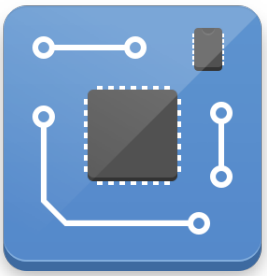
CrossPack for AVR® Development



CrossPack is a development environment for Atmel's AVR® microcontrollers running on Apple's Mac OS X, similar to AVR Studio on Windows. It consists of the GNU compiler suite, a C library for the AVR, the AVRDUDE uploader and several other useful tools.

Download

Features

- Does not depend on Xcode for building AVR code.
- Runs on Mac OS X 10.6 and higher.
- Supports 8 bit AVR microcontrollers including XMEGA devices.
- Includes patches to gcc for new devices not yet supported by gcc's main distribution.
- Includes gdb for debugging with simulavr and avarice.
- You can create your own version of CrossPack AVR based on the build script available on github.com.

For a list of included software packages and versions see the Release Notes.

Getting Started

Since CrossPack consists of command line tools only (except the HTML manual which is linked to your Applications folder), you need to know some basic command names. So let's demonstrate CrossPack with a trivial project, a blinking LED implemented on an ATMega8. This project is described in more detail in CrossPack's manual.

```
bash$ cd Desktop
bash$ avr-project Demo
bash$ cd Demo
bash$ ls -l
total 0
drwxr-xr-x   3 cs  cs  102 Nov 22 18:29 Demo.xcodeproj
drwxr-xr-x   4 cs  cs  136 Nov 22 18:29 firmware
bash$ cd firmware
bash$ ls -l
total 24
-rw-r--r--   1 cs  cs  4139 Nov 22 18:29 Makefile
-rw-r--r--   1 cs  cs   348 Nov 22 18:29 main.c
```

The command `avr-project` creates a minimum firmware project which is configured for an ATMega8 with internal RC oscillator at 8 MHz. Now we have something to start with. We edit main.c and implement the blinking loop:

```
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRD = 1 << 4;           /* make the LED pin an output */
    for(;;){
        char i;
        for(i = 0; i < 10; i++){
            _delay_ms(30);  /* max is 262.14 ms / F_CPU in MHz */
        }
        PORTD ^= 1 << 4;    /* toggle the LED */
    }
    return 0;               /* never reached */
}
```

Now we compile the code and send it to the device:

```
bash$ make
avr-gcc -Wall -Os -DF_CPU=8000000 -mmcu=atmega8 -c main.c -o main.o
avr-gcc -Wall -Os -DF_CPU=8000000 -mmcu=atmega8 -o main.elf main.o
rm -f main.hex
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
bash$ make flash
avrdude -c USBasp -p atmega8 -U flash:w:main.hex:i

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.19s
...

bash$ make fuse
avrdude -c USBasp -p atmega8 -U hfuse:w:0xd9:m -U lfuse:w:0x24:m

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.19s
...
```

That's it. The LED should now blink. For a real project you should also edit `Makefile` to configure your uploader hardware (e.g. STK500, USBasp, AVR-Doper or similar), other source code modules, fuse options etc.