfer protocol (aka the Christiensen Protocol) transfers Motorola S-record data in 128-byte blocks and features flow control and error detection with retries. While S-records feature a line-by-line checksum which can be used to validate the data integrity of the line, typical ASCII transfer protocols do not provide for error detection or retries so the only recourse for a checksum error at the target is to discard the entire line and continue to the next line which requires another pass to fully program the device.

A checksum error in the xmodem protocol causes the data block to be resent (up to a pre-determined maximum retry count). In this way, the system can not only detect errors, but also can recover from them. If the original data is error free, there is a measure of assurance that xmodem will transfer the data so that it is received in its entirety.

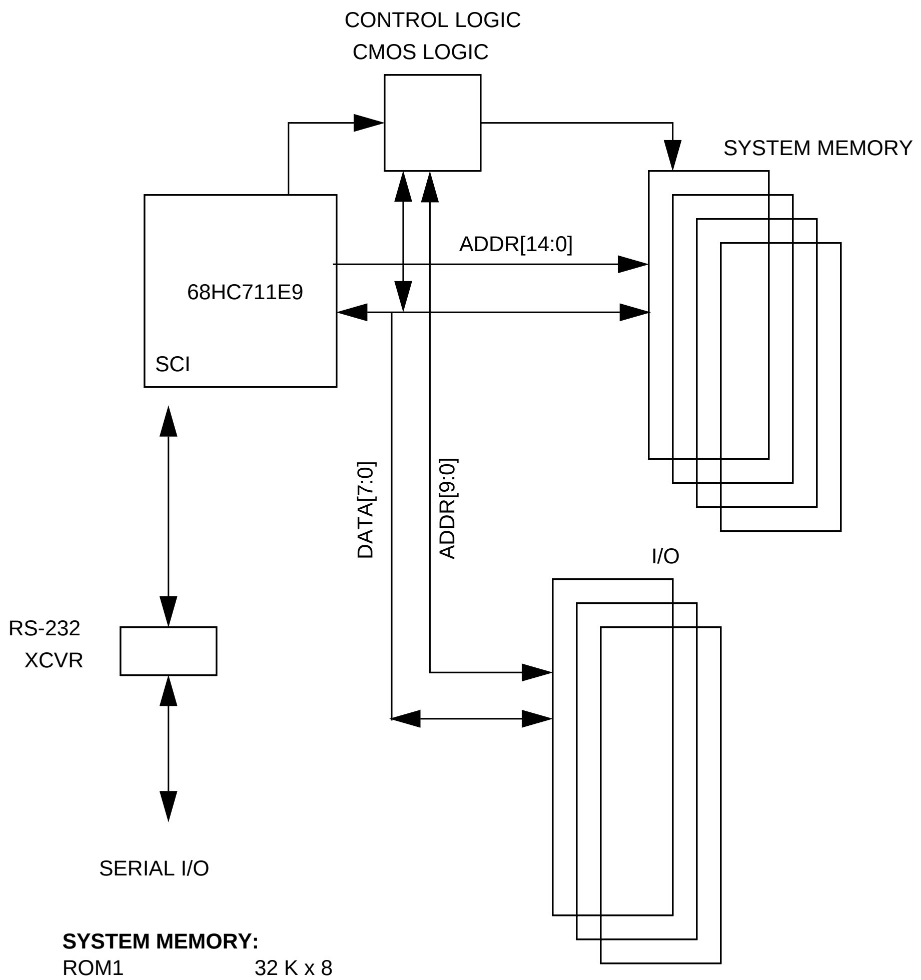
Two subroutines are used to access the target devices:

- REDbyt
- PGMbyt

These routines use the (X) index register and a bank register to point to a physical device address which is provided with the upload data from the host using Motorola S28 or S37 record formats. The BANK command allows manual selection of 64-K segments so that the Motorola S19 format can be used. REDbyt and PGMbyt call the address parser subroutine addrFLA to resolve the bank select address bits for the selected device and present them accordingly. The index register is then converted to an MCU logical address which points to the desired physical address. Once this procedure is complete, the MCU can present indexed read/write sequences to the target device to perform the desired operation.

A personality system is used which allows the device to be defined for the programming algorithm. This is important because the algorithm must distinguish between FLASH, NVRAM, shadowRAM, and EEPROM technologies. While most devices allow data simply to be read in programming mode, some require a pre-command before data can be presented to the device. To accomplish this, each device in the system has its own read/write subroutine which handles any required command sequences.

Retrofit Example



SYSTEM MEMORY:	
ROM1	32 K x 8
ROM2	128 K x 8 (16 K WINDOW)
RAM1	8 K x 8
RAM2	2 K x 8
I/O:	
MCU	PORTS A, D, E (22 BITS)
EXTERNAL	3 VIA LSI ICs (48 BITS)

Figure 4. EPROM-Based Design Example

Figure 4 illustrates the block diagram of an MCU-based system that uses EPROM memories for code and data storage. It is desired that this system be converted so that FLASH devices may be used, but only minimal hardware modifications are allowed due to cost constraints. A

32 K x 8 EPROM (U3) is the primary code memory and is located at the top of the memory map as shown in **Figure 5**. A 64 K x 8 EPROM (U4) is used to hold secondary code and data for the application. Since these two devices would exceed the available MCU address space, U4 is bank switched in a 16-K window with provisions to allow expansion to a 128 K x 8 device. Since only three bank select lines are required, they are derived directly from MCU I/O pins.

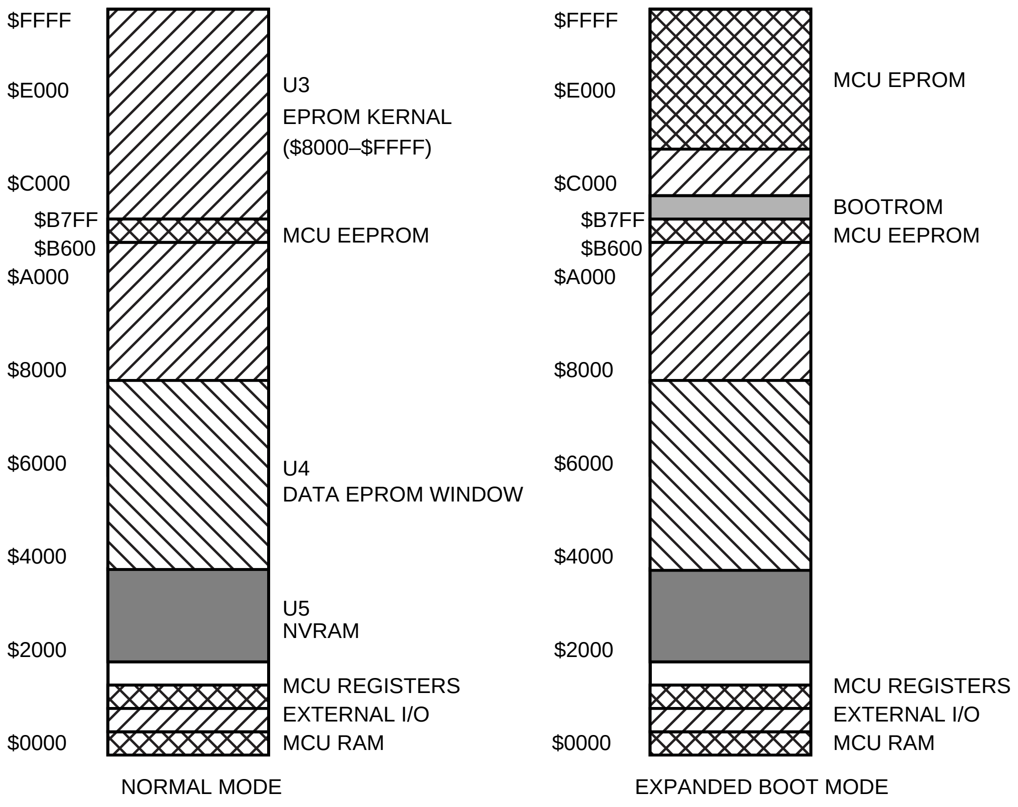


Figure 5. Retrofit Example System Memory Map

The primary difference between this system and the SBC presented earlier lies in the fact the U3 kernal memory is only accessible via a single window in the memory map. Furthermore, several MCU resources also occupy this memory space during bootstrap mode, including the MCU EPROM, which provides serious hurdles to implementing the FLASH system.

The bootROM and EEPROM conflicts can be addressed in firmware. For the EEPROM, this part of the FLASH array will never be accessible anyway, so no system code or data will ever need to be written to this area. For reliability reasons, it is necessary to write those FLASH locations to $00 during the FLASH pre-erase cycle. Since the EEPROM cannot be turned off without an MCU reset, the only choice is for the firmware to blind write these locations. This can be accomplished because the M68HC11 Family drives external address and data during writes to internal resources. This allows FLASH programming commands to be written to the device, but prevents reads from the FLASH because the internal resource takes priority. While not optimal, this will at least serve to improve reliability by reducing over-erase stress on these memory cells.

Since the bootROM overlaps memory that is accessible in the MCU’s normal mode, it is desirable that the system be able to read/write/verify these memory locations. This can be done by briefly turning off the bootROM using the RBOOT bit in the HPRIO register. It is important to suspend all interrupts for this process and that ~XIRQ must not be enabled. This is because the bootROM holds the interrupt vectors for the bootstrap mode and an interrupt which occurs while the bootROM is off will result in a false vector fetch, causing unrecoverable system disruption. Since the FLASH algorithm used here depends on interrupts, they can not be disabled entirely, but instead must be disabled only during the time that the bootROM is off.

The MCU EPROM conflict is more difficult to overcome as the EPROM cannot be turned off and on like the bootROM (a reset is required). Since it covers a large and important area of the memory map, it also can not be ignored, and blind programming is not acceptable as there is no way to ensure data integrity without overstressing the FLASH core. The only viable alternative is to use the external RAM (U5) to hold the FLASH algorithm so that the MCU EPROM can be turned off.

Normally, this would offer another difficulty due to the fact that the RAM at U5 is an NVRAM and holds vital system data that would have to be saved and restored by the host system. In this case, however, the RAM at U5 is a shadow RAM which means that this device keeps non-volatile data in a shadow EEPROM array separate from the SRAM array (all on the same die). Special read sequences initiate store-and-retrieval operations which copy data between the U5 SRAM and EEPROM. This means that the FLASH algorithm can be copied into the SRAM array without disturbing the shadow EEPROM data which eliminates the need for the host to save and restore the RAM data. When the programming operation is complete, the original U5 data is restored automatically on system power-up.

Because the FLASH algorithm must be copied to SRAM, an additional step is required in the reset sequence.

- Reset in bootstrap mode
- Host commands EPROM on (set ROMON bit in CONFIG)
- Reset again (still in bootstrap mode)
- Copy algorithm to U5 RAM
- Turn EPROM off; clear ROMON bit in CONFIG
- Reset again (still in bootstrap mode)
- Perform programming operations
- Reset in normal mode (cycle target system power to restore U5 data)

At this point, the system behaves the same as described under the SBC system. The host may read, write, and erase the U3 and U4 devices as needed using the commands provided by the FLASH algorithm operating in the U5 RAM.

Retrofit Hardware Changes

The required target hardware changes to the retrofit system are:

- Expand U3 and U4 sockets to 32 pins and connect ~WE and V_{PP} signals.
- Add programming mode connector. This system already has an external serial port connection, so the mode connector only requires the MODA/B, $\overline{\text{RESET}}$, and V_{PP} signals. (+5 V is also added to support the programming mode interface.)

As is readily apparent, the hardware changes are minimal. The costs associated with these changes will focus on the minimal engineering effort to modify the PCB layout and the re-tooling costs from the PCB vendor. In this case, the increased component costs are less than 0.5 percent of the finished product cost.

While the modifications to the target system are a relatively small part of the overall product cost, the addition of the programming adapter results in a noticeable cost increase. Even so, these costs can be minimized as the design of **Figure 6** illustrates. This is a simple example of a host interface card which uses a simple toggle switch to select the mode and enable V_{PP} and an inexpensive DC-DC converter to develop the FLASH V_{PP} voltage.

NOTE: *The serial loop-back switch is not present in this module. Since cost is an issue, the loop-back switch was omitted, which requires the host to send the break signal or ASCII-NULL after each reset.*

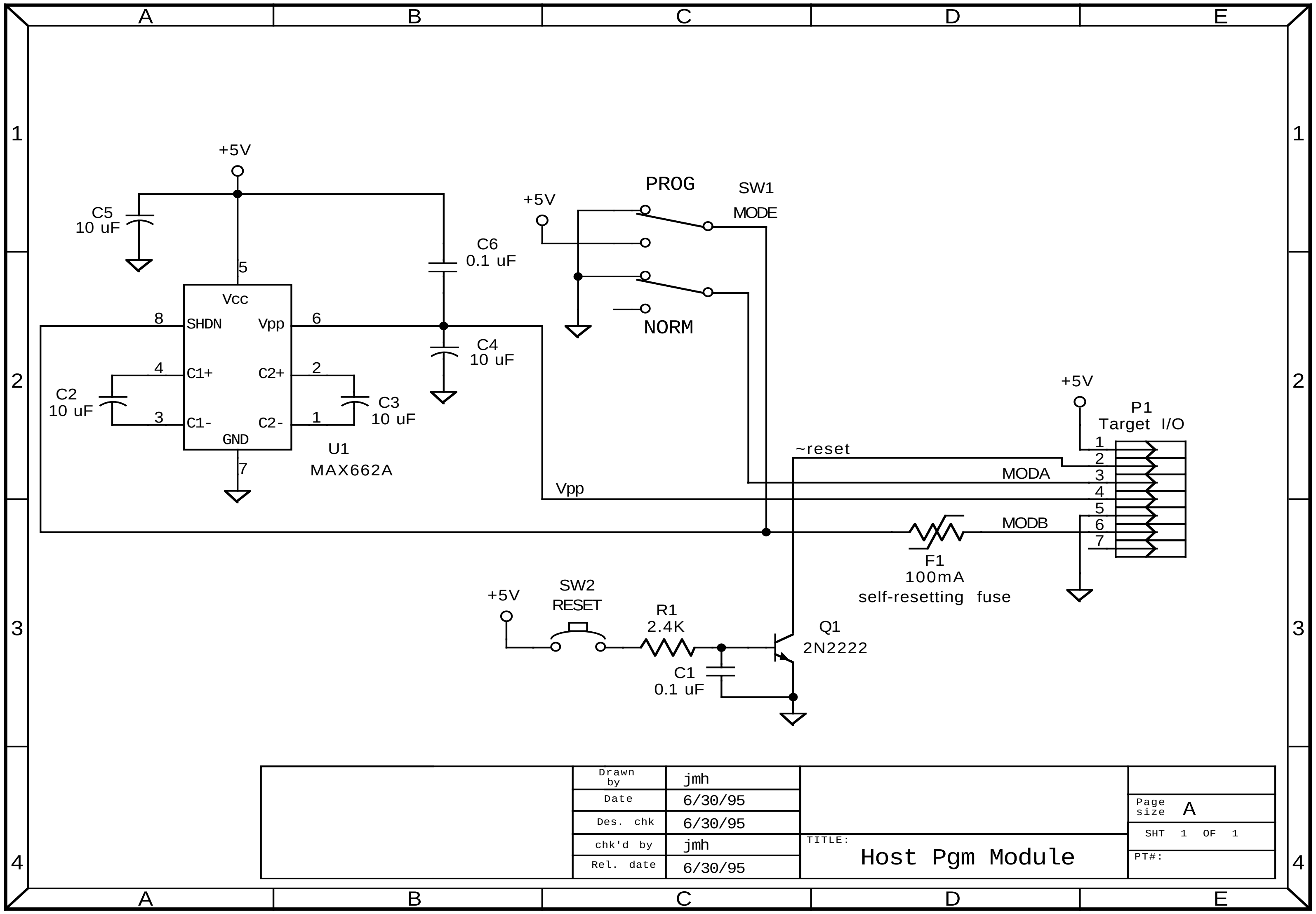


Figure 6. Simplified Host Programming Adapter Schematic

While this retrofit example illustrates a means to implementing FLASH on an existing design with minimal hardware changes, some compromises are necessary to meet that end. Some systems may not have a RAM array available as temporary algorithm storage. In those that do, it may not be feasible to use a shadow RAM because these devices are available only from a limited number of suppliers. For these situations, it may be necessary to implement the techniques described for the SBC design which would require additional hardware modifications. These trade-offs must be examined by the system designer to determine the best and most cost-effective route to upgrade their system.

Conclusion

The systems and techniques described in this application note demonstrate the feasibility of implementing an external FLASH memory system in a M68HC11 design with minimal hardware/firmware overhead. Since the FLASH support is virtually transparent to the MCU normal mode, it requires a minimum of effort on the part of firmware designers to avoid resource conflicts. In addition, firmware and data updates are straightforward and can be accomplished using a simple terminal program running on the host PC.

Program Listings

Host Interface Reset Controller — Listing 1

```
        PROC          "6805"

;***********************************************************************
; flash mode controller for the FFlash programming system.                 *
;                                                                          *
; Using 4 buttons, this system controls the reset, moda/b,                 *
; vpp, and communications switching.                                       *
;                                                                          *
; RESET(PA3/~IRQ): when a target reset is sensed:                          *
;NORM: no action                                                           *
;PGM: for 300 ms after ~reset is released,close                            *
;       the SHORT connection to feed back the MCU                          *
;       break signal.  Use the ~irq to detect the                          *
;       falling edge of ~reset.                                            *
;                                                                          *
; VPP(PA0):                                                                *
;       PGM: This switch toggles the ~vppen signal (PB1).                  *
;       NORM: switch has no effect.                                        *
;                                                                          *
; MODE(PA1): Toggles the MODA/B signal (PB0) between logic 0               *
;       and hi-z.  Any activity on this input also disables                *
;       Vpp.  MODA/B = 0 is PGM mode...MODA/B = hi-z is NORM               *
;       mode.                                                              *
;                                                                          *
; MONITOR(PA2):                                                            *
;       PGM: this switch has no effect                                     *
;       NORM: This switch cycles through three states:                     *
;       mode  SHORT  txm   OPEN   rxm                                      *
;                     I/O:           0     1      1     0                  *
;                     mon TX:        0     1      0     0                  *
;                     mon RX:        0     0      0     1                  *
;                     PGM reset:     1     1      0     0                  *
;                     PGM mode:      0     1      1     0 (same as I/O)    *
;                                                                          *
;       I/O is for normal communications with the target.                  *
;       mon TX monitors info originating FROM target.                      *
;       mon RX monitors info directed TO target.                           *
;       The state status is saved so that after entering PGM               *
;       mode and returning to NORM mode, the state setting is              *
;       restored to the last selected.                                     *
;                                                                          *
; POR:                                                                     *
```

```
   ;       On power-up, the port initialization is as follows:                *
   ;       PB0 = inputno pull-downs                                            *
   ;       PB1 = 1                                                             *
   ;       pulldown reg                                                        *
   ;       PA0 = input   0                                                     *
   ;       PA1 = input   0                                                     *
   ;       PA2 = input   0                                                     *
   ;       PA3 = input   1                                                     *
   ;       PA4 = 0      n/a            (= 0)                                   *
   ;       PA5 = 1      n/a            (= 0)                                   *
   ;       PA6 = 1      n/a            (= 0)                                   *
   ;       PA7 = 0      n/a            (= 0)                                   *
   ;                                                                           *
   ;********************************************************************jmh

           ;Thu, Dec 5, 1996, 11:14

                  INCLUDE    "a.equ"

           ;      ORG    MOR
           ;      FCB    LEVEL|LVIE|SWAIT ;mask option register settings

                  ORG    ROM

0200   9C  ENTRY RSP                                ; reset stack
0201   A608       LDA    #restarg                   ; set the pull-downs
0203   B710       STA    PDRA
0205   A601       LDA    #modab
0207   B711       STA    PDRB
0209   A602       LDA    #vppen                     ; set DDRs
020B   B701       STA    IDRB
020D   B705       STA    IDDRB
020F   A660       LDA    #OPEN|txm
0211   B700       STA    IDRA
0213   B7E2       STA    MONREG                     ; init monitor register
0215   A6F0       LDA    #rxm|OPEN|txm|SHORT
0217   B704       STA    IDDRA
0219   A603       LDA    #RT1|RT0                   ; set the RTI timer, 65.5ms,
                                                    ; NO RTI
021B   B708       STA    TSCR
021D   A682       LDA    #IRQE|IRQR                 ; enable irq
021F   B70A       STA    ISCR
0221   9A         CLI                               ; enable ints

0222   01050C TOP BRCLR  MODAB,IDDRB,:01            ; skip vpp if normal mode
0225   AD52       BSR    VPPSW
0227   2408       BCC    :01                        ; no switch
0229   B601       LDA    IDRB
022B   A802       EOR    #vppen                     ; toggle VPP
022D   A4FE       ND     #~modab                    ; always leave this one = 0
022F   B701       STA    IDRB
```

```
0231   AD57 :01   BSR    MODSW
0233   2421       BCC    :02                         ; no mode switch
0235   A602       LDA    #vppen                      ; disable VPP
0237   B701       STA    IDRB
0239   B605       LDA    IDDRB                       ; toggle mode
023B   A801       EOR    #modab
023D   B705       STA    IDDRB
023F   000506     BRSET  MODAB,IDDRB,dopgmrst        ;skip mon restore if PGM mode
0242   B6E2       LDA    MONREG                      ; restore monitor setting
0244   B700       STA    IDRA
0246   200E       BRA    :02

0248       dopgmrst
0248   1700       BCLR   RESTARG,IDRA
024A   1604       BSET   RESTARG,IDDRA               ; force retarg low to..
024C   9D         NOP                                ; ..autotrip the target reset
024D   9D         NOP
024E   9D         NOP
024F   9D        NOP
0250   9D         NOP
0251   9D         NOP
0252   A660       LDA    #txm|OPEN                   ; set I/O mode
0254   B700       STA    IDRA
0256   0005C9 :02 BRSET  MODAB,IDDRB,TOP             ; skip mon if PGM mode
0259   AD3E       BSR    MONSW                       ; monitor switch?
025B   24C5       BCC    TOP                         ; no,
025D   B600       LDA    IDRA
025F   A4F0       AND    #txm|OPEN|SHORT|rxm
0261   A160       CMP    #txm|OPEN                   ; no,
0265   A610       LDA    #rxm                        ; set state 2
0267   200A       BRA    doall

0269   A110  not1 CMP    #rxm                        ; is state 2?
026B   2604       BNE    not2                        ; no,
026D   A640       LDA    #txm                        ; set state 3
026F   2002       BRA    doall

0271        not2
            ;     CMP #txm                           ; is state 3?
            ;     BNE not3                           ; no,
0271   A660       LDA    #txm|OPEN                   ; set state 1
0273   B700 doall STA    IDRA
0275   B7E2       STA    MONREG
0277   20A9       BRA    TOP
                  ;
                  ;
                  ; the xxxSW routines test for switch closure (with debounce).
                  ; On return, C = 0 no switch -- C = 1, switch cycled.

0279  01000C      VPPSW  BRCLR vPPON,IDRA,:04
027C   AD2A              BSR   dly20ms
```

```
027E   010007            BRCLR vPPON,IDRA,:04
0281   0000FD      :03   BRSET          vPPON,IDRA,:03
0284   AD22              BSR            dly20ms
0286   99                SEC
0287   81                RTS

0288   98          :04   CLC
0289   81                RTS

028A   0300FB     MODSW  BRCLR          MODE,IDRA,:04
028D   AD19              BSR            dly20ms
028F   0300F6            BRCLR          MODE,IDRA,:04
0292   0200FD     :05    BRSET          MODE,IDRA,:05
0295   AD11              BSR            dly20ms
0297   99                SEC
0298   81                RTS

0299   0500EC     MONSW  BRCLR          MONITOR,IDRA,:04
029C   AD0A              BSR            dly20ms
029E   0500E7            BRCLR          MONITOR,IDRA,:04
02A1   0400FD     :06    BRSET          MONITOR,IDRA,:06
02A4   AD02              BSR            dly20ms
02A6   99                SEC
02A7   81                RTS
                         ;
                         ;
                         ; dly1ms does a simple dex loop to delay 20ms

02A8   A614      dly20ms LDA  #20
02AA   AD04        :07   BSR   dly1ms
02AC   4A                DECA
02AD   26FB              BNE   :07
02AF   81                RTS

02B0   AEA6       dly1ms LDX  #166                ; set count for 1ms (@2mhz)
02B2   9D         :08    NOP                      ; delay 12~ per count
02B3   9D                NOP
02B4   2046              BRA   newdly1            ; patch in a fix for ms1val

                  ;      BRA   :09
02B6   5A         :09    DECX
02B7   26F9              BNE   :08
02B9   81                RTS
                         ;
                         ;
                         ; RTII does the ~~1 sec timer.

02BA   B608       RTII   LDA   TSCR
02BC   AA04              ORA   #RTIFR
02BE   B708              STA   TSCR               ; reset flag
02C0   3AE0              DEC   prescaler
```

```
02C2   260B             BNE   :10                ; not sec yet,
02C4   A60F             LDA   #psval
02C6   B7E0             STA   prescaler
02C8   B6E1             LDA   TIMER1
02CA   2703             BEQ   :10                ; no value to decrement,
02CC   4A               DECA
02CD   B7E1             STA   TIMER1             ; new timer value
02CF   80         :10   RTI
                        ;
                        ;
                        ; operate the reset detect alg.

02D0   010504     IRQI  BRCLR MODAB,IDDRB,:11    ; skip reset if normal mode
02D3   A6C0             LDA   #SHORT|txm
02D5   B700             STA   IDRA
02D7   CD02A8     :11   JSR   dly20ms
02DA   1704             BCLR  RESTARG,IDDRA      ; force reset sw = input
02DC   0700FD     :12   BRCLR RESTARG,IDRA,:12
02DF   01050F           BRCLR MODAB,IDDRB,nopgm  ; skip reset if normal mode
02E2   A61E             LDA   #30                ; delay 600ms
02E4   B7E3             STA   TEMP
02E6   CD02A8     :13   JSR   dly20ms
02E9   3AE3             DEC   TEMP
02EB   26F9             BNE   :13
02ED   A660             LDA   #OPEN|txm
02EF   B700             STA   IDRA
02F1   CD02A8     nopgm JSR   dly20ms
02F4   A682             LDA   #IRQE|IRQR
02F6   B70A             STA   ISCR
02F8   80               RTI
```

FPGA HDL Listing — Listing 2

```
;*******************************************************************************
TITLE 'HC11 bank expander
PATTERN indexer.PDS
REVISION 1.1
AUTHOR Joe Haas
COMPANY
DATE 17 November 1996
;       DATE 12 November 1996 - Creation
;       Tue, Dec 17, 1996, 14:42
;       Removed fLAT.SETF and fLAT.RSTF equations (were invalid)
;       changed lefck to active hi
;       design verified in-circuit by logic analyzer

CHIP  bank_logic11  MACH215

;*******************************************************************************
;PIN   1      gnd
PIN    7..2   D[5..0]       PAIR fLAT[5..0]
PIN    10     A[11]         COMB
PIN    11     A[10]         COMB
;PIN   12     gnd
PIN    13     A[9]          COMB
PIN    15     A[15]         COMB
PIN    14     A[14]         COMB
PIN    9      A[13]         COMB
PIN    8      A[12]         COMB
PIN    21..16 rA[18..13]    LAT
;PIN   22     vcc
;PIN   23     gnd
PIN    24     /rcs          COMB
PIN    25     /acs          COMB
PIN    26     lea           COMB
PIN    27     /rec          COMB
PIN    28     /iocs0        COMB
PIN    29     /iocs1        COMB
PIN    30     /WR           COMB
PIN    31     /RWO          COMB
PIN    32     E             COMB
PIN    33     RW            COMB
;PIN   34     gnd
PIN    35     lefck         COMB
PI     36     lef           COMB
PIN    42..37 fA[18..13]    COMB
PIN    43     /fcs          COMB
;PIN   44     vcc

NODE    ?       fLAT[5..0]      LAT
NODE    ?       DOE             COMB
;*******************************************************************************
EQUATIONS

RWO = RW
WR = /RW * E
```

```
RWO.TRST = VCC
WR.TRST = VCC

D[5..0] = (/A[15] * /A[14] * /A[13] * /A[12] * /A[11] * /A[10] * A[9] * E * RW) *
fLAT[5..0] +
         (/A[15] * /A[14] * /A[13] * /A[12] * /A[11] * A[10] * /A[9] * E * RW) *
rA[18..13]

DOE = (/A[15] * /A[14] * /A[13] * /A[12] * /A[11] * /A[10] * A[9]) +
      (/A[15] * /A[14] * /A[13] * /A[12] * /A[11] * A[10] * /A[9])

D[5..0].TRST = DOE * E * RW

fLAT[5..0] = D[5..0]
fLAT[5..0].CLKF = lefck
lef = /A[15] * /A[14] * /A[13] * /A[12] * /A[11] * /A[10] * A[9] * E * /RW

lef.TRST = VCC

fA[18..15] = A[15] + fLAT[5..2]
fA[14] = (A[14] * A[15]) + (/A[15] * fLAT[1])
fA[13] = (A[13] * A[15]) + (/A[15] * fLAT[0])
fA[18..13].TRST = VCC

fcs = A[15] * E + /A[15] * /A[14] * A[13] * E
fcs.TRST = VCC

rA[18..13] = D[5..0]
rA[18..13].CLKF = /A[15] * /A[14] * /A[13] * /A[12] * /A[11] * A[10] * /A[9] * E * /RW

rA[18..13].SETF = gnd
rA[18..13].RSTF = gnd
rA[18..13].TRST = VCC

rcs = /A[15] * A[14] * /A[13] * E
rcs.TRST = VCC

acs = /A[15] * A[14] * A[13] * E
acs.TRST = VCC

iocs0 = /A[15] * /A[14] * /A[13] * /A[12] * A[11] * A[10] * /A[9] * E

iocs1 = /A[15] * /A[14] * /A[13] * /A[12] * A[11] * A[10] * A[9] * E

iocs0.TRST = VCC
iocs1.TRST = VCC

lea = /A[15] * /A[14] * /A[13] * /A[12] * /A[11] * A[10] * A[9] * E * /RW

rec = /A[15] * /A[14] * /A[13] * /A[12] * A[11] * /A[10] * /A[9]* E * RW

lea.TRST = VCC
rec.TRST = VCC
;******************************************************************************
```

FLASH Algorithm Listing — Listing 3

```
                          PROC         "68HC11"
  = FFFF           ROMON  EQU          ~0      ; development trap
                  ;
                  ;***********************************************************
                  ;*   FF-FLASH programmer firmware for 68HC711E9            *
                  ;*   Written by: Joseph M. Haas  Originated Feb-05-93      *
                  ;*   Revision MEZ-11  Retrofit   Jun-30-95                 *
                  ;*********************************************************
                  ;
                  ;**** Revision History ****
                  ;added BANK command to allow access to multi-megabit device
                  ;in 64K banks.
                  ;u3hilim now stores a18..12.  all references to A16 now deal
                  ;with a18..16
                  ;eflas now handles multi-megabit devices...technically
                  ;abandoned u3hi and u5hi
                  ;added COPON/OFF commands
                  ;
                  ;*********************************************************
                  ;*   END OF REVISION HISTORY                             *
                  ;*   Last release date:   Fri, Jun 30, 1995, 16:05       *
                  ;*********************************************************
                        ;
                        ;
                        ; REDbyt gets byte in (B) from (X) addr of target device

D742             REDbyt
D742   3C               PSHX            bytSET       ; set addr
D743   8D07             BSR             bytSET       ; set addr
D745   BDD81D           JSR             adrFLA
D748   E600             LDAB            0,X          ; get EPROM
D74A   38               PULX
D74B   39               RTS
                        ;
                        ;
                        ; bytSET sets the bank addr bits for the current device

D74C              bytSET                           ; check devtyp
D74C   D639             LDAB      DEVwin
D74E   C160             CMPB      #U4win
D750   260C             BNE       bytSU

D752   9C17       bytSU4CPX       HiLim+1          ; use hilim for address limit
                                                   ; test
D754   2302             BLS       :64              ; yes,
D756   0D         byter SEC                        ; set address err
D757   39               RTS

```

```
D758   9C14       :64   CPX       LoLim+1          ; is out of flash?
D75A   25FA             BLO       byter            ; yes,
D75C   0C               CLC
D75D   39               RTS

D75E              bytSU                            ; set SPBNK bits A unused on
                                                   ; entry
D75E   3C               PSHX
D75F   C120             CMPB      #U3win           ; is U3?
D761   2716             BEQ       :65              ; yes,
D763   8F               XGDX                       ; --- DO MEZ-11 U5 BANKS ---
D764   84E0             ANDA      #%11100000       ; mask hi addrs
D766   44               LSRA
D767   44               LSRA
D768   44               LSRA
D769   44               LSRA
D76A   44               LSRA
D76B   D62F             LDAB      FBANK
D76D   C4F8             ANDB      #%11111000
D76F   1B               ABA
D770   972F             STAA      FBANK
D772   B70400           STAA      rbank
D775   8F               XGDX
D776   38               PULX
D777   0C               CLC
D778   39               RTS

D779   8F         :65   XGDX                       ; --- DO MEZ-11 U3 BANKS ---
D77A   49               ROLA
D77B   49               ROLA
D77C   49               ROLA
D77D   49               ROLA
D77E   8407             ANDA      #%00000111       ; mask hi addrs
D780   D62F             LDAB      FBANK
D782   C438             ANDB      #%00111000
D784   1B               ABA
D785   972F             STAA      FBANK
D787   B70200           STAA      fbank
D78A   8F               XGDX
D78B   38               PULX
D78C   0C               CLC
D78D   39               RTS
                        ;
                        ;
                        ;PGMbyt programs, verifies byte in (A) at device addr (X)

D78E   3C         Verif1          PSHX
D78F   BDD81D           JSR       adrFLA
D792   C6C0             LDAB      #flaPV           ; flash pgm verify
D794   E700             STAB      0,X              ; send data to target
D796   183C             PSHY
```

```
D798   1838              PULY
D79A   01                NOP
D79B   01                NOP
D79C   01                NOP
D79D   01                NOP                          ; ~=5+6+(2*4)=17 = 8.5uS
D79E   E600              LDAB      0,X                ; get data
D7A0   11                CBA                          ; compare device to orig
D7A1   38                PULX
D7A2   39                RTS


D7A3   0F         PGMbyt          SEI
D7A4   12288004          BRSET     FLAG2,FFSKIQ,:66 ; no $FF skip allowed
D7A8   81FF              CMPA      #$FF               ; do $FF skip?
D7AA   272A              BEQ       FFskip             ; yes,
D7AC   37         :66    PSHB
D7AD m                   BOOTOF                       ; turn off bootrom
D7AD   8D9D              BSR       bytSET             ; test addr
D7AF   2408              BCC       :67
D7B1   142720            BSET      PGMER,DAE          ; address error
D7B4   BDD9F3            JSR       FLASHon            ; flash error led
D7B7   201C              BRA       PGMqq

D7B9             :67
                  ;      STAA      0,X                ; write byte
D7B9   C619              LDAB      #25                ; set cycle count
D7BB   D721              STAB      TMP2
D7BD   D639              LDAB      DEVwin             ; is U3?
D7BF   C120              CMPB      #U3win
D7C1   2704              BEQ       :68                ; yes,
D7C3   C601              LDAB      #1
D7C5   D721              STAB      TMP2               ; set all others for 1 cycle
D7C7   8D10       :68    BSR       Prog1              ; prog pulse 1ms
D7C9   11                CBA                          ; verify byte
D7CA   2709              BEQ       :69                ; is ok,
D7CC   7A0021            DEC       TMP2               ; next cycle
D7CF   26F6              BNE       :68                ; not done,
D7D1 m            PGMerr          BOOTON             ; turn on bootrom
D7D1   33                PULB
D7D2   0D                SEC                          ; set byte err
D7D3   0E                CLI
D7D4   39                RTS

D7D5             :69
D7D5             PGMqq
D7D5 m                   BOOTON                       ; turn on bootrom
D7D5   33                PULB
D7D6   0C         FFskipCLC                           ; ok exit
D7D7   0E                CLI
D7D8   39                RTS
                         ;
```

```
                        ;
                        ; Prog1 writes (a) to the current device

D7D9   140020     Prog1 BSET      FLAG,ms1         ; set for 1ms
D7DC   D639       ProgX LDAB      DEVwin           ; is U3?
D7DE   C120             CMPB      #U3win
D7E0   2603             BNE       :70              ; no,
D7E2   8D11             BSR       wrFLA
D7E4   39               RTS

D7E5   8D01       :70   BSR       wrRAM            ; do U4/5
D7E7   39               RTS
                        ;
                        ;
                        ; wr?? is a collection of routines that handle the
                        ; programming operations of all supported devices.
                        ; devices that subscribe to the multi pulse programming
                        ; will return w/ C = 0.  single write devices (ie., rams)
                        ; return w/ (B) = read of pgm'd byte (for verify).
                        ; A=data, X=addr

D7E8   3C         wrRAM PSHX                       ; save addr
D7E9   8D32             BSR       adrFLA           ; set write address
D7EB   A700             STAA      0,X              ; send data to target
D7ED   D600             LDAB      $0               ; clear bus
D7EF   D647             LDAB      $47
D7F1   E600             LDAB      0,X              ; read data for verify
D7F3   38               PULX                       ; restore addr
D7F4   39               RTS


D7F5   3C         wrFLA PSHX                       ; save addr
D7F6   8D25             BSR       adrFLA           ; set write address
D7F8   C640             LDAB      #flaPG           ; flash pgm cmd
D7FA   E700             STAB      0,X              ; send data to target
D7FC   16               TAB
D7FD   E700             STAB      0,X              ; send data to target
D7FF   C609             LDAB      #9               ; delay 12us
D801   5A         :71   DECB
D802   26FD             BNE       :71
D804   C6C0             LDAB      #flaPV           ; flash pgm verify
D806   E700             STAB      0,X              ; send data to target
D808   183C             PSHY
D80A   1838             PULY
D80C   01               NOP
D80D   01               NOP
D80E   01               NOP
D80F   01               NOP                        ; ~=5+6+(2*4)=17 = 8.5uS
D810   E600             LDAB      0,X              ; get data
D812   38               PULX                       ; restore addr
D813   39               RTS
```

```

D814   3C          FLwrRM PSHX
D815   8D06              BSR       adrFLA           ; set phys address
D817   C600              LDAB      #flaRM           ; set for read mem mode
D819   E700              STAB      0,X              ; send data to target
D81B   38                PULX
D81C   39                RTS
                         ;
                         ;
                         ; adrFLA moves addr in (X) to window @ selected device:
                         ;     U3: $2000-3FFF       %00100000 = U3win
                         ;     U4: $6000-7FFF       %01100000 = U4win
                         ;     U5: $4000-5FFF       %01000000 = U5win
                         ;  files that program these devices must start @ $0000
                         ;  and proceed
                         ;  to $FFFF.

D81D   36         adrFLA PSHA
D81E   8F                XGDX
D81F   841F              ANDA      #%00011111       ; mask hi addrs
D821   9A39              ORAA      DEVwin           ; combine window mask
D823   8F                XGDX                       ; back to X
D824   32                PULA
D825   39                RTS
```

References

Motorola's HC05/08 Website:

http://www.mcu.motsps.com/index.html

Application Note

MOTOROLA