master | Go to file | Code

MCUdude Add PICkit4 and MPLAB SNAP programmers ... ✓ 3 days ago 248

avr	Add PICkit4 and MPLAB SNAP programmers	3 days ago
.gitignore	Update .gitignore	5 years ago
.travis.yml	Force installation of latest arduino:avr platform version during ...	3 months ago
LICENSE	Update license to LGPL v2.1	4 years ago
PlatformIO.md	Update PlatformIO.md	2 months ago
README.md	Add EEPROM retain menu option	4 months ago
Wiring_reference.md	Add wiring reference	2 years ago

README.md

MiniCore

build passing | support forum

An Arduino core for the ATmega328, ATmega168, ATmega88, ATmega48 and ATmega8, all running a custom version of Optiboot for increased functionality. This core requires at least Arduino IDE v1.6.2, where v1.8.5+ is recommended.
This core gives you two extra IO pins if you're using the internal oscillator! PB6 and PB7 is mapped to Arduino pin 20 and 21.
If you're into "generic" AVR programming, I'm happy to tell you that all relevant keywords are being highlighted by the IDE through a separate keywords file. Make sure to test the example files (File > Examples > AVR C code examples). Try writing a register name, *DDRB* for instance, and see for yourself!

Table of contents

Supported microcontrollers:

About

Arduino hardware package for ATmega8, ATmega48, ATmega88, ATmega168, ATmega328 and ATmega328PB

#arduino #avr #atmel #microcontroller #atmega168 #atmega88 #atmega48 #atmega8 #atmega328 #atmega328pb

Readme

View license

Releases 15

MiniCore v2.0.9 Latest
29 days ago

+ 14 releases

Packages

No packages published

Contributors 7

Languages

C++ 37.8% C 28.9% Shell 25.9% Makefile 6.7% Other 0.7%

- ATmega8
- ATmega48
- ATmega88
- ATmega168
- ATmega328

* All variants (A, P, PA and PB)

Can't decide what microcontroller to choose? Have a look at the specification table below:

	ATmega328	ATmega168	ATmega88	ATmega48	ATmega8
Flash	32kB	16kB	8kB	4kB	8kB
RAM	2kB	1kB	1kB	512B	1kB
EEPROM	1kB	512B	512B	256B	512B
PWM pins	6/9*	6	6	6	3

* ATmega328PB has 9 PWM pins

Supported clock frequencies

MiniCore supports a variety of different clock frequencies. Select the microcontroller in the boards menu, then select the clock frequency. You'll have to hit "Burn bootloader" in order to set the correct fuses and upload the correct bootloader.
Make sure you connect an ISP programmer, and select the correct one in the "Programmers" menu. For time critical operations an external crystal/oscillator is recommended.

You might experience upload issues when using the internal oscillator. It's factory calibrated but may be a little "off" depending on the calibration, ambient temperature and operating voltage. If uploading failes while using the 8 MHz internal oscillator you have these options:

- Edit the baudrate line in the boards.txt file, and choose either 115200, 57600, 38400 or 19200 baud.
- Upload the code using a programmer (USBasp, USBtinyISP etc.) or skip the bootloader by holding down the shift key while clicking the "Upload" button
- Use the 4, 2 or 1 MHz option instead

Frequency	Oscillator type	Comment
16 MHz	External crystal/oscillator	Default clock on most AVR based Arduino boards and MiniCore
20 MHz	External crystal/oscillator	
18.4320 MHz	External crystal/oscillator	Great clock for UART communication with no error
14.7456 MHz	External crystal/oscillator	Great clock for UART communication with no error
12 MHz	External crystal/oscillator	Useful when working with USB 1.1 (12 Mbit/s)
11.0592 MHz	External crystal/oscillator	Great clock for UART communication with no error
8 MHz	External crystal/oscillator	Common clock when working with 3.3V
7.3728 MHz	External crystal/oscillator	Great clock for UART communication with no error
4 MHz	External crystal/oscillator	
3.6864 MHz	External crystal/oscillator	Great clock for UART communication with no error

Frequency	Oscillator type	Comment
2 MHz	External crystal/oscillator	
1.8432 MHz	External crystal/oscillator	Great clock for UART communication with no error
1 MHz	External crystal/oscillator	
8 MHz	Internal oscillator	Might cause UART upload issues. See comment above this table
4 MHz	Internal oscillator	Derived from the 8 MHz internal oscillator
2 MHz	Internal oscillator	Derived from the 8 MHz internal oscillator
1 MHz	Internal oscillator	Derived from the 8 MHz internal oscillator

Bootloader option

MiniCore lets you select which serial port you want to use for uploading. UART0 is the default port for all targets, but ATmega328PB can also use UART1. If your application doesn't need or require a bootloader for uploading code you can also choose to disable this by selecting *No bootloader*. This frees 512 bytes of flash memory.

Note that you have need to connect a programmer and hit **Burn bootloader** if you want to change any of the *Upload port settings*.

BOD option

Brown out detection, or BOD for short lets the microcontroller sense the input voltage and shut down if the voltage goes below the brown out setting. To change the BOD settings you'll have to connect an ISP programmer and hit "Burn bootloader". Below is a table that shows the available BOD options:

ATmega328	ATmega168	ATmega88	ATmega48	ATmega8
4.3V	4.3V	4.3V	4.3V	4.0V
2.7V	2.7V	2.7V	2.7V	2.7V
1.8V	1.8V	1.8V	1.8V	-
Disabled	Disabled	Disabled	Disabled	Disabled

EEPROM option

If you want the EEPROM to be erased every time you burn the bootloader or upload using a programmer, you can turn off this option. You'll have to connect an ISP programmer and hit "Burn bootloader" to enable or disable EEPROM retain. Note that when uploading using a bootloader, the EEPROM will always be retained.

Link time optimization / LTO

After Arduino IDE 1.6.11 where released, There have been support for link time optimization or LTO for short. The LTO optimizes the code at link time, making the code (often) significantly smaller without making it "slower". In Arduino IDE 1.6.11 and newer LTO is enabled by default. I've chosen to disable this by default to make sure the core keep its backwards compatibility. Enabling LTO in IDE 1.6.10 or older will return an error. I encourage you to try the new LTO option and see how much smaller your code gets! Note that you don't need to hit "Burn Bootloader" in order to enable LTO. Simply enable it in the "Tools" menu, and your code is ready for compilation. If you want to read more about LTO and GCC flags in general, head over to the GNU GCC website!

Printf support

Unlike the official Arduino cores, MiniCore has printf support out of the box. If you're not familiar with printf you should probably read this first. It's added to the Print class and will work with all libraries that inherit Print. Printf is a standard C function that lets you format text much easier than using Arduino's built-in print and println. Note that this implementation of printf will NOT print floats or doubles. This is a limitation of the avr-libc printf implementation on AVR microcontrollers, and nothing I can easily fix.

If you're using a serial port, simply use `Serial.printf("Milliseconds since start: %ld\n", millis());`. You can also use the `F()` macro if you need to store the string in flash. Other libraries that inherit the Print class (and thus supports printf) are the LiquidCrystal LCD library and the U8G2 graphical LCD library.

Pin macros

Note that you don't have to use the digital pin numbers to refer to the pins. You can also use some predefined macros that maps "Arduino pins" to the port and port number:

```
// Use PIN_PB5 macro to refer to pin PB5 (Arduino pin 13)
digitalWrite(PIN_PB5, HIGH);

// Results in the exact same compiled code
digitalWrite(13, HIGH);
```

Programmers

MiniCore does not adds its own copies of all the standard programmers to the "Programmer" menu. Just select one of the stock programmers in the "Programmers" menu, and you're ready to "Burn Bootloader" or "Upload Using Programmer".

Select your microcontroller in the boards menu, then select the clock frequency. You'll have to hit "Burn bootloader" in order to set the correct fuses and upload the correct bootloader.
Make sure you connect an ISP programmer, and select the correct one in the "Programmers" menu. For time critical operations an external oscillator is recommended.

Write to own flash

MiniCore implements @majekw fork of Optiboot, which enables flash writing functionality within the running application. This means that content from e.g. a sensor can be stored in the flash memory directly, without the need of external memory. Flash memory is much faster than EEPROM, and can handle about 10 000 write cycles.
To enable this feature your original bootloader needs to be replaced by the new one. Simply hit "Burn Bootloader", and it's done!
Check out the Optiboot flasher example for more info about how this feature works, and how you can try it on your MiniCore compatible microcontroller.

How to install

Boards Manager Installation

This installation method requires Arduino IDE version 1.6.4 or greater.

- Open the Arduino IDE.
- Open the **File** > **Preferences** menu item.
- Enter the following URL in **Additional Boards Manager URLs**:

  ```
  https://mcudude.github.io/MiniCore/package_MCUdude_MiniCore_index.json
  ```

- Open the **Tools** > **Board** > **Boards Manager...** menu item.
- Wait for the platform indexes to finish downloading.
- Scroll down until you see the **MiniCore** entry and click on it.
- Click **Install**.

- After installation is complete close the **Boards Manager** window.
- **Note**: If you plan to use the *PB series, you need the latest version of the Arduino toolchain. This toolchain is available through IDE 1.8.6 or newer. Here's how you install/enable the toolchain:
    - Open the **Tools** > **Board** > **Boards Manager...** menu item.
    - Wait for the platform indexes to finish downloading.
    - The top is named **Arduino AVR boards**. Click on this item.
    - Make sure the latest version is installed and selected
    - Close the **Boards Manager** window.

Manual Installation

Click on the "Download ZIP" button in the upper right corner. Exctract the ZIP file, and move the extracted folder to the location "**~/Documents/Arduino/hardware**". Create the "hardware" folder if it doesn't exist. Open Arduino IDE, and a new category in the boards menu called "MiniCore" will show up.

PlatformIO

PlatformIO is an open source ecosystem for IoT development and supports MiniCore.

**See PlatformIO.md for more information.*

Getting started with MiniCore

Ok, so you're downloaded and installed MiniCore, but how to get started? Here's a quick guide:

- Hook up your microcontroller as shown in the pinout diagram, or simply just plut it into an Arduino UNO board.
    - (If you're not planning to use the bootloader (uploading code using a USB to serial adapter), the FTDI header and the 100 nF capacitor on the reset pin can be omitted.)
- Open the **Tools** > **Board** menu item, and select a MiniCore compatible microcontroller.
- If the *BOD option* is presented, you can select at what voltage the microcontroller will shut down at. Read more about BOD here.
- Select your prefered clock frequency. **16 MHz** is standard on most Arduino boards, including the Arduino UNO.
- Select what kind of programmer you're using under the **Programmers** menu.
- If the *Variants* option is presented, you'll have to specify what version of the microcontroller you're using. E.g the ATmega328 and the ATmega328P got different device signatures, so selecting the wrong one will result in an error.
- Hit **Burn Bootloader**. If an LED is connected to pin PB5 (Arduino pin 13), it should flash twice every second.
- Now that the correct fuse settings is sat and the bootloader burnt, you can upload your code in two ways:
    - Disconnect your programmer tool, and connect a USB to serial adapter to the microcontroller, like shown in the minimal setup circuit. Then select the correct serial port under the **Tools** menu, and click the **Upload** button. If you're getting some kind of timeout error, it means your RX and TX pins are swapped, or your auto reset circuity isn't working properly (the 100 nF capacitor on the reset line).
    - Keep your programmer connected, and hold down the `shift` button while clicking **Upload**. This will erase the bootloader and upload your code using the programmer tool.

Your code should now be running on your microcontroller! If you experience any issues related to bootloader burning or serial uploading, please use *this forum post* or create an issue on Github.

Wiring reference

To extend this core's functionality a bit futher, I've added a few missing Wiring functions. As many of you know Arduino is based on Wiring, but that doesn't mean the Wiring development isn't active. These functions are used as "regular" Arduino functions, and there's no need to include an external library.
I hope you find this useful, because they really are!

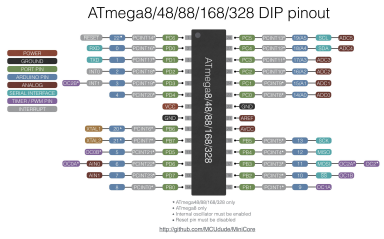
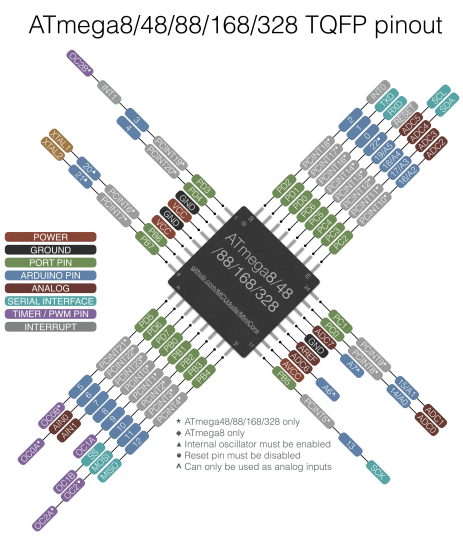
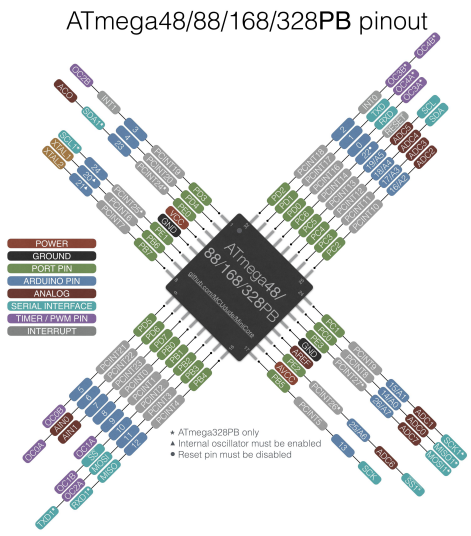
Function list

- portMode()
- portRead()
- portWrite()
- sleepMode()
- sleep()
- noSleep()
- enablePower()
- disablePower()

For further information please view the Wiring reference page!

Pinout

This core uses the standard Arduino UNO pinout and will not break compatibility of any existing code or libraries. What's different about this pinout compared to the original one is that this got three aditinal IO pins available. You can use digital pin 20 and 21 (PB6 and PB7) as regular IO pins if you're ussing the internal oscillator instead of an external crystal. If you're willing to disable the reset pin (can be enabled using high voltage parallel programming) it can be used as a regular IO pin, and is assigned to digital pin 22 (PC6). **Click to enlarge:**

DIP-28 package ***ATmega8/48/88/168/328***	TQFP-32 SMD package ***ATmega8/48/88/168/328***	TQFP-32 SMD package ***ATmega48/88/168/328PB***
		

Minimal setup

Here is a simple schematic showing a minimal setup using an external crystal. Skip the crystal and the two 22pF capacitors if you're using the internal oscillator. If you don't want to mess with breadboards, components and wiring; simply use your Arduino UNO! **Click to enlarge:**

DIP-28 package ***ATmega8/48/88/168/328***	TQFP-32 SMD package ***ATmega8/48/88/168/328***	TQFP-32 SMD package ***ATmega48/88/168/328PB***