

Chapter 1

User manual for AVR32 I/O

In June 2007 Øyvind Netland finished a master thesis[1] about how to use the new Atmel AVR32 processor architecture as a control system. In this work a STK1000 development board was used together with an I/O-card that was developed during the thesis. This I/O-card communicates with the STK1000 through SPI, and it has 6-channel analog output, 8-channel analog input, 8-channel digital output and 8-channel digital input.

This manual describes a control system platform that consists of AVR32 Linux running on a STK1000 or NGW development board from Atmel and the AVR32 I/O-card. This platform is used together with a Linux or Windows workstation that can use Matlab Real-Time Workshop as a rapid prototyping tool. Readers of this manual should have some experience with using Linux, since not all basic Linux commands are explained. Readers should also read the documentation for AVR32 Linux on AVR freaks Wiki <http://www.avrfreaks.net/wiki> and the AVR32 Linux project <http://avr32linux.org>.

1.1 Installing

Before using this system platform, some tools has to be installed on the workstation. This workstation can be either a Windows or Linux workstation that has Matlab Real-Time Workshop installed. The install instructions are based on a Ubuntu Workstation, but installing on other Linux distributions should be similar.

1.1.1 Install AVR32 tool-chain

For Ubuntu the AVR32 tool-chain can be installed with the Apt package management system. To add the AVR32 repository to the Apt sources, the following line has to be added at the end of `/etc/apt/sources.list`.

```
deb http://www.atmel.no/beta_ware/avr32/ubuntu/dapper binary/
```

To install the tools, execute the following commands, and answer yes to all questions.

```
~$ sudo aptitude update
~$ sudo aptitude install avr32-linux-devel
```

If this approach doesn't work, or instructions for installing on other platforms, check the AVR freaks Wiki (<http://www.avrfreaks.net/wiki>).

1.1.2 Compile Linux kernel with AVR32 I/O-card support

This system uses a patched version of the Linux kernel version 2.6.20. The patches needed are the 2.6.20-avr2 patchset from <http://www.avr32linux.org> and the avr32io patch from the CD-ROM. To patch, configure and compile the kernel with AVR32 I/O-card support, Use the commands below from a directory with the Linux source tarball and the two patches.

```
~$ tar xjf linux-2.6.20.tar.bz2
~$ cd linux-2.6.20
~$ patch -p1 < ../linux-2.6.20-avr2.patch
~$ patch -p1 < ../linux-2.6.20-avr32io.patch
~$ make ARCH=avr32 CROSS_COMPILE=avr32-linux- menuconfig
~$ make ARCH=avr32 CROSS_COMPILE=avr32-linux-
```

The second last command will open a menu based configure tool for the Linux kernel. To enable AVR32 I/O, the Atmel SPI Controller and AVR32 I/O-card has to be enabled. These are both found in `device drivers/SPI support`. Another solution is to use the `linux-2.6.20-avr32io-config` file from the CD-ROM and rename it to `.config` and move it to the linux source folder. The last command will compile the kernel and make an `uImage`, which is the kernel binary file.

To be able to use the AVR32 I/O-card support, the file system of AVR32 Linux needs some device nodes for this card. These are made by running a script as root on the workstation called `mknod.sh` that are on the CD-ROM. The script has to be modified to use the correct major number and file system path. The major number are printed out during booting, or can be found when running `more /proc/devices` on AVR32 Linux. The file system path is the path to the file system of AVR32 Linux, either a SD-card or a shared folder on the workstation.

1.1.3 Install AVR32 support in Matlab

Matlab Real-Time Workshop can generate code that with few modifications can be compiled for AVR32, but to make it easier to use, a new system target files have been created. These are located on the CD-ROM, in the folder `matlab/avr32`. The content of this folder should be copied into `matlab_root/rtw/c/avr32` on the workstation computer, where `matlab_root` is the folder where Matlab is installed.

When copying the `avr32` directory, the subdirectory `blocks` is also copied. This directory contain the S-functions that are used to control the I/O-card. These has to be compiled for Matlab. Start up Matlab and move to the `blocks` directory. If using Linux, Matlab has to be started as root. Run the commands below to compile the S-functions.

```
~$ mex avr32io_ain.c
~$ mex avr32io_aout.c
~$ mex avr32io_din.c
~$ mex avr32io_dout.c
~$ mex avr32io_ain_thread.c
~$ mex avr32io_aout_thread.c
```

If this gives an error, the mex compiler has to be configured. The following command starts a simple interactive configuration.

```
~$ mex -s
```

To use the S-functions in the library, this directory has to be in Matlabs path variable. To add the path use the following commands in Linux, or similar if Matlab is installed somewhere else.

```
~$ addpath /usr/local/matlab/rtw/c/avr32/blocks
~$ savepath
```

To add the path when using Windows, the following commands should be used:

```
~$ addpath c:\Program Files\Matlab\rtw\c\avr32\blocks
~$ savepath
```

Now the S-functions for the I/O-card are available in the library in the `blocks` directory, and they can be used in other Simulink models.

1.2 AVR32 I/O-card

1.2.1 Hardware description

AVR32 I/O-card is an add-on card for STK1000. It provides the following I/O:

- 8-channel analog input, with a resolution of 10 bits. The analog value can be between $0V - 5V$, where $0V$ equals ground.
- 6-channel analog output, with a resolution between 8 and 12 bits. 3 of the channels has output between $0V - 5V$, where $0V$ equals ground. The 3 other channels have jumpers for selecting output type, see 1.2.4.
- 8-channel digital input.
- 8-channel digital output.

Figure 1.1 show the front side of the I/O-card, below are a description of the different parts:

1. Screw clamps for ground and V_{CC} , that gives easy access to these signals.
2. Jumper for selecting if analog output channels $0.0 - 0.2$ should have $5V$ or $10V$ output.
3. Screw clamps for analog output signals.
4. Connector or screw clamp for external power supply. The supply should be between $9V$ and $18V$, and the polarity doesn't matter, because of a bridge rectifier.
5. Jumper for selecting if external power supply or supply from STK1000 should be used.
6. Operational amplifiers used as voltage followers or non-inverting amplifiers.

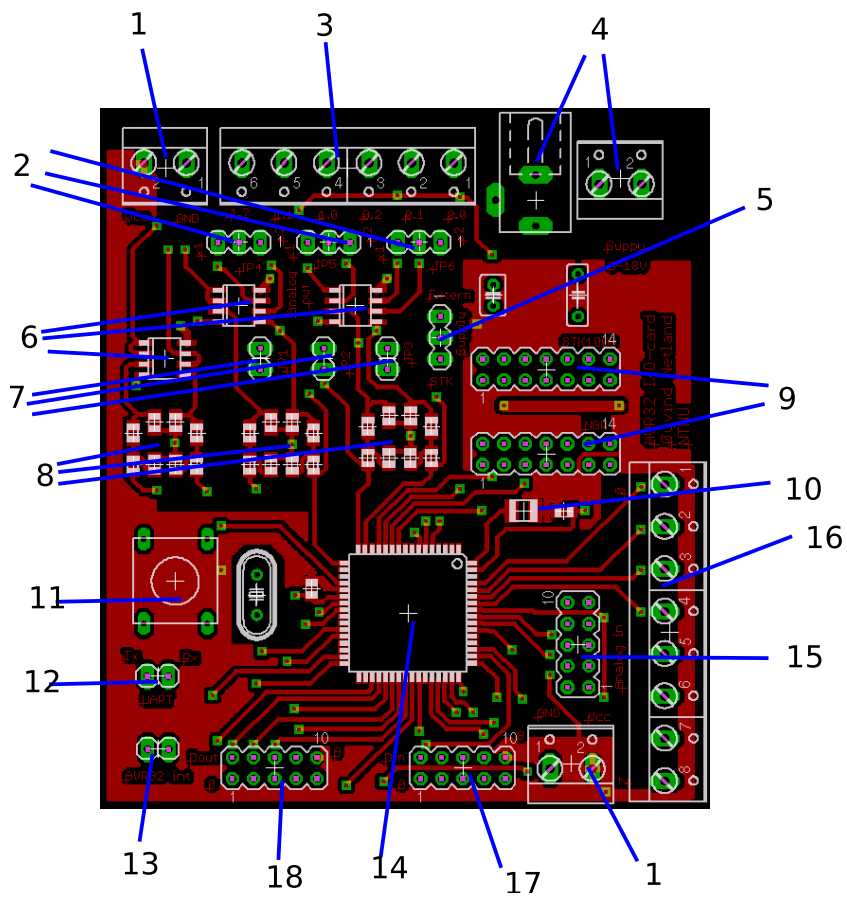


Figure 1.1: I/O-card

7. Jumpers for bypassing the filter which means that the analog output will become a PWM signal.
8. Passive filters that are used to convert a PWM signal into an analog voltage.
9. Headers for connecting STK1000 or NGW development boards.
10. LC-filter for the analog in power-supply.
11. Reset button
12. UART headers that can be used for debugging.
13. AVR32 interrupt signals, that can be connected to STK1000/NGW if interrupts are used. Not normally used, but might be useful for further development.
14. ATmega128 is the microcontroller that controls the I/O-card.
15. JTAG header for programming and debug ATmega128.
16. Screw clamps for analog input signals.
17. Header for digital input signals.
18. Header for digital output signals.

1.2.2 Connecting I/O-card

The I/O-card connects to either a STK1000 or NGW board through a 14 pin header cable. On STK1000 cards this header cable connects from the header marked *STK* on the I/O-card, to the general expansion header marked *J29* on STK1000. On NGW it connects from the header marked *NGW* to the general expansion header called *J5* on NGW. On both boards the header has to be connected to pins 0-13. If interrupts are used, interrupt signals from the I/O-card has to be connected to the STK1000 or NGW boards aswell.

1.2.3 Analog Input

The card has 8 Analog input channels with 10-bit resolution. Each of these can measure an voltage between 0V and 5V above *GND*. The I/O-card will continuously convert analog values into digital values and buffer them. Each analog input channel that are used increased the time a value is stored in the buffer before it's updated by about 110 μ s. This means that if four channels are used, the values are stored about 440 μ s before they are updated.

1.2.4 Analog Output

The card has 2 Analog output channels with 8-bit to 12-bit resolution. Each of these channels have 3 subchannels that can have different analog values. The analog outputs are generated with PWM timers and passive filters. This means that a high resolution will give high ripple on the analog output voltage. Table 1.1 contains the maximum ripple amplitude for different resoulutions.

Table 1.1: Rippe for different resolutions

Resolution	Ripple
8-bit	$10mV$
9-bit	$20mV$
10-bit	$60mV$
11-bit	$180mV$
12-bit	$500mV$

Channel 0 has jumpers that can select the output range of the analog output to be either $(0V - 5V)$ or $(0V - 10V)$. And other jumpers that can be used to bypass the filter, so the output becomes a PWM signal instead of a analog signal. These jumpers are on the card to make it possible control as many instruments as possible.

1.3 Rapid prototyping with Matlab Real-Time Workshop

1.3.1 AVR32 System Target

The AVR32 system target are used to easily configure Real-Time Workshop to generate and compile code correctly for rapid prototyping on AVR32. In addition to selecting this system target file, some Simulink preferences has to be changed. The solver has to be a fixed-step solver, since the generated code has to use a constant time step. It's also a good idea to specify the time step, since Matlab might try and use a lower period than the AVR32 is able to run. The minimum time step is $1ms$, but a complex system has to use a higher time step.

The code automatically logs the average period time, and the average time used for work each period, and this information will be displayed after completing the simulation. This makes it easy to find out if the periods are stable and if the AVR32 are able to complete each time step inside the period.

1.3.2 S-functions for I/O-card

Four S-functions have been made for each of the four I/O on the card, analog input, analog output, digital input and digital output.

The analog input has 8 channels, that gives a value between 0 and 5, which is an analog voltage between $0V$ and $5V$. Each channel that are connected is an active channel. This means that the generated code will read from the I/O-card, and this takes approximately $300\mu s$. If connected, a channel will be read, even if the value isn't used for anything useful.

The analog output has 2 channels with 3 subchannels each. Each channel can choose a resolution between 8-bit and 12-bit. The I/O-card will give out a voltage between $0V$ and $5V$, depending on the value of the S-function blocks input. Each channel connected to the will be written to each period, and it will use about $250\mu s$.

The digital input has 8 channels, and each channel gives out the value 0 if the channel is low and the value 5 if the channel is high. If any digital input channels are used, it will be

read each period and it takes $220\mu s$ no matter how many of the channels that are in use.

The digital output has 8 channels, and each channel can be configured with a threshold value. For all values over this threshold the digital output channel will give a voltage of $5V$, and all values under it will give a voltage of $0V$.

Bibliography

- [1] Øyvind Netland. Rapid prototyping with matlab real-time workhsop on an embedded control system with atmel avr32 processor. Master's thesis, NTNU, 2007.