68HC11 Microcontroller
Design Project

Amir Malik

CMPE 121L, Spring 2006
UCSC Baskin School of Engineering

1 Introduction

The objective of this laboratory project was to design, construct, and debug an expanded-bus microcontroller system using the Motorola 68HC11 microcontroller. We were required to implement at least 8 Kbytes of ROM, 8 Kbytes of RAM, and an RS-232C serial interface. The system was to be operable over 7 to 12 V DC, with the use of a voltage regulator. For the software portion of the project, we were required to implement code to test for "stuck" address-lines in the RAM, and code to test for bad memory locations in the RAM chip.

2 Design

In my particular design, I decided to implement 30 Kbytes of SRAM, 32 Kbytes of FLASH ROM, and eight dedicated I/O ports. In my memory map, I allocated space for 2048 I/O ports, but only eight are currently available due to space limitations on the board, among other factors. I used Orcad Capture CIS to perform schematic capture design for this project. It made the bird's eye view of the layout much easier to understand.

3 Construction

After the engineering schematic and the wiring diagrams were realized, I began construction on the physical board. Because my wiring diagram was almost exactly to the board's scale in terms of pin count and spacing, I merely started placing all of the major IC sockets into their respective positions and soldered them to the board. I then proceeded by adding required bypass capacitors, resistors, the crystal, and other components. The 4004 diode and the 7805 3-terminal 5 V power regulator, along with the heatsink and respective power and ground rails, were placed near the HC11 chip at the top of the board. Furthermore external interface devices such as the RESET and IRQ pushbuttons, as well as the RS232 header, were placed at the top edge of the board to facilitate physical manipulation. I used a green screw-in type terminal connector to connect the power supply lines because I found that the 9V connector was not reliable; I had to resolder it numerous times before I decided to use this.

4 Software

I think the most difficult part of the project was the hardware design and construction, as I had neither seriously soldered before (except for a Radio Shack wireless FM transmitter kit) nor had I ever wirewrapped. Software has always been my better side, and I thoroughly enjoyed the embedded programming task. I would not have been hesitant to implement a full-blown embedded real-time operating system, utilizing all of the features of the HC11 chip, but time did not permit this! I used the excellent GNU HC11 compiler toolkit package written and maintained by Stephane Carrez. My choice of a free compiler as opposed to the Cosmic HC11 compiler, allowed me to work on the software at home. Furthermore, the GNU toolkit had a simulator, which allowed me to significantly test portions of my code without going to the PROM burner every time I wanted to try out something new. There were, however, some deficiencies in the simulator, as I found out in some mails sent to the author. Apparently, I could not change my 64 byte register block inside the simulator, as it was hard coded to 0x1000. Thus, when I ran my code in the simulator, I made sure to define by register base at 0x1000, and whenever I went to burn it to the ROM chip, I had to change it back to my remapped location o 0xF000. Although this procedure itself proved cumbersome, in the end, I think it saved me some frustration.

4.1 Boot code

The boot code listing appears in the Appendix. Essentially the boot code sets up the serial interface with a default baud rate of 9600. The default baud rate is selectable at runtime in the software by running the “baud 9600” or “baud 1200” commands in the operating system shell prompt. In addition to SCI setup, I wrote code to test all addressable SRAM memory locations for bad locations. That is, if a particular byte of memory returns a non-deterministic value after being written to, I notify the user with a message on the serial port. This is accomplished by first writing 0x00 to the location and then reading it back and making sure it is still 0x00. We then write 0xFF to the location and read it back to make sure it is still 0xFF. This assures us that none of the bits are "stuck" to 0 or 1. This covers all our bases. Finally, we set up the stack pointer at the bottom of our RAM, and let it grow up to zero, as is customary.

4.2 Application Code

The application code is written in C, and consists of an interrupt-driven SCI driver which offers non-blocking getc() and putc() functionality. The main event loop is a shell prompt designed to emulate a UNIX(tm) system. I implemented an output-compare timer using interrupts, which would count a tick every 8 microseconds, and my software would subsequently update the HH:MM:SS time at the lower-left side of the terminal screen. The use of VT100 escape commands greatly helped in achieving this requirement. A rudimentary command parser also exists in the operating system. It supports the following commands: help, ?, cls, clear, baud 1200, baud 9600, halt, reboot. A Ctrl-C command is also implemented to allow the user to "cancel" a command on the shell line, although it does not interrupt a running "process."

5 Conclusion

Overall, I think this project was a valuable multi-disciplinary learning experience because it combined aspects of electrical engineering, physics, and computer science into a real-world embedded application. I would have loved to implement a full-blown operating system with POSIX-compliant threading, but I think that may have been slightly overkill for a project of this size. As you can probably tell, I'm more of a software guy than a hardware guy, but I appreciate all the hardware-software interfacing that is required to build a successful system. Had I had the chance to do it over again, I would have completed my design much quicker, and actually implemented a hardware expansion. Initially I wanted to use an Ethernet controller on my board, but as none of the companies I had sent sample requests to replied, I decided not to do that. Then I wanted to create a simple VU-meter, so that I would connect an electret microphone (which I also ordered) through a low-pass filter and an op-amp to one of the analog input pins of the HC11's A/D converter. The software would have merely read in the converted values from the ADC and created a short graphical representation of the intensity of the signal from the microphone. Overall, however, I think I understood the basics required to make this work, but my time should have been budgeted a little better.

6 Appendix

```
; boot.asm
; M68HC11 expanded-mode boot code
;
; Author: Amir Malik <amalik@ucsc.edu>
; Written for CMPE 121/L Spring 2006
; Portions of this file are based on v2_18g3.asm distributed with CE12 Microkit

#include "buildversion.s"

#ifndef VERSION
#define VERSION "1.0"
#endif

#ifndef BUILDDATE
#define BUILDDATE "unknown"
#endif

#define REGBASE     0xF000          // originally: 0x1000
#define TCNT        REGBASE+0x0E    // master Timer Counter
#define TCTL1       REGBASE+0x20    // Timer Control Register 1
#define TMSK1       REGBASE+0x22    // Timer Interrupt Mask 1 Register
#define TFLG1       REGBASE+0x23    // Timer Interrupt Flag 1 Register
#define TMSK2       REGBASE+0x24    // Timer Interrupt Mask 2 Register
#define TFLG2       REGBASE+0x25    // Timer Interrupt Flag 2 Register
#define PACTL       REGBASE+0x26    // Pulse Accumulator Control
#define BAUD        REGBASE+0x2b    // sci baud register
#define SCCR1       REGBASE+0x2c    // sci control1 register
#define SCCR2       REGBASE+0x2d    // sci control2 register
#define SCSR        REGBASE+0x2e    // sci status register
#define SCDR        REGBASE+0x2f    // sci data register
#define OPTION      REGBASE+0x39
#define CONFIG      REGBASE+0x3f    // originally: 0x103F
#define INIT        0x103D
```

```
  .sect page0
  .global _.frame
  .global _.d1
  .global _.d2
  .global _.d3
  .global _.d4
  .global _.z
  .global _.xy
  .global _.tmp
_.tmp:
  .word 0
_.z:
  .word 0
_.xy:
  .word 0
_.frame:
  .word 0
_.d1:
  .word 0
_.d2:
  .word 0
_.d3:
  .word 0
_.d4:
  .word 0

  .sect .vectors
  .global vectors
vectors:
  .word def                   ; ffc0
  .word def                   ; ffc2
  .word def                   ; ffc4
  .word def                   ; ffc6
  .word def                   ; ffc8
  .word def                   ; ffca
  .word def                   ; ffcc
  .word def                   ; ffce
  .word def                   ; ffd0
  .word def                   ; ffd2
  .word def                   ; ffd4

  ;; SCI
  .word interrupt_sci         ; ffd6

  ;; SPI
```

```
  .word def                     ; ffd8
  .word def                     ; ffda (PAII)
  .word def                     ; ffdc (PAOVI)
  .word def                     ; ffde (TOI)

  ;; Timer Output Compare
  .word interrupt_oc5           ; ffe0 (OC5)
  .word def                     ; ffe2 (OC4)
  .word def                     ; ffe4 (OC3)
  .word def                     ; ffe6 (OC2)
  .word def                     ; ffe8 (OC1)

  ;; Timer Input compare
  .word def                     ; ffea (IC3)
  .word def                     ; ffec (IC2)
  .word def                     ; ffee (IC1)

  ;; Misc
  .word def                     ; fff0 (RTII)
  .word interrupt_irq           ; fff2 (IRQ)
  .word interrupt_xirq          ; fff4 (XIRQ)
  .word def                     ; fff6 (SWI)
  .word def                     ; fff8 (ILL)
  .word def                     ; fffa (COP Failure)
  .word def                     ; fffc (COP Clock monitor)
  .word _start                  ; fffe (reset)


;********************************************************************************
; Write the NULL-terminated string whose starting address is in Y
; Clobbers: A, B, Y

.macro OUTSTRING_stackless:
_OUTSTRINGstart\@:
  ldaa    0, y                  ; get character into A
  cmpa    #0x00
  beq     _OUTSTRINGdone\@      ; we're done if it's NULL

  ; output the char in A
_OUTCHARstart\@:
  ldab    SCSR                  ; read SCSR
  bitb    #0x80                 ; get TDRE bit
  beq     _OUTCHARstart\@       ; loop until TDRE = 1
  staa    SCDR                  ; write character
```

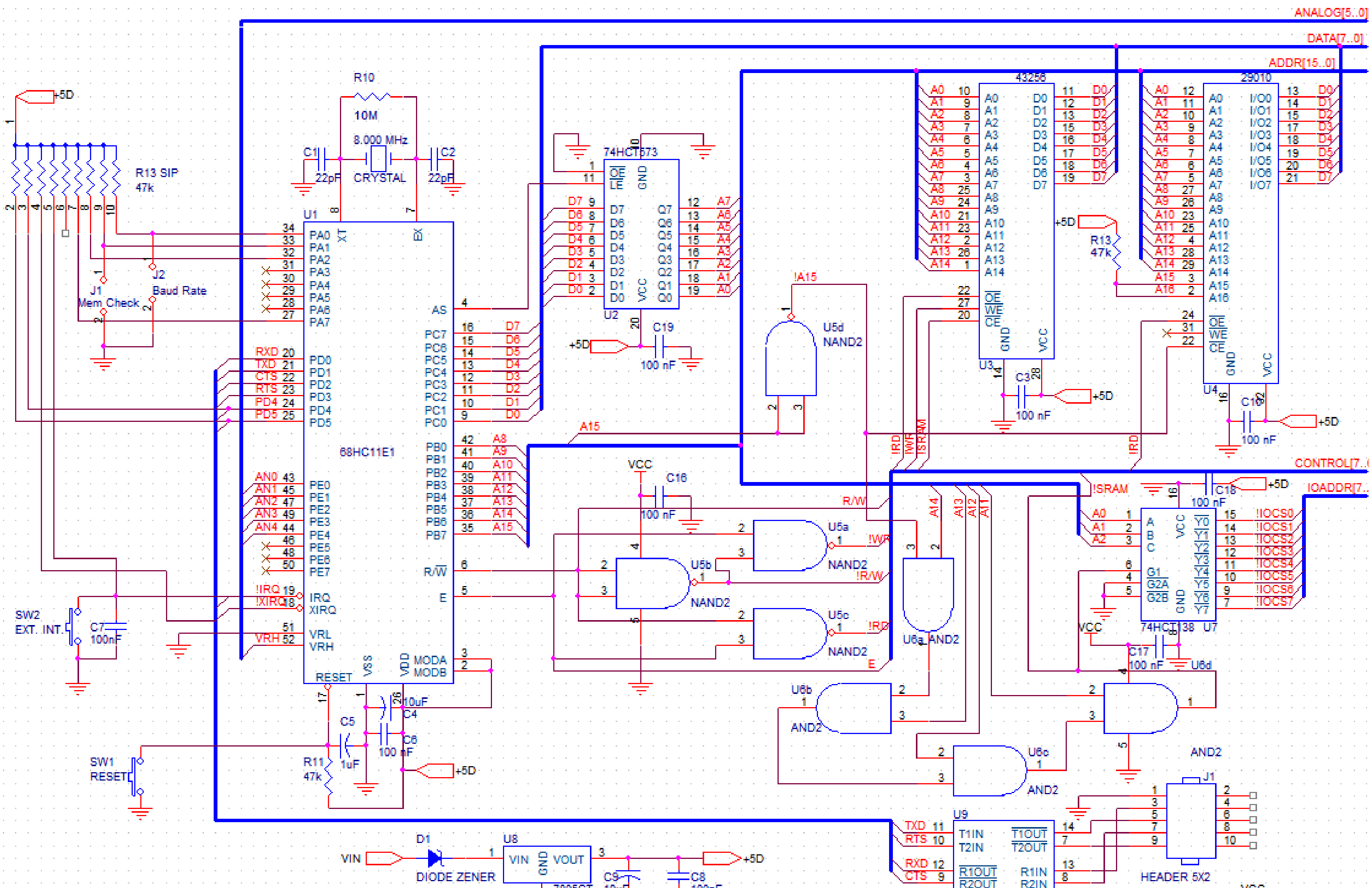
```
  iny
  bra     _OUTSTRINGstart\@
_OUTSTRINGdone\@:
  nop                         ; kludge required for assembler
.endm


;*******************************************************************************
; Write the upper nibble of A to the serial port as a hex character
; Clobbers: A, B
.macro OUTHEX_stackless:
  ;;tba
  anda    #0xF0               ; mask low nibble
  lsra
  lsra
  lsra
  lsra
  cmpa    #0x0A               ; compare with 10 (decimal vs hex)
  blt     _OUTHEXdigit\@      ; A = [0:9]
  suba    #0x0A               ; convert A-F to 0-5
  adda    #'A'                ; add hex bias (to bump it back to A-F in ASCII)
  suba    #'0'                ; bias already added above; hack for next line.
_OUTHEXdigit\@:
  adda    #'0'                ; add ASCII bias

_OUTCHARStart\@:
  ldab    SCSR                ; read SCSR
  bitb    #0x80               ; get TDRE bit
  beq     _OUTCHARStart\@     ; loop until TDRE = 1
  staa    SCDR                ; write character
.endm



;*******************************************************************************
; HC11 boot code and other code to be run within the first 64 clock cycles
  .sect .text
  .global _start
def:                          ; default interrupt
  rti

_start:                       ; label to the base of the text segment

  ; move the 64B register block from 0x1000 to 0xF000
  ldaa    #0x0F               ; REG3 = REG2 = REG1 = REG0 = 1
```

```
  staa    INIT

  ; disable on-chip ROM & EEPROM
  ldaa    #0x0C               ; NOSEC=1, NOCOP=1, ROMON=0, EEON=0
  staa    CONFIG

  ; change prescaler to 8us resolution (must be done within 64 clock cycles)
  ldaa    #0x03               ; PR1 = PR0 = 1
  staa    TMSK2

  ; set up output capture on OC5
  ldaa    #0x00               ; we do not want output pins to be changed
  staa    TCTL1
  ldaa    #0x08               ; select OC5
  staa    TMSK1               ; mask enable

  ; set up the SCI interface
  ldaa    #0x30               ; 9600 baud
  staa    BAUD
  ldaa    #0x00
  staa    SCCR1
  ldaa    #0x0C               ; no interrupts yet
  staa    SCCR2

  ; print the boot screen
  ldy     #bootmsg0
  OUTSTRING_stackless         ; clrscr and print BIOS banner

  ; print message
  ldy     #bootmsg3
  OUTSTRING_stackless

;*******************************************************************************
; Memory Tester (clobbers registers)
MemoryTest:
_MemTest0:

  ; check 0000-77FF (RAM range)
  ldx     #0x0000             ; starting address

_MemoryTestGo:
  ldaa    #0xFF               ; test pattern
  staa    0, x                ; store pattern in memory
  cmpa    0, x                ; compare A and Mem(X)
  bne     _MemoryTestBad      ; verification failed!
```

```
  ldaa    #0x00               ; test pattern
  staa    0, x                ; store pattern in memory
  cmpa    0, x                ; compare A and Mem(X)
  bne     _MemoryTestBad      ; verification failed!

_MemoryTestNext:
  inx                         ; increment address
  cpx     #0x77FF             ; are we at the end?
  beq     _MemoryTestDone
  bra     _MemoryTestGo

_MemoryTestBad:
  ldy     #bootmsg4b
  OUTSTRING_stackless

  ; print out address in X
  txs                         ; save X to SP
  xgdx                        ; swap D and X

  ; print first nibble
  OUTHEX_stackless            ; clobber D
  xgdx                        ; restore D
  tsx                         ; restore X

  ; print second nibble
  xgdx
  lsla
  lsla
  lsla
  lsla
  OUTHEX_stackless            ; clobber D
  xgdx                        ; restore D
  tsx                         ; restore X

  ; print third nibble
  xgdx
  tba
  OUTHEX_stackless            ; clobber D
  xgdx                        ; restore D
  tsx                         ; restore X

  ; print fourth nibble
  xgdx
  tba
```

```
  lsla
  lsla
  lsla
  lsla
  OUTHEX_stackless            ; clobber D
  xgdx                        ; restore D
  tsx                         ; restore X

_MemTest6:
  bra     _MemoryTestNext     ; continue to next address

_MemoryTestDone:
  ldy     #bootmsg5
  OUTSTRING_stackless



FinalizeSCI:
  ldaa    #0xAC               ; full RX/TX interrupt-driven mode
  staa    SCCR2

  ; newline
  ldy     #bootmsg6
  OUTSTRING_stackless

  ; clean up
  clra
  clrb
  ldx     #0
  ldy     #0

  ; set up the stack
  lds     #0x7BFF             ; sp <- 8000 - 1K - 1

  ; enable interrupts
  cli

InvokeMain:
  jsr     main                ; jump to event loop in main()

  ; something is wrong
  sei
  ldy     #bootmsg9
  OUTSTRING_stackless

Halt:
```

```
  bra     Halt



bootmsg0:     .ascii  "\x1B[2J\x1B[0;0H"

bootmsg0a:    .ascii  "\x1B[?7"    ; auto-wrap

bootmsg0b:    .ascii  "\x1B[?3"    ; 80 columns wide

bootmsg0c:    .ascii  "\x1B[0;22r" ; 0-22 usable lines, 23 is clock line

bootmsg1:     .ascii  "      _\r\n     / \\\r\n    / _ \\    AmirBIOS\r\n   / ___ \\   (c) 2006
Version " VERSION "\r\n  /_/   \\_\\\r\n"

bootmsg2:     .ascii  "\r\n  BIOS Build Date " BUILDDATE "\r\n\r\n  On-chip ROM & EEPROM disabled.\r\n
SCI initialized.\r\n  Detected 30 KB RAM.\000"

bootmsg3:     .asciz  " Verifying... "

bootmsg4a:    .asciz  "\r\n    Please check address line wiring!"

bootmsg4b:    .asciz  "\r\n    Bad memory location: 0x"

bootmsg5:     .ascii  "\r\n  ...DONE!"

bootmsg6:     .asciz  "\r\n"

bootmsg9:     .asciz  "\r\nBIOS: System halted.\r\n"


  end
```



Title		
CMPE 121L: 68HC11 Design Project (Amir Malik)		
Size: B	Document Number: Schematic 1	Rev: 3
Date: Sunday, May 07, 2006	Sheet 1 of 1	