An AVR programmer development base on the UNO

Introducing the UNO programmer

An Atmega 328 device pre-loaded with the Arduino bootloader and the UNO hardware make a very effective combination for developing and running C applications. It is also possible to program the UNO using a range of external programmers or with another UNO. The purpose of the development described here is to extent this flexibility, allowing the UNO to be used as a general purpose programmer for a number of the Atmega devices.

An Arduino application "UNO_programmer" is introduced that is hosted on the UNO and enables it to be used to program Atmega devices as follows:

Flash with a hex file loaded at address zero.
Flash with a text file loaded from the maximum address.
The configuration bytes.
The EEPROM with text and numbers.

Two Atmel Studio 7 applications are provided for an Atmega target device.

The first, "Cal_auto_plus_manual" is used to calibrate the target device.
The second, "text_reader" contains the subroutines needed to read text from the flash.
Sample text files are also provided.

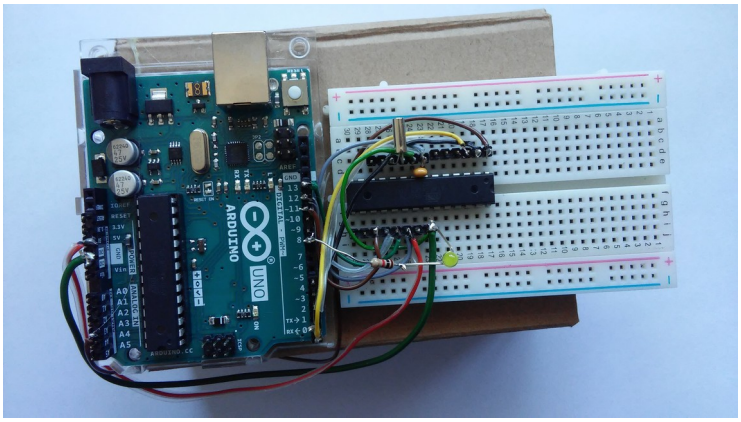
Once programming is complete, the target device is automatically put into the run state and the UNO reset button must be pressed to reinitialise the programmer.
Note: Atmel Studio has been chosen because it can build projects for any Atmega device.

UNO programmer Hard Ware.

A photograph of the HW is shown below. The target device is loaded into some plug-in prototyping breadboard and the following connections between the UNO and target device are made:

UNO output		Atmega 328 target pin and pin number	
5V		5V	7 & 20
0V		0V	8 and 22
Rx		USART TXD	3
Tx		USART RXD	2
Digital 11	MOSI	MOSI	17
Digital 12	MISO	MISO	18
Digital 13	SCK	SCK	19

In addition the following extra components are used;
A watch crystal between pins 9 and 10 (TOSC1 and TOSC2) of the target device
A 100nF capacitor between pins 7 and 8 of the target device
A led and 1K resistor between digital output 8 (PB0) of the UNO and 0V.



UNO Programmer HW:

The UNO and breadboard are mounted on a cardboard box.

Note especially the 100nF capacitor beside the crystal. This is particularly important for obtaining an accurate user OSCCAL value.

A user interface for the UNO programmer

A PC application normally used for sending hex files to the target processor hardware has not been written. Instead a terminal program is used and the entire hex file is sent to the UNO. Arduino comes with a basic terminal program and there are several others on the web.

My favourite can be downloaded from https://sites.google.com/site/terminalbpp/. But take care to download the version 20130820, other versions may have an issue with the "scroll" button. Settings for the terminal program are: Baud rate: 38400, 8 data bits, no parity, 1 stop bit and no handshaking.

Running the "UNO_programmer"

Connect the UNO to the PC and open the Arduino application "UNO_programmer". It should upload OK. Check the comm port number if there are problems.

Open the terminal program. User prompt "s s s s ….." should be generated.
Left click the mouse in the grey area at the bottom of its window and press 's'.
The programmer should respond with the message "Target_not_detected".

Complete the test jig and repeat. This time the response should be either

"Atmega unknown"
or
"Atmega 168/P detected.
Press -p- to program flash, -e- for EEPROM or -x- to escape."

Assuming that the target device is an Atmega 168 or 168P.

Programming hex files

Press 'p' to program a hex file.
Click the "Send File" box and send a hex file when requested.
Press '0' at the "Integer(0-FF)?" prompt.
The file size and config bytes should be printed out.

Note: The config bytes have not been programmed.

Programming text files

If 'y' is pressed at the "Text_file? y or n" prompt a text file will be requested and immediately echoed back to the terminal program.
Which ever is pressed the program will automatically be launched.
Note that the baud rate must temporarily be reduced while the text is being uploaded.

Target device calibration

The application “Cal_auto_plus_manual” stores calibration bytes in the top two addresses of the target device eeprom. User applications can do without the crystal. Instead they read these bytes and use them to replace the default value of the OSCCAL register.
There is of course no need to restrict the target device to using its internal RC oscillator. However it is the default selection and is in many cases very convenient to do so.

Running the “Cal_auto_plus_manual” applications

Connect the programmer to the PC and open the terminal program.
Download the hex file as described above.
There is no text file.
Auto cal will start running automatically and the following dialogue will be printed.

“Calibrating Atmega 168/P
New OSCCAL value 78 (for example)
Press 'x' to finish or Any Other Key for manual cal”

Reading text from the Flash

Upload the application “Text reader” followed by the text file.
Note the temporary change in baud rate required by the text programmer.
The text strings are echoed to the terminal program in numbered format.
Three subroutines are introduced:

“string_counter()”: This counts the number of strings programmed to the flash.
“print_string_num(string_num)”: This prints out any numbered string.
“Char_from_flash()” which implements the “lpm” (load program memory) command.

Note:
Each string is terminated by a null character and the final one is terminated by two.
Both subroutines also need to know the size of the flash in words (3FFF for the Atmega 328).

Programming the EEPROM

At the “s s s…….” prompt press ‘s’, ‘e’ and ‘w’.
Press ‘0’ at the “Integer(0-FF)?” prompt and send the file.
Note that the file has to be sent several times however there is no need to change the baud rate.

Each string is terminated in a null character and is echoed to the screen together with its address.
User applications will require these addresses so that they can print the strings out.

Configuring the programmer for a range of target devices.

Details of the target device must be entered in two subroutines called by the “UNO_Programmer”.

Both are contained in:

File Resources_UNO_programmer/Device_characteristics.c”

Details have already been entered for the Atmega 48, 88, 168, 328 and 644 devices.
It is up to the user to ensure that details for their chosen device are entered.

Target device data required by the programmer

To program text, data and the calibration bytes

EE_size	The amount of EEPROM
FlashSZ	The amount of flash

To identify the device more fully:

Target_type_M/P (options for the second signature byte)

More details of the flash:

PageSZ	The page size
Pamask	The page address mask which equals FlashSZ – PageSZ.

The configuration bytes:

Fuses: extended, high and low and the lock byte

Programming the Config bytes

Keypress ‘p’ programs the flash with a hex file. It does not program the configuration or lock bytes but does read them out. For an unused device this gives the default values.

Key press ‘P’ programs the flash as above and also programs the configuration bytes

Setting up user programs

“UNO Programmer” programs the first of the target’s signature bytes at the top of its EEPROM (in all the top four locations should be considered as reserved).

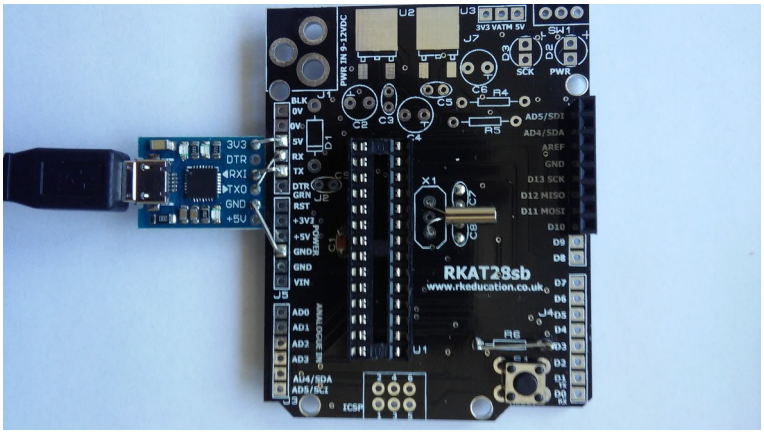
Code segment “set_device_type_and_memory_size” is provided for inclusion in user programs. It reads the signature byte and hence reads off the flash and eeprom sizes.

In the “Text_reader” application code segment “Setup_HW” calls “set_device_type_and_memory_size”

In its turn it calls “Set_device_signatures”. Full details of these code segments can be found in “Text_reader.h”. It is anticipate that these segments would be copied into user applications.

Note:
This process simplifies porting programs between one Atmega device and another.
Thanks to the Atmega architecture it is not necessary to know the EEPROM size to read the signature byte.

Test jig for programmed devices



Consists of an RKAT28sb Arduino shield
plus
CP2102 micro USB bridge.

Both available on ebay.

18pF trimmer capacitors are used with the watch crystal.

A device having been programmed on the UNO programmer HW was transferred to the test jig where it functioned satisfactorily. A crystal was also added to confirm that the user calibration byte obtained using breadboard was compatible with a double sided pcb.

NOTE: if reloading the UNO_programmer the target device Rx/Tx ports will usually have to be temporarily disconnected.

Devices with which the UNO_programmer has been tested

Atmega 328/P
Atmega 168/P