

AWC

**APP-IV ATmega
Development Kit**

© 2004 by AWC

AWC
1279 FM 518 Rd #2
Kemah, TX 77565
(281) 334-4341
<http://www.awce.com>
V1.1 10 August 2004

Table of Contents

Overview	1
If You Need Help	1
What Else You'll Need	1
Features	2
Assembly	2
Testing	4
Programming	5
Using AVRStudio 4	5
Using gcc	7
About Cygwin	9
Debugging C with AVRStudio 4	9
Debugging with Insight	9
Debugging Issues	10
Other Languages	11
System Environment	12
Resources	12
Connections to JP1/JP2	13
C Language Libraries	14
I/O	14
Delays	14
A/D	15
Uart	15
LCD	16
Demo	16

Overview

The APP-IV allows you to develop ATmega (Atmel) code for the powerful ATmega 8 microcontroller. The kit includes a special ATmega 8 (28 pin microprocessor) that operates at 10MHz and a 10MHz resonator. It also includes a PC board (the GPMPU40) which allows you to provide power to the board, connect an RS-232 cable to the board, and optionally plug the board into a standard solderless breadboard. See the enclosed manual for the GPMPU40 for more information about this board (including assembly instructions).

If You Need Help

If you require assistance, please feel free to contact us. The best way to get support is via e-mail (stamp@al-williams.com). However, you may also call between 9AM - 4PM Central Time at (281) 334-4341. You can also fax to (281) 754-4462. Be sure to check out our Web page for updates at www.al-williams.com/awce.

What Else You'll Need

In addition to the APP-IV kit, you'll also need a few other easy to obtain items:

- A solderless breadboard
- An unregulated power supply (DC between 8 and 13V) or a 5V regulated power supply.
- Development software (assembly and C options are described later in this manual)
- A serial cable (a DB9 male to female, if you are using a PC).

Features

The ATmega 8 core of the APP-IV has A/D inputs, digital I/O, and a hardware serial port usable by your programs. The device has 1K bytes of RAM, 512 bytes of EEPROM, and nearly 8K of program space (512 bytes are reserved). The chip achieves speeds of 10 MIPS with the supplied ceramic resonator.

Assembly

Please refer to the enclosed GPMPU40 manual for assembly instructions. There is only one modification required on the board. Place a two pin male header in pins 4 and 5 of the JP7 connector. Leave the other holes of this connector empty. Remember, pin 1 of the ICSP connector is closest to the mounting hole at the corner of the board – be careful not to get this connector backwards.

The two-pin header you install will allow you to select programming mode or execution mode by placing (or removing) a jumper on these two pins. With the jumper in place, the APP-IV will allow programming (via the download software). When the jumper is not present, the APP-IV will run your program.

The small pushbutton switch provided fits in the RESET holes instead of a two pin jumper. There are no connections required for JP6, nor are any capacitors required for C7 and C8 (these are built into the ceramic resonator which installs at X1). Note that the resonator can go in facing either direction. The center pin is ground and the outer two pins are interchangeable. T1 is not required. Be sure that the five electrolytic capacitors are installed properly (note the + marking on the PC board).

The APP-IV's CPU (an ATmega 8) is installed so that pin 1 of the IC lines up with pin 1 of IC1. Note that the 28 pin IC fits in the inner set of holes. You can use the outer set of holes along with JP3 and JP4 to make connections between the CPU and other circuitry on the board. Here are the connections you must make from the GPMPU40 subsystems to the APP-IV CPU:

Connection 1	Connection 2	Note
JP5-T	JP5-2	RS232 connection
JP5-R	JP5-3	RS232 connection
RS-T	IC1-3	TX
RS-R	IC1-2	RX
RST (either pin)	IC1-1	Reset
RST (either pin)	JP8-1 ^{***} (closest to mounting hole)	Reset for optional programmer
Vcc	IC1-7, IC1-32*, IC1-33*	+5V
Ground	IC1-8, IC1-34**	Ground
CLK (either pin)	IC1-9	Clock
CLK (either pin)	IC1-10	Clock
JP8-2	IC1-29 ^{***}	MOSI (pin 17)
JP7-3	IC1-30 ^{***}	MISO (pin 18)
JP7-4	IC1-31	SCK/Program select (pin 19)
JP7-5	Ground	Ground
JP7-6	Vcc	+5V for optional programmer

*IC1-32 and IC1-33 are AVcc and ARef; you may wish to make custom connections if you are using the A/D.

**IC1-34 is AGnd; you may wish to make a custom connection if you are using the A/D.

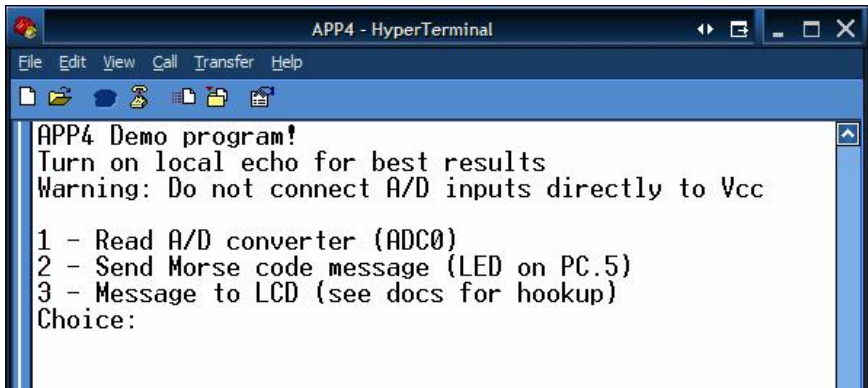
***These connections are only used for a typical AVR programmer like our XCP-1 board. You do not need this type of programmer and, in fact, using one will erase the APP-IV firmware. To use such a programmer, you'd need to install a 6 pin header at JP8.

Testing

The APP-IV is shipped with a test program already on board. To run the test, you'll need to connect the board to a computer running a terminal program with a normal straight cable. In addition, if you want to run all of the test program's functions (which is not strictly necessary) you'll want to connect an LED (with appropriate dropping resistor; say 470 ohms) to PC.5 (pin 28). The "banded" end of the LED will connect to ground and the other end will connect to PC.5 through the resistor. If you wish to experiment with the A/D converter, you can connect ground or a voltage less than 5V to pin 23. Do not connect pin 23 directly to 5V! A breadboard is useful for making these connections. However, you don't need the LED or analog connections to see that the board will pass the test.

Remove the shorting cap on the programming header and connect a PC running a terminal program (such as Hyperterminal) to the serial port. The terminal program should be set up for 19200 baud, 8 bits, 1 stop bit, no parity, and no handshaking (very important). Of course, you should set it for the COM port you are using with the APP-IV.

When you power up the APP-IV, you should see a display like the one below:



```
APP4 - HyperTerminal
File Edit View Call Transfer Help
APP4 Demo program!
Turn on local echo for best results
Warning: Do not connect A/D inputs directly to Vcc

1 - Read A/D converter (ADC0)
2 - Send Morse code message (LED on PC.5)
3 - Message to LCD (see docs for hookup)
Choice:
```


Congratulations! Your APP-IV is working. Now you can do some programming.

Programming

The APP-IV emulates a standard AVR910 serial programmer. That means that you can use any compatible download program to send a hex file to it. You can use WinAVR (supplied with AVR Studio), UISP, or AVRdude. We recommend UISP, although any of these programs will work, and any software that supports the AVR910 protocol should work.

You'll use a development tool such as AVR Studio or gcc to produce a programming file. This is usually a .hex (Intel format) or .s19 (Motorola format) file. The programmer will download this file down to the APP-IV. To enter programming mode, you must install the jumper on pin 4 and 5 of JP8 and press the reset button.

Using AVRStudio 4

You can download an example AVR Studio project from our Web site (see <http://www.awce.com/app4.htm>). You'll also find detailed instructions there. Once the project is built, you can start AVRProg from the Tools menu and follow this procedure to download the program:

1. Start AVRStudio 4.
2. Use the Welcome dialog or the Project | Open menu to open the ademo.aps project you downloaded from the Web.
3. Use the Project | Build menu to build the project; observe that there are no errors.
4. Make sure the APP-IV is powered up, the programming jumper is installed, and you've pressed the reset button. Also, be sure no other programs have the serial port open (including another copy of AVR Prog).

5. Select Tools | AVR Prog... from the menu.
6. Use the Browse button to select the ademo.hex file (if you've done these steps before, it will already be selected, in which case you can skip this step).
7. Press the (Flash) Program button. You may get a message about "Flushing" which you can dismiss.
8. After completion, use the "X" button in the window's title bar to close AVR Prog. Do not use the Exit button.
9. Remove the jumper cap and press reset to start your program.

Keep in mind that you may find it easier to use uisp or AVRdude, both of which can program the hex file. A typical uisp command line would look like this (all on one line, of course):

```
uisp -dprog=avr910 -dpart=auto -dserial=/dev/com1 --erase --upload  
if=ademo.hex --verify -v
```

For AVRdude, you would issue the command:

```
avrdude -patmega8 -P/dev/com1 -c avr910 -Uf:w:ademo.hex:i
```

Keep in mind that in either case some versions of Cygwin will expect you to use com1 instead of /dev/com1.

Using gcc

You can download gcc for Linux or WinAVR (pronounced “whenever”) for Windows by referring to the Resources section. In either event, you’ll be able to write C programs with this powerful compiler. At <http://www.awce.com/app4.htm> you will find a sample makefile plus several library files:

1. app4io – Simplifies digital I/O operations
2. app4delay – Programs various delays easily
3. app4adc – Read the A/D converters
4. app4uart – Read and write the serial port
5. app4lcd – Drive a standard 4-bit LCD

If you’ve written a C program (or want to use the demo program from the Web site) you need to follow these steps:

1. Open a shell and navigate to the directory that contains the C project
2. Open makefile with a text editor and change the PROGPORT line to indicate the port you are using to talk to the APP-IV. If you are not using uisp, change the PROGRAMMER line accordingly.
Notice that these lines will not require future changes unless your setup changes.
3. If you are not using the demo program, copy the demo program’s makefile to your project directory and make the following changes:
 - a) Change TARGET to reflect the name of your project
 - b) Change the SRC+= line to choose other files to compileYou may wish to verify that PROGPORT and PROGRAMMER are set correctly. You may wish to

change some of the other lines, but for most projects they will be fine as they are.

4. Run “make” – this will build your program and generate a hex file.
5. Run “make program” – this will program the chip (and build any files that are out of date). The makefile will prompt you to set the jumper and press reset on the board.
6. Remove the jumper cap and press reset to start your program.

The program keyword is a “make target” that tells make not only to build the program, but to do other commands (in this case, program the chip). There are several targets available:

- all – Do everything.
- clean – Erase output files.
- coff – Make coff debugging file (use with AVR Studio 3).
- extcoff – Make extended coff (use with AVR Studio 4).
- program – Program device.

In addition, if you provide the program name with a .s suffix as a make target, make will produce a file of assembly language instead of a hex file. For example, if you run “make demo.s” on the demo project, the file demo.s will have the assembly language equivalent of the C program.

About Cygwin

If you use WinAVR, the installer will load a copy of the Cygwin environment to your PC. Cygwin is an open source program that makes your Windows operating system behave like Linux. There are two things to be wary of with Cygwin. First, if you already use Cygwin, you'll have to rename (or otherwise hide) WinAVR's cygwin1.dll file or all of your Cygwin programs will be confused because they will find two copies of the Cygwin DLL. If you encounter unexpected problems, you can try taking your normal Cygwin installation off of your PATH and reverting to the one that ships with WinAVR. It is possible that the version of Cygwin you use does not work properly with WinAVR, although usually any newer version will work.

The other thing you should note is that some versions of Cygwin name the serial ports using /dev. So to specify COM1 to UISP or AVR Dude, you may have to use /dev/com1 or you may have to use com1. If in doubt, try both and see which one works.

Debugging C with AVRStudio 4

It is possible to debug C programs using AVRStudio 4 under Windows. You must first create an extended coff file using the makefile. Simply run "make extcoff" from the shell.

Start AVRStudio and (for the demo program) open demo.cof. From the resulting dialog, select AVR Simulator as the platform and ATmega8 as the device. Note this is a PC-based simulation. It does not debug the part in circuit. Press Finish and you are ready to debug C code using AVRStudio.

Debugging with Insight

You can also debug with gcc's debugger, although the simulation of AVR devices is not as complete as that of AVRStudio.

To start debugging, issue the command "make debug" which will do a build and launch the simulator plus insight (a graphical version of gdb, the debugger). If you are using Cygwin, you must

have X windows running for this to work. Also, if you are using Linux, you'll need to reconfigure the `START` variable in the makefile to be empty.

Once insight starts, you must follow these steps:

1. Select Run | Connect to Target. If the target settings box appears, select GDBserver/TCP as the target, localhost as the hostname, and 1212 as the port (the port should be 1212 – you can check the simulator window to be sure).
2. Select Run | Download to load the program into the simulator.
3. Do not use the Run | Run command!
4. To execute the program, use Control | Continue. You may want to set breakpoints first. You can view the files in your project by selecting them from the leftmost combo box.
5. When you are done debugging, close insight and the simulator window separately.

Debugging Issues

Keep in mind that debugging is a simulation and does not use the APP-IV at all. Therefore, the APP-IV doesn't even have to be connected to the PC for debugging. However, it also means that code that expects actions from I/O devices may hang. For example, UART code that is waiting for a received character will never complete. In some cases, you can use the simulator to “fake” an input bit. In other cases, you can pause the debugger, set the next statement to be past the portion that hangs, and then resume the program. Of course, you can also replace problem areas with debugging-specific code that provides simulated input.

Other Languages

Most other languages that can target the ATmega 8 will work with the APP-IV. AWC's SeaBass, for example, provides a Basic-like language that works in conjunction with the GNU C compiler. As another example, MCS makes the BASCOM/AVR Basic compiler (you can download a free demo version that is limited to 2K of program space). You can use BASCOM to write Basic programs for the APP-IV.

Although the BASCOM programmer supports the AVR910 protocol, it doesn't understand the signature for the ATmega 8, so you will have to use a different programmer as described above. If you don't want to manually operate the programming software, you can download the `bascompqm.bat` and `bascompqm.sh` files from our Web site. These files will allow you to use UISP from within BASCOM.

You'll need to edit the files to set your specific path information and COM port. Inside BASCOM, select Options | Programmer and pick "External Programmer" in the drop down box. On the "Other" tab, you'll enter either `cmd.exe` (for Windows NT/2000/XP) or `command.com` (for Windows 95/98/ME) in the Program box. For the parameters, you'll enter something like this:

```
/c c:\app4\bdemo\bascompqm.bat {FILE}
```

Naturally, you'll need to adjust the path to suit your system. The {FILE} keyword will be replaced by the correct file name by BASCOM.

The final step is to make sure the "Use HEX file" box is checked. Once these steps are complete, you can use UISP from within BASCOM to program flash memory. Note that if your program is using EEPROM data you may need to modify these scripts slightly.

System Environment

When using the APP-IV, the ATmega 8 is at your disposal with a few caveats:

1. The top of flash memory is at 0x1BFF
2. Pin 5 of PORTB is the program select pin. On reset, the device enables the pull up resistor on this pin and samples it. If you wish, you can use this pin as an output during program execution as long as the circuitry connected does not interfere with the sampling process on reset. You could also use the pin for an input if you were certain it would not be low during a reset (for example, a push button switch). However, for production use, we recommend not using this pin at all, but simply tying to ground to prevent accidental entry into program mode.

Resources

<http://www.al-williams.com/app4.htm> – Examples and files

<http://tutor.al-williams.com> - Tutorials

<http://www.atmel.com/products/avr/> -AVRStudio and AVRProg

www.avrfreaks.net/AVRGCC/ - Windows C language tools and programmers

<http://cdk4avr.sourceforge.net/> - Linux C language tools and programmers

<http://www.nongnu.org/avr-libc/user-manual/index.html> - C library documentation

<http://www.awce.com/seabass.htm> - Basic language tool

<http://www.avrfreaks.com> – Many links and resources

<http://www.openavr.org/> - Resources

Connections to JP1/JP2

For the purposes of this table, JP1 and JP2 are numbered sequentially from 1 to 40. The last pin of JP1 is pin 20, and the first pin of JP2 is 21.

JP1/2 pin	Signal	ATMega 8 Pin	Note
1	RESET	1	Connected to reset circuit
2	RX	2	Connected to RS232
3	TX	3	Connected to RS232
4	PD2	4	
5	PD3	5	
6	PD4	6	
7	Vcc	7	
8	Gnd	8	
9	XTAL1	9	Connected to clock
10	XTAL2	10	Connected to clock
11	PD5	11	
12	PD6	12	
13	PD7	13	
14	PB0	14	
27	PB1	15	
28	PB2	16	
29	PB3	17	MOSI (on JP8 for external programmer)
30	PB4	18	MISO (on JP8 for external programmer)
31	PB5	19	SCK (also used as program select jumper)
32	AVCC	20	
33	AREF	21	
34	AGND	22	
35	PC0	23	PC0-PC5 also analog inputs
36	PC1	24	
37	PC2	25	
38	PC3	26	
39	PC4	27	
40	PC5	28	

C Language Libraries

The APP-IV has several libraries you can download for free. Some of these have been adapted from public domain or open source libraries (see the header files for appropriate credits).

I/O

The `app4io.h` file contains several macros which simplify digital I/O (although they may not be as efficient as manually coding digital I/O operations):

`HIGH(port, pin)` – Set pin of PORT to high. The pin is forced to output status. Example: `HIGH(B,1)`;

`LOW(port, pin)` – Same as `HIGH`, but forces pin low.

`TOGGLE(port, pin)` – Same as `HIGH` but toggles output from high to low, or low to high.

`OUTPUT(port, pin)` – Make pin an output.

`INPUT(port, pin)` – Make pin an input.

`REVERSE(port, pin)` – Make an output pin an input or vice versa.

`IREAD(port, pin)` – Read input from port. Forces pin to an input and return 1 or 0.

`BIN(num)` – Specifies a number in binary Example: `x=BIN(1101)`;

Delays

The `app4delay.h` file allows you to easily implement delays:

`delay_ms(ms)` – Delay for ms number of milliseconds.

`delay_us(us)` – Delay for us number of microseconds.

A/D

The `app4adc.h` file contains functions that allow you to work with the A/D converter. If you set `ADC_NOISE_REDUCTION` to 1, the library will force other A/D pins to low digital outputs which significantly improves the measurement noise. However, it means you should not have anything connected to the A/D pins that would be damaged by shorting them to ground. In particular, do not connect +5V directly to the A/D pins when using this mode.

`adc_init()` – Initialize the A/D converter library.

`adc_convert(chan)` – Returns an integer reading for the specified channel (0-5 or `ADC_CH0` to `ADC_CH5`).

Uart

You can use the APP-IV's serial port by calling the routines in `app4uart.h`.

`UartInit(baud)` – Initialize the port at a particular baud rate (use defined constants like `BAUD_9600` or the `BAUD` macro). This sets 8 bits, no parity, 1 stop bit.

`UartRead(timeout)` – Reads a character. If `timeout` is 0, this call will not return until a character is available. If `timeout` is not 0, it indicates the number of loops the code will wait for a character. The `TOMULT` definition allows you to adjust the exact value of the timeout delay. Returns 0 in the case of a timeout.

`UartWrite(c)` – Write a character to the UART.

`UartReady()` – Returns 0 if no characters are available and non-zero if there is a character waiting to be read.

`UartSetStdio()` – Sets the UART to be the standard I/O device (for calls like `printf` or `gets`).

LCD

You can connect a standard 4-bit LCD to the APP-IV and communicate with it using routines defined in `app4lcd.h`. Use the macros in this file to describe your LCD and the connections to the APP-IV.

`lcd_init(set)` – Initialize LCD (see `app4lcd.h` for set values).

`lcd_clear()` – Clear the LCD.

`lcd_home()` – Set cursor to home.

`lcd_gotoxy(x,y)` – Set cursor position.

`lcd_putc(c)` – Write character to LCD.

`lcd_puts(string)` – Write string to LCD.

`lcd_puts_p(pstring)` – Like `lcd_puts`, but takes string from program memory.

`lcd_puts_P(string)` – A macro that automatically creates constant strings in program memory.

`lcd_setstdio()` – Sets stdout to LCD (and sets stdin to NULL). Writing a form feed (0xC) to the output stream will clear the LCD.

Demo

The demo program (available [online](#)) demonstrates most of the library functions.