Click to verify

Hi, I am new to Arduino and am planning to read a serial data UART from a BMS and show the data on a IPS screen. With raspberry pi, it took over 30s to just boot which is not ideal. I want to try Arduino and just saw you could set arduino as a target and just install the program and there is no delay after reboot. Examples I saw included Uno as programmer and nano as target but I wanted to know if 2 arduino nano (every) could be used so one of them can have the executable and boots instantly. If so, please let me know or direct me to any tutorials. I couldn't find this in my search. I am asking because I have 2 nanos and getting a UNO might take a long time as they aren't available in my country. Thanks. maharzan: I have 2 nanos It sounds like You can get a nano started and thereby be programmed. Why not use that approach for both? Why program one nano to be a programmer for the second one? I saw a video where it said there is no delay without bootloader. I am still researching on how it works. Eventually, I want to put the arduino in a car connected to a BMS and a screen and once the key is on, it powers the arduino and shows me the battery status. I did it with Rasberry Pi but it takes forever to load its OS, even without OS (Terminal only) it takes 15 secs. I want to do that instantly. Is that doable? Hi @maharzan. maharzan: Examples I saw included Uno as programmer and nano as target I'm sure you are referring to turning the Uno into a "Arduino as ISP" programmer, as shown here: docs.arduino.cc Arduino ISP turns your Arduino into an in-circuit programmer to re-program AtMega chips. Two important points: Nano Every as Target The Nano Every can not be programmed using an "Arduino as ISP" programmer. The reason is that, unlike the ATmega328P microcontroller on the classic Nano, the ATmega4809 primary microcontroller on the Nano does not support that ISP programming interface. Instead it uses UPDI. So you need a UPDI programmer to program the Nano Every. The ATSAMD11 microcontroller used as the USB interface on the Nano Every board actually programs the ATmega4809 via UPDI. You should also note that the UPDI interface is not available on the pin headers of the Nano Every as is the case with the ISP interface on the classic Nano. You can access the UPDI interface via the test pads on the bottom of the Nano Every board on the end opposite the USB jack: Nano Every as "Arduino as ISP" Programmer Even though in theory any Arduino board should be usable as an "Arduino as ISP" programmer, my experiments show that it does not work when the Nano Every board is used as an "Arduino as ISP" programmer. Thank you for the detailed explanation. I am new to this so might take sometime to understand. So, is it possible to load a program on the nano from say mac interface and it will run standalone on boot? maharzan: I want to put the arduino in a car connected to a BMS and a screen and once the key is on, it powers the arduino and shows me the battery status. I did it with Rasberry Pi but it takes forever to load its OS, even without OS (Terminal only) it takes 15 secs. I want to do that instantly. Is that doable? Yes, the Nano Every should start up very quickly with power on. There is no need to use a separate UPDI programmer with the Every. The Nano Every can be programmed over the USB connection using the Arduino IDE with either the Arduino megaAVR Core or alternatively with MCUdude MegaCoreX. There is a 1200 baud trigger signal which which puts the SAMD11 into UPDI programmer mode and the program is loaded onto the AT4809 processor. The Nano Every also has multiple hardware serial ports independent from the USB port which sound like they will be useful for your application. Great. I will look into it. will start testing...just need to find that micro USB cable.. Thanks. This topic was automatically closed 180 days after the last reply. New replies are no longer allowed. If you are using an Arduino Nano or Arduino UNO R3 in a project and are reaching the limits of the SRAM or flash, the Arduino Nano Every Board could be a suitable alternative. If you limit yourself to the functions of the “Arduino language” and do not address any registers directly, you can transfer your sketches without any problems. On the surface, therefore, the differences are small. However, if you look deeper, you will see that the underlying ATmega4809 (data sheet ATmega4808/4809) as a member of the megaAVR® series differs significantly from the ATmega328P. And it is precisely this look “under the hood” that this article is about. In addition to the original Arduino Nano Every, I will also look at an interesting brother from China, which is based on the ATmega4808 and is known as the “Nano Thinary” or “Nano 4808”. The ATmega4808 is very similar to the ATmega4809. For this reason, the manufacturer Microchip has described both microcontrollers together in a data sheet. And it also makes sense to cover both boards in one article. What topics I will cover: The available options of the boards and their limitations depend on three factors: the underlying microcontroller, the board architecture (especially which MCU pins are available as board pins) the board package used. Here are some key data for the Arduino Nano Every board: Max. 20 MHz clock frequency (depending on the board package) Operating voltage: 5 Volt Max. Current per I/O pin: 20 milliamps Flash: 48 KB SRAM: 6 KB EEPROM: 256 byte Interfaces: 1x SPI, 1x I2C, max. 4x USART (depending on the board package) I/O pins: 22 (+2) I’ll come to the “+2” later. PWM pins: 5 External interrupts on all I/O pins event system Configurable Customer Logic 1x 16-bit Timer A, 4x 16-bit Timer B The same applies to the “Nano 4808” (Nano Thinary), with the following exceptions: max. 3x USART PWM pins: 8 I/O pins: 25 (+1) 1x 16-bit Timer A, 3x 16-bit Timer B You can get the Arduino Nano Every from the Arduino Store. I paid €12.50 plus tax and shipping, which is relatively cheap for the original boards. For comparison: The Arduino Nano costs €20.50 in the Arduino Store (as of 10/23). I found the “Nano 4808” for about €7.50 on AliExpress. But allow a few weeks for delivery, at least if you live in Europe. The Arduino Nano Every uses the 48-pin version of the ATmega4809, the “Nano 4808” is based on the 32-pin version of the ATmega4808: 32-pin ATmega4808 and 48-pin ATmega4809 Both microcontrollers belong to the megaAVR®0 series, which differs fundamentally from the “traditional” AVR® series. For example, programming is done via UPDI (Unified Program Debug Interface) and not as usual via ISP or using the Optiboot bootloader via the UART interface. This is why the boards discussed here do not have a USB-to-serial adapter, but a UPDI programmer. UPDI is a 1-wire interface that you may recognize from my article about the megaTinyCore package from Spence Konde. Pinout Arduino Nano Every As you can see, the pins of the ATmega4809 are only partially available on the Arduino Nano Every Board. The rest is lost. In some cases, however, you can use PORTMUX to reassign the functions of non-available pins to existing pins. The pin designations and pin numbers (light blue / pink) essentially correspond to those of the ATmega328P-based Arduino Nano. However, there are some differences: A6 and A7 can be used as I/O pins. A4 and A9 have two pin numbers, namely 18/22 and 19/23. AREF’s pin number is 39, but only when using the MegaCoreX package (therefore not mentioned in the pinout). The port pins Pxy on which the Arduino pins are based are entirely different from those of the Arduino Nano or Arduino UNO. Sketches that address the ports directly must be rewritten if you transfer them to the Arduino Nano Every. We will come to the other functions, such as TCA0-x or EVENTy, in the course of this article. Pinout “Nano 4808” (Thinary) The pinout diagram of the “Nano 4808” differs from the Arduino Nano Every as follows: Pins 22 and 23 are separate, pin 24 is added. The PWM pins are assigned differently. A UPDI pin is available. You can therefore use a UPDI-capable programmer such as the Atmel ICE instead of the existing on-board programmer, provided the board package has implemented the programmer. The port pins on which the Arduino pins are based are different again. I will not explain here in detail how to install board packages. Instructions can be found here, for example. The easiest way is to install the “Arduino megaAVR Boards” package, as you do not need to enter an additional board manager URL in the settings. After installation, select the Arduino Nano Every as the board and set the option “None (ATMEGA4809)” under Tools → “Registers emulation”. As an alternative, I recommend the MegaCoreX package, as it contains libraries for many of the ATmega4809 functions that the standard package does not have. This is particularly useful for those who do not want to deal with registers. You can apply many more settings with this package, such as conveniently converting the reset pin into a normal I/O pin or changing the clock rate. The documentation on GitHub is exemplary. You can find installation instructions here. Don’t worry, it’s done in just a few minutes. After installation, make the following settings in the Tools menu: Board: MegaCoreX → ATmega4809 Pinout: Nano 4809 Programmer: JTAG2UPDI The rest is best left unchanged for now. I tried two options for the “Nano 4808”. One of them was again the MegaCoreX package, but I also tried the ThinaryArduino package. Do yourself a favor and take the MegaCoreX package! I do not recommend the ThinaryArduino package, as it is apparently not really maintained. It works in principle, but has some special features: Serial must be replaced by Serial1. analogWrite() is only available on pins A6 and A7. The documentation leaves a lot to be desired. You can use the AREF pin as a GPIO without any further action. The only limitation: It has a higher capacitance than other pins and is therefore somewhat less responsive. The MegaCoreX package has assigned its own number for the pin, namely 39 for the Arduino Nano Every and 25 for the “Nano 4808”. When using the megaAVR package, you could use it directly via port manipulation (PD7). Since the ATmega4808/4809 is flashed via UPDI, you do not need the reset pin to upload the sketches. The MegaCoreX allows you to set the pin as a GPIO via the Tools → Reset menu. But be careful: If you configure the pin as OUTPUT and press the reset button, you will create a short circuit! With the megaAVR package, you can apply this change in the boards.txt file. On my computer it is under: “C:\Users\Ewald\AppData\Local\Arduino15\packages\arduino\hardware\megaavr\1.8.8”. Replace the following line in the file: nona4809.bootloader.SYSCFG0=0xC9 by: nona4809.bootloader.SYSCFG0=0xC1. This clears the RSTPINCFG bit (no. 3) in the SYSCFG0 register as part of the next upload. As the register can only be written to via UPDI, you have to take the detour via boards.txt. But only go for it if you know exactly what you’re doing! If you limit yourself to the Arduino functions, your sketches should work as usual. It all “feels like Arduino Nano”. This could be the end of this article. But there is still the exciting world of the megaAVR0 series to discover beneath the Arduino surface. Nevertheless, a little familiarization is required, even if you should already have experience with register programming of AVR microcontrollers. Here is a simple blink sketch as a taster (Arduino Nano Every): void setup() { PORTE.DIRSET = PIN2_bm; // set PE2 (D13, LED_BUILTIN) to OUTPUT } void loop() { PORTE.OUTSET = PIN2_bm; // set PE2 to HIGH delay(200); PORTE.OUTCLR = PIN2_bm; // set PE2 to LOW delay(200); } For the “Nano 4808”, replace PORTE with PORTC to make the board LED flash. A statement like PORTE |= (1 TCB_CLKSEL_gp)); It may all sound a little complicated at first, but you soon get used to this type of notation. It involves a certain amount of writing, but it pays off because the code is still easy to understand even after a few months. If you want to see how the structures for the modules and registers are set up, then take a look at “C:\Program Files (x86)\Arduino\hardware\tools\avr\avr\include\avr\iom4809.h” or “…iom4808.h”. This new way of programming registers takes some getting used to at first. The ATmega4808/4809 data sheet will help you find your way around. It is organized according to the modules and for each module there is a “Register Summary” with links to the individual registers. PORTx Register Summary As demonstrated by the register summary above, there are a number of registers whose names begin with “DIR” or “OUT”. The “DIR …” registers control which pins are used as INPUT or OUTPUT. With the help of the “OUT …” registers you set the pin level, i.e. HIGH or LOW. The second part of the name means: SET: sets bits in OUT and DIR → OUTPUT or HIGH CLR: clears bits in OUT and DIR → INPUT or LOW TGL: toggles the status. You can also write directly to the DIR and OUT registers, but the beauty of OUTSET, OUTCLR, OUTTGL and their “DIR” counterparts is that the instructions are pin-selective. Example: PORTx.OUT = PINy_bm; sets pin y of port x to HIGH and all other pins of port x to LOW. PORTx.OUTSET = PINy_bm; only acts on pin y, i.e. the statement corresponds to PORTx.OUT |= PINy_bm;. You can read the status of the pins from the IN registers (digitalRead(), so to speak). However, if you write a bit to the PORTx.IN register, the corresponding bit is toggled in PORTx.OUT. The IN register is the counterpart to the PINx register of traditional AVR microcontrollers. Each pin has its own control register, namely PINxCTRL: PORTn.PINxCTRL Register The INVEN bit allows you to invert the INPUT and OUTPUT values. PULLUPEN activates the internal pull-up resistor. With ISC you can define the interrupt conditions or switch off the digital function of the pin completely (INPUT_DISABLE): ISC[2:0] Bit Group Configuration I’ll come back to the blink sketch and show that there are other ways to achieve the same effect: void setup() { PORTE.DIRSET = PIN2_bm; // set PE2 (D13) to OUTPUT // PORTE.DIR |= (1