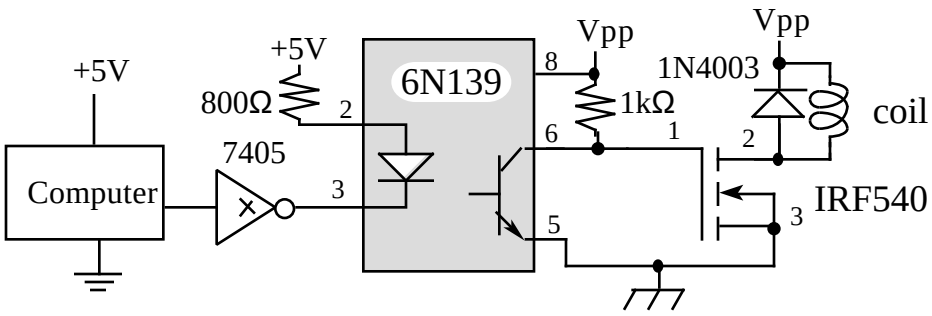
Embedded Microcomputer Systems: Real Time Interfacing

Laboratory Manual



Fall 2001

Jonathan W. Valvano
Electrical and Computer Engineering
University of Texas at Austin

email: valvano@uts.cc.utexas.edu
web: http://www.ece.utexas.edu/~valvano
how to program: http://www.ece.utexas.edu/~valvano/embed/toc1.htm

I. Introduction to Microcomputer Laboratory

I.1. Grading Policies

Groups will consist of exactly two students. Lab partners have separate lab quiz and checkout grades, but share the preparation report and late penalty grades. For each lab assignment, there are a number of preparation tasks that must be completed before the laboratory period. The purpose of the lab quiz is to evaluate the completeness of the lab preparation. At the beginning of lab, you turn in your preparation and take the lab quiz. If a student has completed the preparation, he/she should perform well on the lab quiz. During the quiz, the TA should check your preparation and return it after the quiz.

Responsibility		grade
Quiz		20^{+}
Preparation		10^{*}
	Printout of software, due at the start of lab	
	Edited and compiled with no syntax errors	
	Not handwritten, hard copy printout, but not necessarily debugged	
	Hardware circuit diagram, due at the start of lab	
	Chip numbers, pin numbers, resistor/capacitor types and tolerances	
	Connections to computer,	
	Procurement of all necessary parts	
	Preliminary hardware circuit diagrams but not necessarily built or debugged	
Report		45^{*}
	(5) Final hardware circuit diagrams (handwritten drawings are OK)	
	(25) Software quality	
	Documentation, the comments, choice of good variable names	
	Interface to the human operator, menus, error messages	
	Style, organization, top down structure, local variables, careful use of the stack	
	(15) Results printout, performance/data graphs (handwritten drawings are OK)	
Checkout, demonstration to TA		
	Performance, correctness of the program function	15^{*}
	Oral understanding of engineering tradeoffs	10^{+}
Penalty	Late Checkout	$-5/day^{*}$
	Late Report	$-5/day^{*}$
Total		100^{+}

$^{+}$ partners receive a separate grade
* partners receive the same grade

I.2. SAFETY REGULATIONS:

IN CASE OF EMERGENCY DIAL 911 or 9-911

Since there will be times when students will work other than the regularly scheduled lab sections, it is necessary that certain regulations be observed for the convenience and safety of all. Since the possibility of lethal shock exists in those circuits utilizing low potentials, the following should always be observed:

1. Working alone in a lab room is not permitted.

2. Working after regular hours without written permission is not permitted.

3. Work benches must be clear of all coats, knapsacks and extraneous materials. Coat racks are desired for those desiring this convenience. Otherwise all materials must be stored under the work area or out of the way.

4. Shoes must be worn in the lab at all times. Shoes represent a significant protection against electrical shock.

5. Smoking, food and beverages (e.g., coffee) are not permitted anywhere in the lab area.

IN CASE OF INJURY OR SHOCK:

Turn off power, do not move the injured. Start artificial respiration if breathing has stopped. Have someone else call 911 or 9-911 if CPR is needed.

IN CASE OF FIRE:

Turn off the power, call 911 or 9-911, fight fire with available extinguisher, have someone clear the building.

I.3. LAB PROCEDURES AND POLICIES

FIRST_WEEK_OF_CLASSES: Go to the regularly scheduled lab during the first week of classes. During this time, you will be introduced to the lab equipment. You will also be instructed on lab procedures and grading policy. If you missed your regularly scheduled lab, attend one of the other lab periods. Note that attending a lab session for which you are not registered is not permitted except during the first week of classes.

LAB_PARTNERS: Every student is required to have a lab partner. You will perform all labs with a partner. Students choose their own lab partners during the first week.

LAB_EQUIPMENT_USAGE: Lab hours are posted in the laboratory. There are no sign-up sheets, but cooperation is expected. If you start debugging on a station, you may stay as long as you like, with three exceptions:

You must leave when the second floor labs are closed for the day;
You must leave during the first hour of the other regularly scheduled lab periods;
You may not leave the station unattended for more than 15 minutes.

If you would like to use a station that has been left unattended for more than 15 minutes:

1) Carefully disconnect the hardware and eject any floppy disks;
2) Do not save any software files;
3) Return all materials (hardware, disks, paper) to the front desk;
4) Leave a note on the station with your name and time;
5) Write a note to the TA describing exact times listing what you turned in.

LAB_QUIZ: The purpose of the lab quiz is to encourage students to prepare for the lab. The scope of the quizzes will be material relevant to the lab preparation. The quiz will be conducted during the first 10-15 minutes of each lab session.

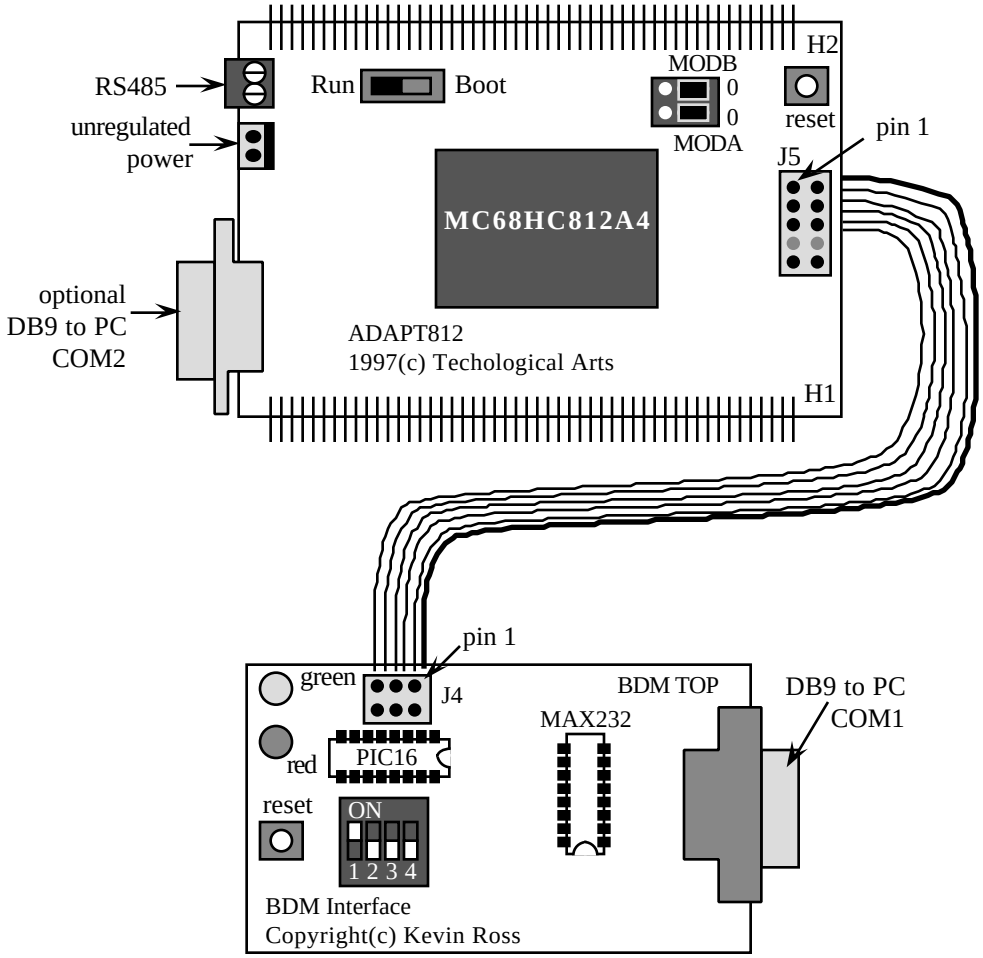
LAB_PREPARATION: Lab preparation must be performed prior to the regularly scheduled lab period. All software must be written, edited and designed before coming to the lab. Hardware must be designed down to the pin numbers. Label all resistance and capacitance values and types. For example, 1kΩ 5% carbon, or 0.01μF 5% ceramic. In this way, the lab period may be spent in debugging your system with the TA's help. The preparation is due at the start of the lab. Preparation includes gathering all the physical components required to perform the lab.

LAB_REPORT: Each lab report is bound separately (you should not put them together.) *No purpose, methods, or analysis required.* Every lab report should include the following items:

1. Detailed hardware designs with pin numbers.
   Include all external devices used (chips, R's C's values and types)
   Show connections to the 6812 board.
2. Source code listings: Document your software so that what you did and why you did it could be easily understood by the TA (and by you a year from now.)
   Make it clear how the individual files are compiled together.
3. Data/Results: Whenever appropriate, use the printer to get a hardcopy listing of your results. Include graphs and figures as specified in the assignment.

CHECKOUT: A rough draft of your hardware diagrams and hard copy printout of your source code listings must be given to the TA before you demonstrate. If your experiment works you will be assigned a good score on the performance part. The TA will ask the partners oral questions that test your "understanding" of the computer engineering concepts of the lab. The partners will answer separate questions and receive separate "understanding" grades. You must get your rough draft software listings signed and dated by a TA to prove that the lab was completed in a satisfactory manner. Late checkouts will result in lost points.

I.4. PROJECT DEVELOPMENT



Each of the stations is equipped with the following:
- The PC COM1 is connected to a serial port 2-to-1 switch box
  one serial port is connected to a Kevin Ross BDM-12 module
  the other serial port is connected to a Motorola MC68HC912B32 EVB (pod mode)
- The PC COM2 is connected to your Adapt812 SCI0 port (optional)
- A Kevin Ross BDM-12 background debug module
  you will connect this BDM-12 to the BDM port of your Adapt812
- An unregulated power supply
  you will connect this power to the power connector of your Adapt812

I.4.1. BDM-12/Adapt812 interface

The Kevin Ross BDM-12 is a debugging interface between the PC and the Adapt812. It allows us to download programs, view/change registers, view/change memory, single step, and run software. When the logic analyzer is used to debug, we will use the BDM-12 to modify the MC68HC812A4 mode so that the address and data bus are available on the Adapt812 H2 connector. Circuit diagrams and software manuals are included at the end

of this introduction chapter. With Windows NT or 95, you will run Kevin Ross's DOS-level **DB12.EXE** program using the batch file we wrote called **B.BAT**. You can purchase your own BDM-12 board in kit form for about $48 from Kevin Ross at "Kevin Ross" <kevinro@nwlink.com> Identity yourself as a student.
The proper DIP settings on the Kevin Ross BDM-12 board are

1 ON for 38400 bits/sec
2 OFF for debugger mode
3 OFF for 8 MHz 6812
4 OFF for 8 MHz 6812

The 6-pin ribbon cable between the Kevin Ross BDM-12 and the Technological Arts Adapt812 should be attached so that Pin 1 is connected to Pin 1. It is possible to reverse the polarity so be careful. The power lights come on (but the system doesn't work) if the cable is reversed. There are two versions of the Adapt812 board[1]. The Technological Arts APAPT812 Rev1 settings are

Run/Boot switch (SW2) should be in the *Run* position (bypass the on-chip boot loader)
MODA/MODB jumpers should both be in the *0* position (single chip mode.)

The Technological Arts APAPT812 Rev2 settings are

Run/Boot switch (SW2) should be in the *Run* position (bypass the on-chip boot loader)
Exp/Sgl switch (SW3) should be in the *Sgl* position (for single chip mode)
Rterm jumper (W12) should be inserted (enables resistor for RS485 network)
MODA jumper (JB2) should be inserted
MODB jumper (JB1) should both be in the *0* position (single chip mode.)

I.4.2. IBM-PC and ICC11/ICC12

I.4.2.1. ImageCraft

The PC can also be used as an intelligent terminal to interface with the 6812 micro-controller. It will also be used to write the software, assemble it, and then down load it to the 6812. There are professional quality C compilers from ImageCraft at a low cost. IMAGECRAFT ICC12 IS NOT FREEWARE. You would need to add an editor and a terminal program. The ImageCraft Windows-based integrated development system includes an editor and terminal program for about $165+shipping. The full system with IDE supports direct programming of 'E2's EEPROM in single chip mode at 8 MHz crystal (e.g. the MIT Miniboard, BotBoard, TechArt board etc.) ImageCraft can be contacted at

ImageCraft
706 Colorado Ave. Suite 10-88
Palo Alto, CA 94303
info@imagecraft.com
http://www.imagecraft.com

cool web site: **http://www.interlog.com/~techart/myfiles/links.ltml**
Developing Embedded Software in C **http://www.ece.utexas.edu/~valvano/embed/toc1.htm**

File type	Example	How you use it
C source code	LED1.C	Modify using the editor, save to floppy
C header code	HC12.H	Modify using the editor, save to floppy
preprocessed C source	LED1.I	you don't use it
assembly source	LED1.S	normally you don't use it[2]
relocatable object	LED1.O	you don't use it
library archive	LED1.A	you don't use it for small programs
object code	LED1.S19	you download this into the µC
assembly listing	LED1.LIS	you don't use it
linker listing	LED1.LST	very helpful for debugging
linker map	LED1.MP or LED1.MAP	very helpful for debugging

[1] Rev1 and Rev2 have different H1 connector pin assignments for PS5 PS6, PS7
[2] You could write assembly programs by creating file.s assembly source files

I.4.2.2. Interrupts, vectors, and Dealing with the COP on the Adapt812

In ICC12, the steps required to initialize an interrupt handler are illustrated in the following example. When the Adapt812 software is started with the reset button, it is in *normal single chip mode* with Computer Operating Properly (COP) enabled. You must either disable COP as is done in this example or periodically refresh it as is done in the second example. Notice how the reset vector is defined at the end of the program. After reset on a software system created with ICC12, the 6812 will begin at `_start`, which initializes RAM and calls your `main()` program. In ICC12, the entry point `_start` is defined in the file crt12.s.

In any mode, your software can disable the COP feature with:

```
    COPCTL=0x00; // disable COP reset
```

This first example disables the COP. The Red LED (PT6) on the Adapt812 will flash on/off.

```
// *********LED1.C****************
//  PT6 LED flashes on and off
#include "HC12.H"
void init(void){
  COPCTL=0x00;     // disable COP
  DDRT|=0x40;}     // PortT bit 6 is output to LED
void main(void){ int i;
  init();     // initialize COP, Port T
  while(1){ i++;
    if(i>0)
       PORTT |= 0x40;  // LED is on
    else
       PORTT &= ~0x40;}} // LED is off
extern void _start();
#pragma abs_address:0xfffe
void (*reset_vector[])() = { _start };
#pragma end_abs_address
```

In this second example, a background TOF interrupt is requested every time the TCNT register overflows from 0xFFFF to 0x0000. The ritual `TOFinit()` will activate TOF periodic interrupts. The software interrupt handler will keep the COP happy. This TOF interrupt also toggles bit 6 of Port T, flashing the red LED. The TOF interrupt vector is right after the interrupt service routine, and the reset vector is defined right after the main program.

```
// *********LED2.C*********************
//  PT6 LED flashes on and off
#include "HC12.H"
void TOFinit(void){
  DDRT|=0x40;  // PortT bit 6 is output to LED
  TSCR=0x80;   // TEN(enable)
  TMSK2=0xA5;  // TOI arm, TPU(pullup) timer/32 (4us)
  CLKCTL=0x00;
asm(" cli");}  // enable interrupts
#pragma interrupt_handler TOFhandler
void TOFhandler(void){
   TFLG2=0x80;           // TOF interrupt acknowledge
   PORTT^=0x40;          // toggle bit 6
   COPRST=0x55;          // make COP happy
   COPRST=0xAA;}
#pragma abs_address:0xffde
void (*TOF_vector[])() = { TOFhandler };
#pragma end_abs_address
void main(void){
  TOFinit();     // Enable TOF interrupt to make COP happy
  while(1){
  }
};
extern void _start(); /* entry point in crt12.s */
#pragma abs_address:0xfffe
void (*reset_vector[])() = { _start };
#pragma end_abs_address
```

Normally we will start our Adapt812 software by pushing the reset vector, which runs our software in normal single chip mode. When our Adapt812 software is started with the BDM-12 module, then it is possible to run software in *special single chip mode.* We can run our software in special single chip mode by typing the following lines using the BDM-12 debugger.

```
reset
load LED2.S19
g F000
```

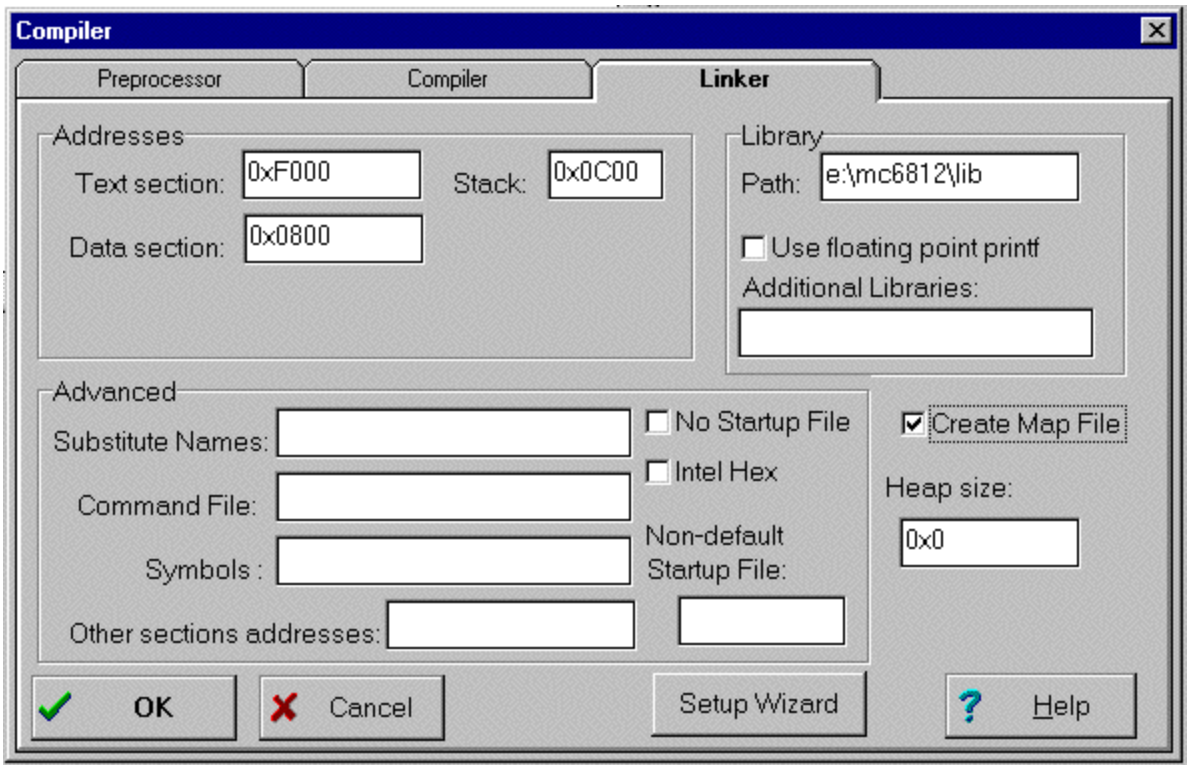
There are many good examples on the web at `http://www.ece.utexas.edu/~valvano/programs`

MEALY12.C, MOORE12.C, MOORE2.C	finite state machines (Valvano book Chapter 2)
DSTEST.C, DS1620.H, DS1620.C	bit bang interface with DS1620 (Valvano book Chapter 3)
RTI.C	Periodic interrupt (Valvano book Chapter 4)
THREAD3.C	thread scheduler with spinlock semaphores (Valvano book Chapter 5)
PWMOD.C	Pulse width moduation interrupt (Valvano book Chapter 6)
SCIDEMO.C, SCI12.H, SCI12.C	simple serial port I/O (Valvano book Chapter 7)
SCI12A.C, RXFIFO.*, TXFIFO.*	interrupting serial port I/O (Valvano book Chapter 7)
DSTESTB.C, DS1620.H, DS1620B.C	SPI interface with DS1620 (Valvano book Chapter 7)
LCDTEST.C, LCD12.H, LCD12.C	simple interface to Hitachi HD44780 LCD (Valvano book Chapter 8)
OC3TEST.C, OC3.H, OC3.C	Periodic interrupt, stepper driver (Valvano book Chapters 6,8)
ADTEST.C	Data acquisition system using an ADC (Valvano book Chapter 11)

I.4.2.3. ICC12 options menu for developing software for the Adapt812

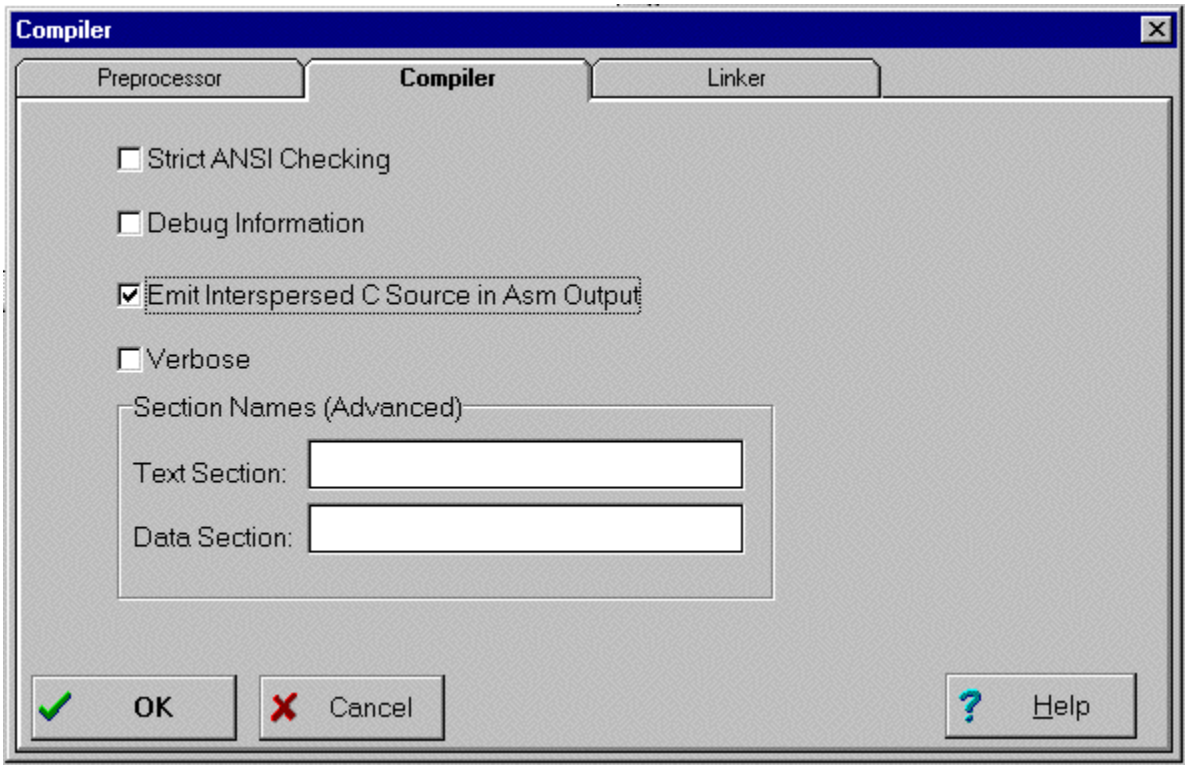
On the ICC12 development system, we use the *options_compiler_linker* menu command to specify where the start of globals (the data section grows towards higher addresses), and the start of the object code (the text section grows towards higher addresses). The initial stack pointer (stack grows towards lower addresses), is specified in *options_compiler_linker* menu. On our Adapt812 system,

the globals (data) should start at 0x0800
the program (text) should start at 0xF000
the stack could be set to 0x0C00
the Library path is set to e:\mc6812\lib (ask you TA about this field)
the “Create Map File” button is selected
the “Heap size” is 0.



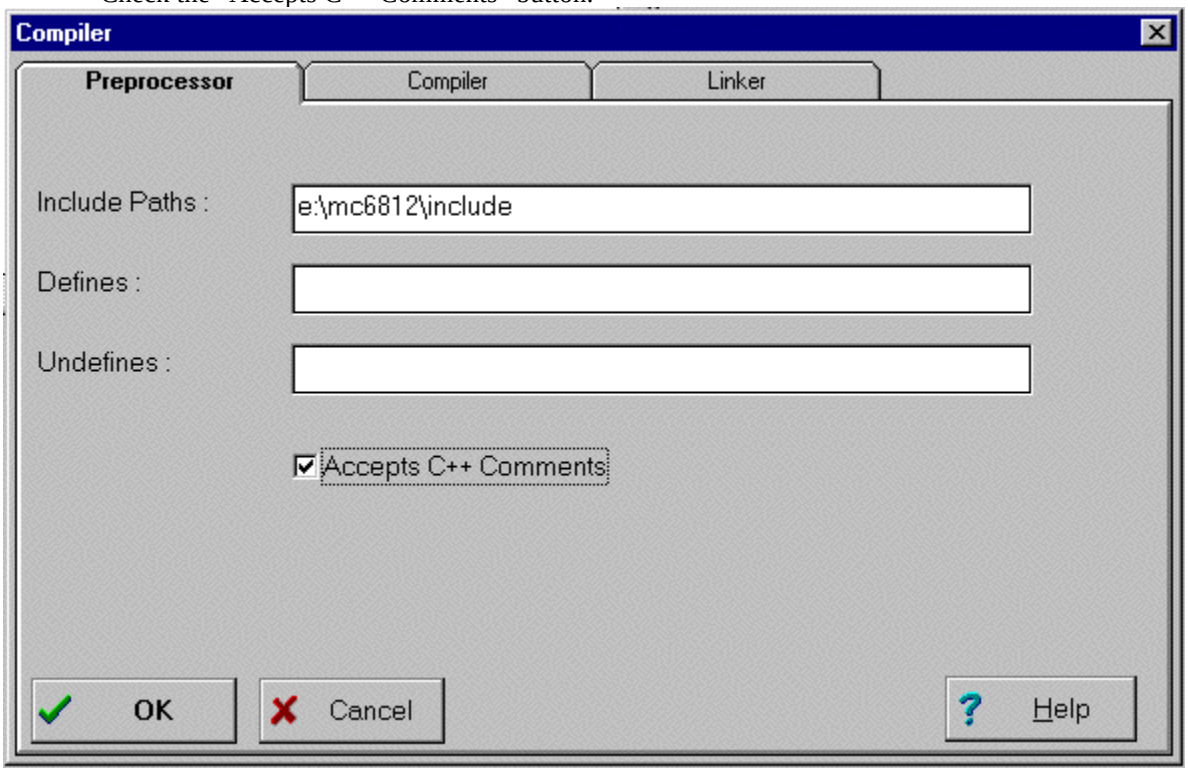
We use the *options_compiler_compiler* dialog to specify that C source code be added to the Asm Output On our Adapt812 system,

Check the “Emit Interspersed C Source in Asm Output” button.



We use the *options_compiler_preprocessor* dialog to specify that C++ comments can be used. On our Adapt812 system,

the Include path is set to e:\mc6812\include (ask you TA about this field)
Check the "Accepts C++ Comments" button.



I.4.2.4. Backing up software onto your floppy

One of the things you need to do often is copying files between the floppy and the hard drive. A batch file is a convenient way to do this. Create a text file on your floppy called `LOD.BAT` with the following contents:

```
copy A:\*.c D:\TEMP
copy A:\*.h D:\TEMP
copy A:\*.bat D:\TEMP
```

When you execute this file (from DOS or Windows File Manager) it will copy your 6812 development files from your floppy disk to the system hard drive. If you are working with other types of files (e.g., *.s or *.map) you can add them to the list.

Create a second file called `SAV.BAT` with the following contents:

```
copy D:\TEMP\*.c A:
copy D:\TEMP\*.h A:
copy D:\TEMP\*.bat A:
```

When you execute `SAV.BAT`, it will copy files from the hard drive back to your floppy disk. During software development you should rotate at least 3 floppy disks, so that you can return to previous states when you get a computer crash, when one file becomes corrupted, or when you make a software modification you wish to undo. Notice how the three floppy disks are used in a rotation. In each case, you load the files from the newest floppy, edit the files on the hard drive, then save the files onto the oldest floppy disk. When you have a software system that runs enough to allow a partial credit demonstration to the TA, put that disk aside and entire new floppy disk into the rotation. The following table shows the contents of the disks after each software development session.

time	session	disk A	disk B	disk C
1	create new programs, save on A	session 1	blank	blank
2	load from A, edit, save on B	session 1	session 2	blank
3	load from B, edit, save on C	session 1	session 2	session 3
4	load from C, edit, save on A	session 4	session 2	session 3

I.4.2.5. How to compile with a mixture of assembly and C files

The following C program embeds an assembly language file (programs and data). To access an assembly function, the C program simply calls it, with the standard ICC12 parameter passing rules. To access an assembly level global variable, the C program types it with the extern.

```
/* C level program  file="high.C"  Jonathan W. Valvano */
int highGlobal;
extern int lowGlobal;  // typed here but defined in low.s
asm(".include 'low.s' ");   // insert assembly here
void main(void){
  lowSub(5);      // call to assemble routine
  lowGlobal=6;    // access of assembly global
}
int highSub(int input){return(input++);}
```

The following assembly program is embedded into the above high level C program. The double colon, `::`, specifies the label as external and will be available in the *.map file. The `.area text` is the standard place for programs, and the `.area bss` is the standard area for globals. Assembly level functions (e.g., `_lowSub`) and variables (e.g., `_lowGlobal`) are defined beginning with an underscore, "_". To access a C function, the assembly program simply calls it (the name begins with an underscore.) The assembly program has full access to high level global variables (the name begins with an underscore.)

```
; assembly language program file="low.s" Jonathan W. Valvano
    .area text
_lowSub::            ; definition of low level subroutine
    jsr _highSub     ; call to high level function
    std _highGlobal  ; access to high level global
    rts
    .area bss
_lowGlobal::         ; definition of low level global
    .blkb 2
```

Again, parameter passing with both functions (the assembly calls to the C and the C calls to the assembly) must adhere to the standard ICC12 parameter passing rules:

- The output parameter, if it exists, is passed in Register D,
- The first input parameter is passed in Register D,
- The remaining input parameters are passed on the stack,
- 8-bit parameters are promoted to 16 bits.

If you are writing an assembly language function that is to be called from C, one method to get the parameter passing correct is to write a simple C function that simply passes the parameters. Compile this simple C function with your other C code, and observe the assembly language created by the compiler for the simple C function. Next draw a stack picture that exists at the start of the function. The C compiler will do some weird things within the function (like pushing register D on the stack, and shifting some 8 bit parameters around), which you do not have to duplicate. One difficulty with mixing the assembly with C is that when the compiler is upgraded, this compatibility matching must be redone. For more information on parameter passing see

`http://www.ece.utexas.edu/~valvano/embed/toc1.htm`

All source files used with ICC12 must end in a carriage return (ENTER.) We have noticed that the following error means the last character in this file is not a carriage return.

```
!E E:\ICC\STUDENTS\345M\LAB1/SCI12.C(352): illegal character `#'
!E E:\ICC\STUDENTS\345M\LAB1/SCI12.C(352): syntax error; found `54' expecting `;'
!E E:\ICC\STUDENTS\345M\LAB1/SCI12.C(352): unrecognized declaration
!E E:\ICC\STUDENTS\345M\LAB1/SCI12.C(352): unrecognized declaration
```

I.4.2.6. How to set interrupt vectors for ICC12

The reset vector tells the computer where to start after a power up or reset. The following code must be included if your system is to begin execution on power up, or is to restart execution after the reset button is pushed.

```
extern void _start(); /* entry point in crt12.s */
#pragma abs_address:0xfffe
void (*reset_vector[])() = { _start };
#pragma end_abs_address
```

An interrupt vector tells the computer what interrupt service routine to execute on a hardware or software interrupt. Normally, we define the interrupt vector right after the interrupt service routine itself. For example,

```
#pragma interrupt_handler TOFhandler
void TOFhandler(void){
   TFLG2=0x80;           // TOF interrupt acknowledge
   PORTT^=0x40;          // toggle bit 6
}
#pragma abs_address:0xffde
void (*TOF_vector[])() = { TOFhandler };
#pragma end_abs_address
```

There is a ICC12 Version 5 bug, such that two `#pragma abs_address` commands can not follow each other. For example this code will not compile under ICC12 V5 (you get a very weird error)

```
#pragma interrupt_handler TOFhandler
void TOFhandler(void){
   TFLG2=0x80;           // TOF interrupt acknowledge
   PORTT^=0x40;          // toggle bit 6
}

#pragma interrupt_handler OC5handler
void OC5handler(void){
   TFLG1=0x20;           // C5F interrupt acknowledge
   TC5=TC5+1000;
   PORTT^=0x40;          // toggle bit 6
}
#pragma abs_address:0xffe4
void (*OC5_vector[])() = { OC5handler };
#pragma end_abs_address

#pragma abs_address:0xffde
void (*TOF_vector[])() = { TOFhandler };
#pragma end_abs_address
```

On the other hand, this code will compile (I moved the #pragma)

```
#pragma interrupt_handler TOFhandler
void TOFhandler(void){
   TFLG2=0x80;           // TOF interrupt acknowledge
   PORTT^=0x40;          // toggle bit 6
}
#pragma abs_address:0xffde
void (*TOF_vector[])() = { TOFhandler };
#pragma end_abs_address

#pragma interrupt_handler OC5handler
void OC5handler(void){
   TFLG1=0x20;           // C5F interrupt acknowledge
   TC5=TC5+1000;
   PORTT^=0x40;          // toggle bit 6
}
#pragma abs_address:0xffe4
void (*OC5_vector[])() = { OC5handler };
#pragma end_abs_address
```

I.5. Hardware/Software Development in Lab

I.5.1. Power

The Adapt812 board can accept power from one of two sources. We will use an unregulated AC adapter (greater than 6 volts) connected to the power connector J1. The other option for the Adapt812 would be to connect a regulated +5v supply to the H1 connect (we will not be doing this.) The BDM-12 helper board received its power from the Adapt812 through the BDM cable.

I.5.2. Development Procedure

The following steps are required for the project development:

1. Design the software
2. Design the hardware (if any)
3. Using ICC12, write (modify) the code
4. Remove compiler or assembly errors from the code, and also any logical errors in the code which you think might be present, print out the MAP (or MP) file to assist in debugging
5. Build the hardware(if any) on your own protoboard
   the Adapt812 board should remain plugged in your prototype board for the entire semester.
6. Eliminate as many hardware errors as possible before connecting to the 68HC812A4.
7. With the power of the PC and BDM-12 off, connect a PC serial port to the BDM-12 helper board
8. Start the Kevin Ross BDM software in a DOS window. The batch file B.BAT will execute DL12.EXE with a baud rate of 38400. You must type `reset` to the BDM, and hit the reset buttons on the BDM-12 and Adapt812 boards until the `status` command returns a **C0**.

```
reset
status
```

9. Down load the object code of your software into the 68HC812A4. Do not attempt to download the object code until the `status` command returns a **C0**. Any S19 file can be downloaded. If the BDM-12 red light comes on, hit reset on both boards and try again.

```
load LED2.S19
```

10. Start execution. You can start your 6812 in normal mode by hitting the Adapt812 reset button, or by turning the power from off to on. You can start your 6812 in special mode by typing.

```
g F000
```

11. Debugging. You can use the BDM-12 to observe global memory including I/O ports while your software is running.

```
d 0800
```

while stopped DL12 can observe/modify memory and registers

I.5.3. The address map of the Adapt 812 development board

same as MC68HC812A4 in single chip mode.

I.5.4. Adapt 812 Board Jumpers

JB1 has two jumpers for MODA and MODB. They both should be in the 0 position. This will bring the MC68HC812 up in single chip mode, placing the 4K EEPROM from $F000 to $FFFF. There is a run/boot switch that should be in the “run” position. There is a boot loader in high EEPROM memory that can be used to program the Adapt812 from a serial port without the need for a Kevin Ross BDM. This procedure places the run/boot switch in the “boot” position.

I.6. Hardware/Software Development at home

I.6.1. Software

If you have a PC at home it is possible to develop software at home. All you need is a power supply, a serial modem cable, and an assembler or compiler. ImageCraft ICC12.EXE IS NOT FREEWARE. To use ICC12 at home, you will need to purchase a copy of the compiler. The TExaS simulator includes a 6812 assembler that can generate S19 records for downloading. There is a free ICC11 C compiler on the TExaS CD. You will have the option of simulating some labs (ask you TA about which ones are possible.) With this option, the 6811 chip and the ICC11 compiler can be run at home with no cost to you. On the TExaS CD is a limited-version of the Hiware C compiler. It has some interesting limitations (10 objects and 1000 bytes of object code) so ask your TA about how to develop software with this compiler. Unless you program is very small, you will find it difficult to develop

with the limited-version of Hiware. For a sense of how large a program you can develop with the limited-version, look at the Hiware 6812 examples in the Hiware12 folder of the TExaS application (after TExaS is installed.)

I.6.2. Power Adapter

To run the Adapt812, you will need 5.5 to 9 VDC unregulated power source with a current of at least 100 mA, plus the additional current your external circuits need. A good choice is a +6V AC adapter at 300 to 500 mA. If the voltage is too high then the regulator on the Adapt812 will get hot. Test your power source while attached to the Adapt812 to guarantee the supply voltage is less than +9 V, and the Vcc power on the H1 connector is about +5.0 V.

I.6.3. Serial Cable

The cables between the PC and the development boards are standard 9 pin null modem serial cables.

I.6.4. Downloading to the Adapt812

There are three ways to program the Adapt812. For about $48, you can purchase your own Kevin Ross board in kit form. With this board you download using the DOS-level application DL12.exe. Second, you could check out a Motorola MC68HC912B32 EVB board. The Hiware C compiler supports this configuration. I.e., the PC connected to a 912EVB in POD mode, connected to the 812A4 target. Lastly, the Adapt812 board has a boot loader that can accept S19 records directly from the PC. The disadvantages of this last method are there is no verify and no debugging support (but it is cheap.)

In short, without spending about $210 ($165 for the ICC12 and $48 for the Kevin Ross), you won't be able to much more that edit at home. There are two exceptions to this rule:

- If you use the free Hiware compiler both in lab and at home
- If you perform a lab with the simulator (either in 6812 assembly or 6811 C)

I.6.5. Need more program space?

If you need more program space then buy or borrow a 32K by 8 bit static RAM chip. The pin numbers for the 32K by 8 bit static RAM are

```
Pin 1 - A14             Pin 28 - Vcc (+5V)
Pin 2 - A12             Pin 27 - WE (negative logic)
Pin 3 - A7              Pin 26 - A13
Pin 4 - A6              Pin 25 - A8
Pin 5 - A5              Pin 24 - A9
Pin 6 - A4              Pin 23 - A11
Pin 7 - A3              Pin 22 - OE (negative logic)
Pin 8 - A2              Pin 21 - A10
Pin 9 - A1              Pin 20 - CE (negative logic)
Pin 10- A0              Pin 19 - D7
Pin 11- D0              Pin 18 - D6
Pin 12- D1              Pin 17 - D5
Pin 13- D2              Pin 16 - D4
Pin 14- Gnd             Pin 15 - D3
```

Connect the 32K RAM to the 68HC812A4 like figure 9.60, except use CSP0 inplace of CSD. The memory bus is available on the Adapt812 H2 connector:

6812 Ports A, B contain the 16-bit address A15-A0,
6812 Port C contains the bidirectional 8-bit data bus,
6812 Port E contains the E clock and R/W,
6812 Port F contains the built-in chips selects.

In narrow mode the data is on PORTC, but is labeled as D15-D8 on the Adapt812. The jumpers MODA and MODB are on the Adapt board (change MODA from off to on), remember to put MODA back before you turn the board in. Within ICC12, change the text (program) option from F000 to 8000. You now have 32K of program space. If you need to download, and untether the system from the PC power, then power the Adapt board with batteries (6V into the power connector) or 5V directly into the 6812.

I.7. Web sites

cool web site: **http://www.interlog.com/~techart/myfiles/links.ltml**

see 6812 products at: **http://www.interlog.com/~techart/**
electronic parts at: **http://www.bgmicro.com**
robot parts at: **http://www.RobotStore.com**
ICC11/ICC12 at: **http://www.imagecraft.com**
Writing in C for ICC12 at: **http://www.ece.utexas.edu/~valvano/embed/toc1.htm**
Writing in assembly at: **http://www.ece.utexas.edu/~valvano/assmbly/index.html**
links to companies at: **http://www.ece.utexas.edu/~valvano/book.htm**

I.8. Legal Stuff

The opinions expressed in these notes do not necessarily reflect the opinions of the University, its management or its big time financial donors. Also, there shall be no bologna, Bevis, mustard, chewing the cables, free lunch, sob stories, running & screaming, whining, hitting, spitting, kicking, biting, or tag backs. Quit it or we're telling. (Enjoy the course.)