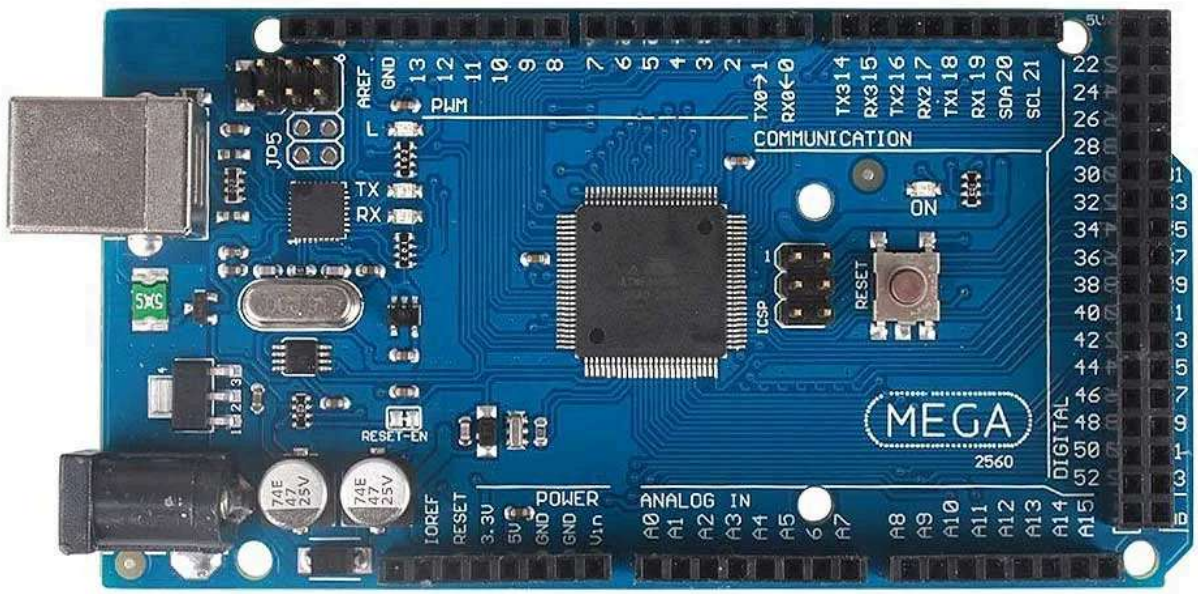


AVR Programming

The Arduino Mega 2560 uses a microcontroller known as the ATMega 2560. The ATMega is built on the AVR architecture.

Arduino provides a single header file <Arduino.h> which gives you access to the GPIO, timers, serial communication and more. There are also many libraries available, which makes Arduino a good choice for rapid prototyping.

However, relying only on the Arduino header file can put limitations on your program. The library functions hide the low-level configurations of peripherals, which may prevent you from using some device features. In particular, it can difficult to manage precise timing of functions inside your program

There is an open source library called AVR libc [1] that can make programming the ATMega much easier. Importantly for us, it allows you access the hardware registers of the microcontroller.

Luckily, <Arduino.h> gives us access to some of the AVR libc headers!

This guide will show you how to use AVR libc on the Arduino Mega 2560 to set timer interrupts for coordinating the different tasks in your program.

The microcontroller contains an AVR CPU, which is connected to other components such as memory and xperipherals such as timers, and controllers for serial communication. The peripherals contain their own registers, which are memory mapped to the CPU through some form of on-chip bus.

Because the peripheral registers are all memory mapped, we can read and modify these registers within our code. These registers are used for reading and writing data, configuring the peripheral, or to check status flags.

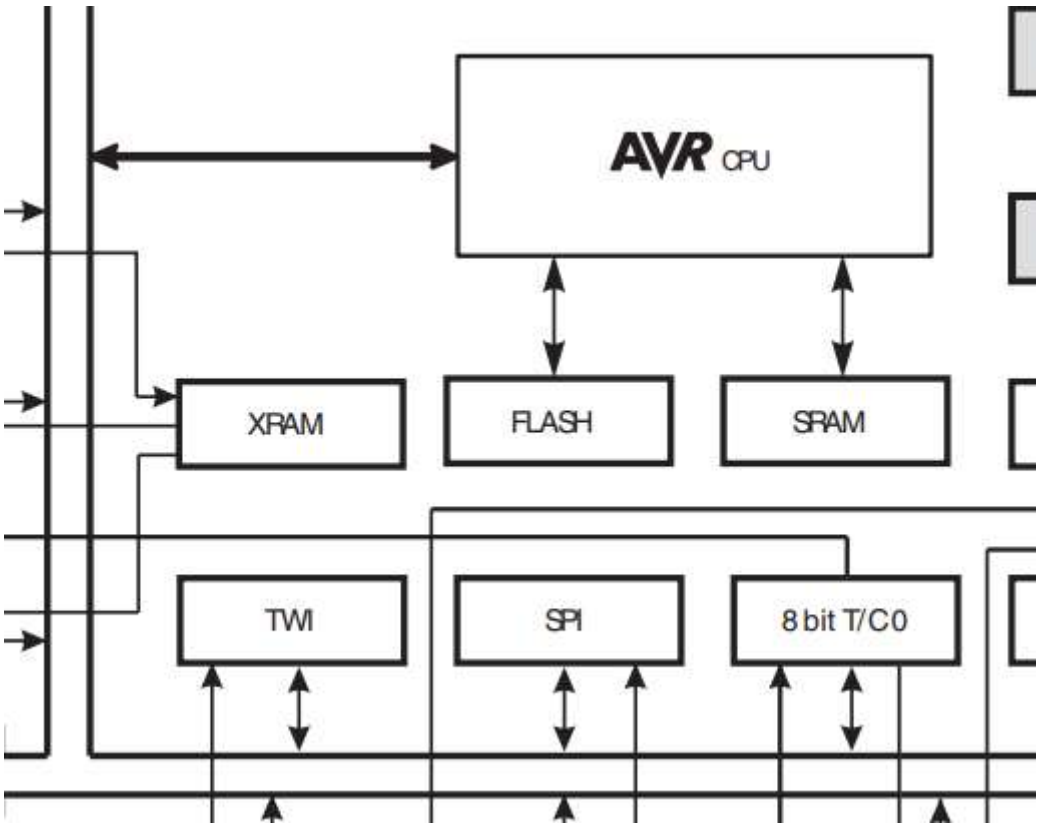


Figure 1: Block diagram of the ATmega 2560 [2]

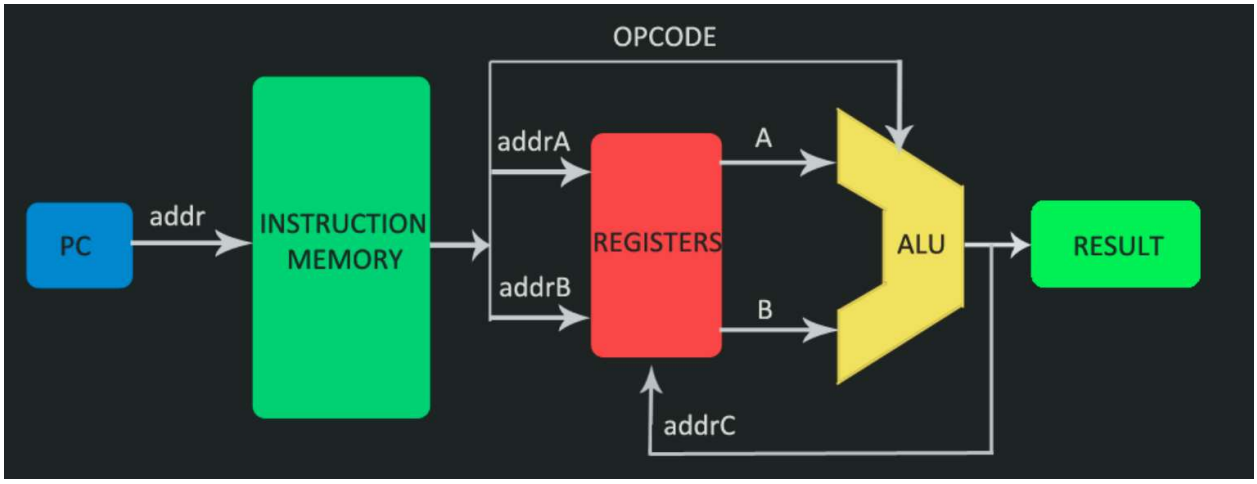
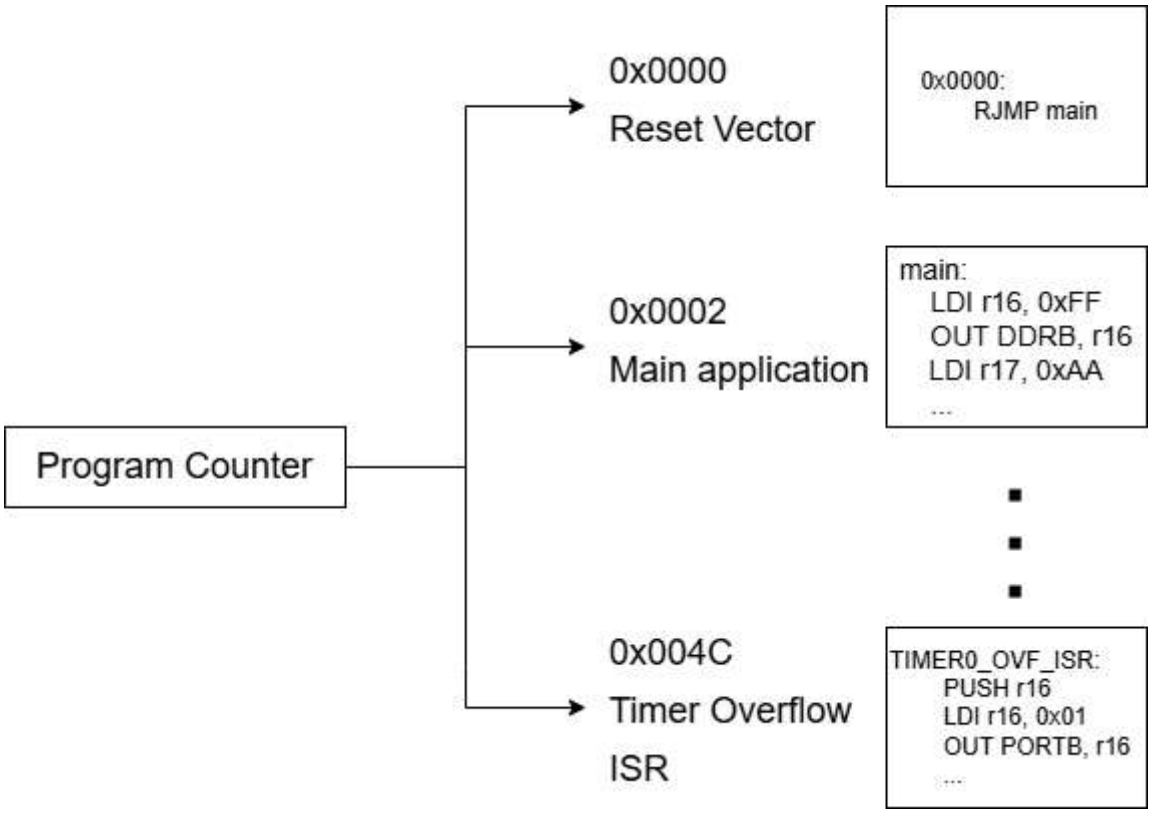


Figure 2: Simple CPU [3]

Inside the AVR CPU, there is a “program counter” which holds the address of the next instruction to be executed. After every instruction, the program counter either increments to the next instruction, or is loaded with some other address.



One reason the program counter might jump to another section code is for a CPU interrupt. Interrupts can be generated from a CPU system exception, or from one of the CPU’s peripherals.

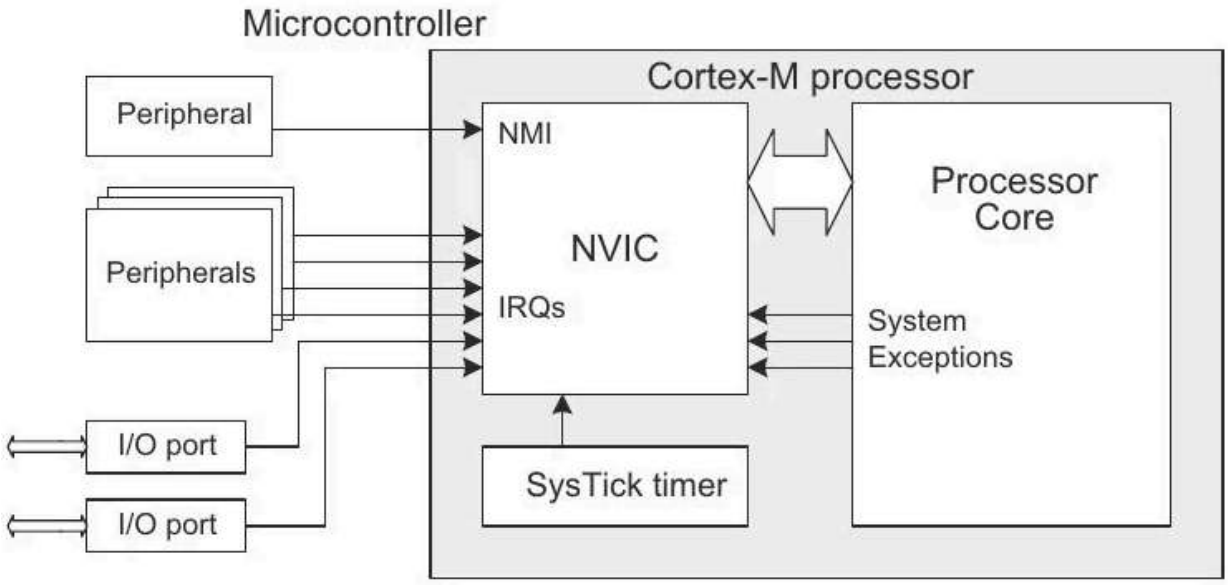


Figure 1: NVIC block diagram for ARM Cortex-M processor. This is from a different CPU than the AVR, but it shows a clear depiction of an interrupt controller.

When a CPU interrupt occurs, the correct subroutine is loaded by indexing a “vector table” using the interrupt “vector number” based on the interrupt source.

Table 14-1. Reset and Interrupt Vectors

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	PCINT1	Pin Change Interrupt Request 1
12	$0016[3]	PCINT2	Pin Change Interrupt Request 2
13	$0018	WDT	Watchdog Time-out Interrupt
14	$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	$0028	TIMER1 OVF	Timer/Counter1 Overflow

Interrupt vectable table for the ATmega 2560

Timer Interrupts

We want to use the timers to set a repeating interrupt in order to generate events with a set frequency. The compare match and overflow interrupts could work for this.

First, let's define an interrupt handle. This function will run whenever the Timer/Counter1 Compare Match A interrupt is raised.

```
ISR(TIMER1_COMPA_vect) {
// process the interrupt here
}
```

The TIMER1_COMPA_vect macro is defined with the other interrupt vectors in <avr/io.h>.

Now let's take a look at the code needed for setting the timer:

```
init_timer() {
TCCR1A = 0; // set entire TCCR1A register to 0
TCCR1B = 0; // same for TCCR1B
TCNT1  = 0; //initialize counter value to 0
// set compare match register for 1hz increments
OCR1A = 15624; // = (16*10^6) / (1*1024) - 1 (must be <65536)
// turn on CTC mode
TCCR1B |= (1 << WGM12);
// Set CS10 and CS12 bits for 1024 prescaler
TCCR1B |= (1 << CS12) | (1 << CS10);
// enable timer compare interrupt
TIMSK1 |= (1 << OCIE1A);
}
```

What's going on here?

First, let's read the Output Compare section of the datasheet for Timer 1.

Section 17.7 - Output Compare Units on page 141 tells us that

"The 16-bit comparator continuously compares TCNTn with the Output Compare Register (OCRnx). If TCNT equals OCRnx the comparator signals a match. A match will set the Output Compare Flag (OCFnx) at the next timer clockcycle. If enabled (OCIEnx = 1), the Output Compare Flag generates an Output Compare interrupt"

Output Compare interrupt isn't quite the same as Timer/Counter1 Compare Match A. Let's check what the OCIE1A flag is.

In the TIMSK register description on page 162 we find the OCIEnA bit. This section tells us

"When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), theTimer/Counter Output Compare A Match interrupt is enabled."

We now know we need to set the Output Compare register to some value so that when the Timer/Count matches, an interrupt will be triggered at a certain frequency.

How fast is the timer counting? The counter will increase every cycle of the “clock source” which is the CPU clock divided by some prescaler.

In the Register description of TCCR1B at the top of page 157, this table shows us how the Clock Select bits are used for selecting the clock source of the timer.

Table 17-6. Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Page 157

The clock on the Arduino Mega 2560 is 16MHz, according to the Arduino Mega datasheet.

We can use these lines to select the 1024 prescaler.

```
TCCR1B |= (1 << CS12) | (1 << CS10);
```

Suppose we want a frequency of 1Hz. Since we selected the 1024 prescaler, our clock source is 15.625kHz. To achieve a frequency of 1Hz, we would want to trigger the compare interrupt every 15,625 clock cycles or 1 second. We can then set the value of OCR1A to 15625 - 1 because we are starting the count at 0.

```
OCR1A = 15624; //  (16*10^6) / (1*1024) - 1
```

Another way to think about this is to consider that when we wait for the counter to reach a certain value, we are dividing the clock by that value.

If we set the counter to 1, the interrupt would go off with a frequency of our clock source 15.625 kHz. If we set the counter to 2, the frequency would be halved to 7.8kHz. So when we set the Output Compare register to 15625, we are dividing our 15625Hz clock source by itself to get a frequency of 1Hz.

timer overflow frequency TOVCK, the maximum value (MaxVal) of the timer, the system clock (CK), and the division factor of the prescaler (PVal).

$$TOV_{CK} = \frac{f_{CK}}{MaxVal} = \frac{(P_{CKx} / PVal)}{MaxVal} = \frac{P_{CKx}}{(PVal \bullet MaxVal)}$$ [4]

We can identify the count needed for a target frequency with the following formula.

$$\frac{clk\ source}{count} = f$$

$$count = \frac{clk\ source}{f}$$

Remember that our clock source is 16MHz divided by our prescaler.

How do we know the TCNTn register resets every time?

In the register description for TCCR1A section on the top of page 155 is an explanation that the combined WGMnx bits from register TCCRxA and TCCRxB sets the mode of operation for the Timer/Counter. Table 17-2 shows the timer mode for each bit combination..

Table 17-2. Waveform Generation Mode Bit Description[(1)]

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP

Page 145

Further down page 145 in section 17.9.2 Clear Timer on Compare Match (CTC), it explains that CTC mode clears the timer counter when TCNT1 matches the value in OCR1A.

That is why we use the line

```
TCCR2A |= (1 << WGM21);
```

Power Management

We might want to reduce the power consumption of our microcontroller while we are waiting for the timer interrupts. We can read about that in section 11 - Power Management and Sleep Modes on page 50.

The table below shows us the different low power modes available on the ATMega 2560. The Idle mode has the least power savings, but will allow us to wake up from our timer interrupts. The Power-save mode will have much lower power consumption, but we can only wake up from Timer2 interrupts.

We also have to be careful about how quickly we wake up after going to sleep, as the overhead from entering sleep mode might outweigh the power reductions during sleep. If we only go to sleep for a very short time, we won't have a chance to save much power.

For our projects today, it might not be worth it at all to enter power-save or even idle mode. We would have to measure the current draw to be sure.

Table 11-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes.

	Active Clock Domains					Oscillators		Wake-up Sources						
Sleep Mode	**clk_{CPU}**	**clk_{FLASH}**	**clk_{IO}**	**clk_{ADC}**	**clk_{ASY}**	**Main Clock Source Enabled**	**Timer Osc Enabled**	**INT7:0 and Pin Change**	**TWI Address Match**	**Timer2**	**SPM/ EEPROM Ready**	**ADC**	**WDT Interrupt**	**Other I/O**
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X
ADCNRM				X	X	X	X(2)	X(3)	X	X(2)	X	X	X	
Power-down								X(3)	X				X	
Power-save					X		X(2)	X(3)	X	X			X	
Standby(1)						X		X(3)	X				X	
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X	

Page 50

The Sleep Mode Control Register SMCR description contains a table showing how to select the mode using the Sleep Mode Bits. The SMCR Sleep Enable bit is used to make the CPU enter sleep mode when the SLEEP instruction is executed.

Table 11-2. Sleep Mode Select

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby(1)
1	1	1	Extended Standby(1)

To set the Idle sleep mode, we can use

```
void init_idle()
{
 SMCR &= ~(1 << SM2) & ~(1 << SM1) & ~(1 << SM0);
 SMCR |= (1 << SE);
}
```

For power save mode

```
void init_power_save()
{
 SMCR |= (1 << SM1) | (1 << SM0);
 SMCR |= (1 << SE);
}
```

When we are ready to enter the sleep mode, we can use

```
void atmega_sleep()
{
 sleep_enable();
 sleep_cpu();
 sleep_disable();
}
```

Let's go into some examples of how to structure our main loop. Suppose we have three tasks or functions we want to run, possibly at different rates.

The first option is to simply run each function one after the other inside the loop. This is known as **superloop**. We might want to delay at the end of the loop, or put the microcontroller to sleep. This will run each task at the same frequency.
https://github.com/albertaloop/elegoo-demo/blob/main/superloop/src/main.cpp

```
loop() {
      task1();
      task2();
      task3();
      atmega_sleep();
}
```

Another approach we could take is to put a switch statement inside the loop to create a small state machine. The state variable is an integer that we increment at the end of each loop, and reset to zero once we move through all the states.

This allows us to run some tasks to run in integer multiple frequencies of each other. Consider the example below. Task 1 runs twice as often as tasks 2 and 3. Let's refer to this approach as **harmonic loop periods**.
https://github.com/albertaloop/elegoo-demo/blob/main/harmonic%20loop%20periods/src/main.cpp

```
#define TASK_STATE1 0
#define TASK_STATE2 1
#define TASK_STATE3 2
#define TASK4_STATE 3
#define NUM_STATES 4

void loop() {
  if(timer_ready)
  {
    switch(state)
    {
      case TASK_STATE1:
        task1();
      break;
      case TASK_STATE2:
        task2();
      break;
     case TASK_STATE3:
       task1();
```

```
      break;
      case TASK_STATE4:
        task3();
      break;
      default: break;
    }
    state++;

    if(state%(NUM_STATES) == 0) state = TASK_STATE1;
    timer_ready = false;
 }
}
```

In this final example each task will get its own timer interrupt. Instead of setting a boolean in the ISR, we will push a value to a circular buffer. The loop will check each buffer to see if a new value came in. We will call this approach **multiple interrupts**.
https://github.com/albertaloop/elegoo-demo/blob/main/multiple%20interrupts/src/main.cpp

The circular buffer

```
#define NUM_BITS    4
#define BUFFER_LEN  (1 << NUM_BITS)

struct circular_buffer_t
{
   int values[BUFFER_LEN];
   unsigned int write_ptr = 0;
   unsigned int read_ptr = 0;
} ;
typedef circular_buffer_t circular_buffer;

void push(int val, volatile circular_buffer *buf)
{
 if (buf->write_ptr >= buf->read_ptr && buf->write_ptr < BUFFER_LEN)
 {
   buf->values[buf->write_ptr] = val;
   buf->write_ptr &= BUFFER_LEN-1;
 }
}

int pop(volatile circular_buffer *buf)
{
 if (buf->read_ptr != buf->write_ptr)
 {
   int ret = buf->values[buf->read_ptr];
```

```
    buf->read_ptr &= BUFFER_LEN-1;
  }
  else
  {
    return -1;
  }
}
```

The loop

```
void loop()
{
 static int val;
 if ((val = pop(&timer1_buffer)) >= 0)
 {
     task1();
 }
 if ((val = pop(&timer3_buffer)) >= 0)
 {
     task2();
 }
 if ((val = pop(&timer5_buffer)) >= 0)
 {
     task3();
 }
 atmega_sleep();
}

ISR(TIMER1_COMPA_vect)
{
 push(1, &timer1_buffer);
}

ISR(TIMER3_COMPA_vect)
```

```
{
 push(1, &timer3_buffer);
}

ISR(TIMER5_COMPA_vect)
{
 push(1, &timer5_buffer);
}
```

References

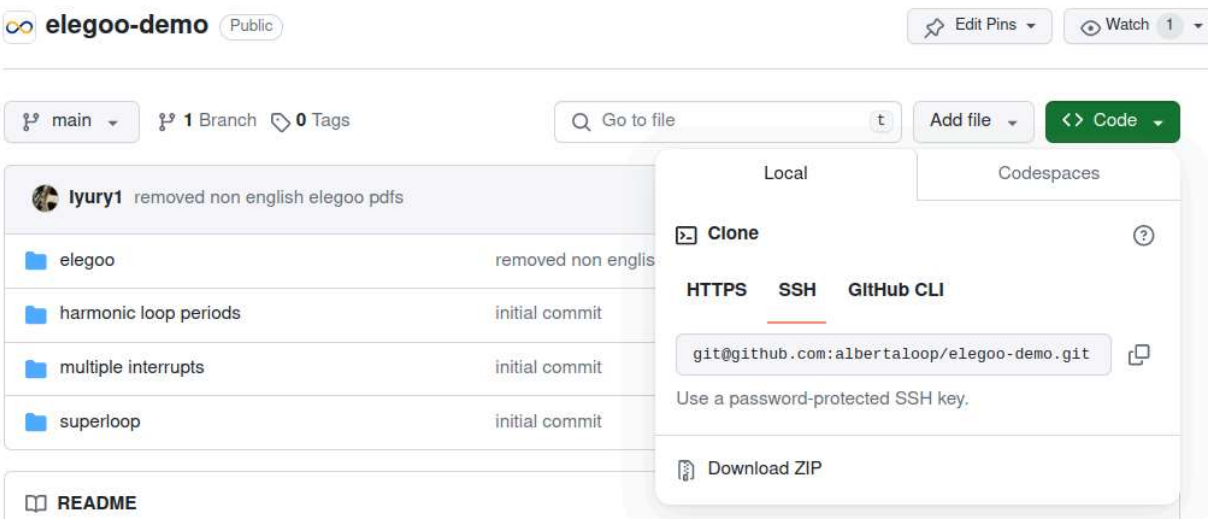
[1] https://github.com/avrdudes/avr-libc
[2]
https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

[3]
https://4.bp.blogspot.com/-kKR5cBljWhY/VSXMrZtgJ9I/AAAAAAAAAcE/dYyekb6uZPk/s1600/procc.png

[4] Atmel AVR130: Setup and Use of AVR Timers [APPLICATION NOTE]

Environment setup

To download the example projects, download the git repository
https://github.com/albertaloop/elegoo-demo



The Arduino Mega can be programmed using the Arduino IDE, the PlatformIO extension for VSCode, or from the command line using avr-dude. PlatformIO is recommended for this workshop.

1. **Arduino IDE**

https://www.arduino.cc/en/software

Under “Download Options”, select the download for your operating system.

Downloads

Arduino IDE 2.3.4

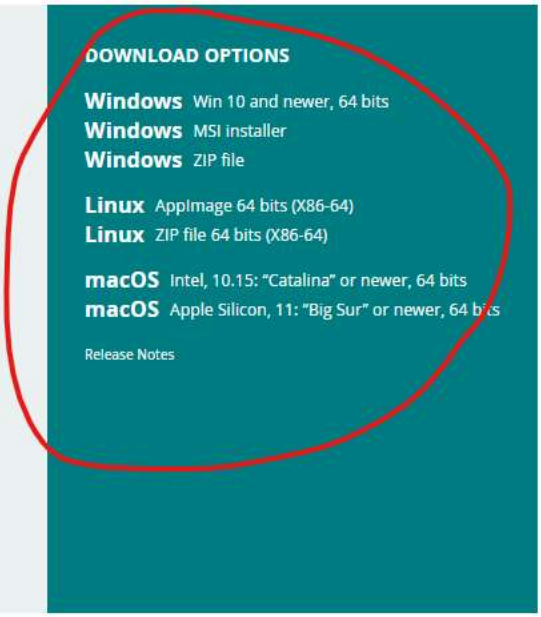
The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the **Arduino IDE 2.0 documentation**.

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on **GitHub**.



Nightly Builds

Download a **preview of the incoming release** with the most updated features and bugfixes.

Windows
macOS Version 10.15: "Catalina" or newer, 64 bits
Linux AppImage 64 bits (X86-64)
Linux ZIP file 64 bits (X86-64)
Changelog

Keep pressing “Just Download” until the download begins.

Linux:
The Zip download contains a directory with the arduino ide already installed. You can extract this directory wherever you wish.

I copied mine to /opt/, which is a place applications can be installed that are not tracked by the package manager by your distribution.

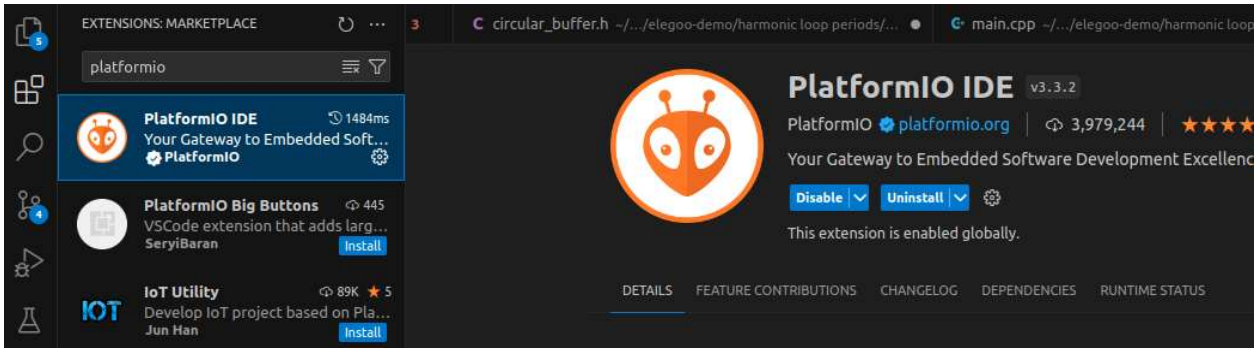
```
ian@ian-GL553VD:~/Downloads$ sudo mv arduino-ide_2.2.1_Linux_64bit /opt/
ian@ian-GL553VD:~/Downloads$ cd /opt/arduino-ide_2.2.1_Linux_64bit/
ian@ian-GL553VD:/opt/arduino-ide_2.2.1_Linux_64bit$ ls
arduino-ide                 libffmpeg.so                resources
```

```
chrome_100_percent.pak  libGLESv2.so           resources.pak
chrome_200_percent.pak  libvk_swiftshader.so   snapshot_blob.bin
chrome_crashpad_handler libvulkan.so.1         v8_context_snapshot.bin
chrome-sandbox          LICENSE.electron.txt   vk_swiftshader_icd.json
icudtl.dat              LICENSES.chromium.html
libEGL.so               locales
ian@ian-GL553VD:/opt/arduino-ide_2.2.1_Linux_64bit$ pwd
/opt/arduino-ide_2.2.1_Linux_64bit
ian@ian-GL553VD:/opt/arduino-ide_2.2.1_Linux_64bit$ gedit ~/.bashrc
```

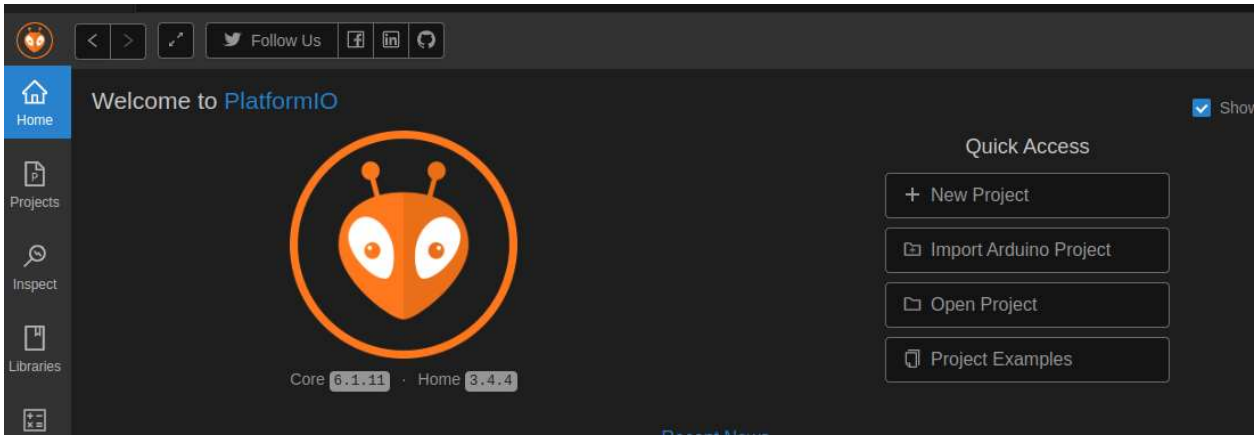
2. Platform IO on VSCode

Download VSCode from their downloads page:
https://code.visualstudio.com/download

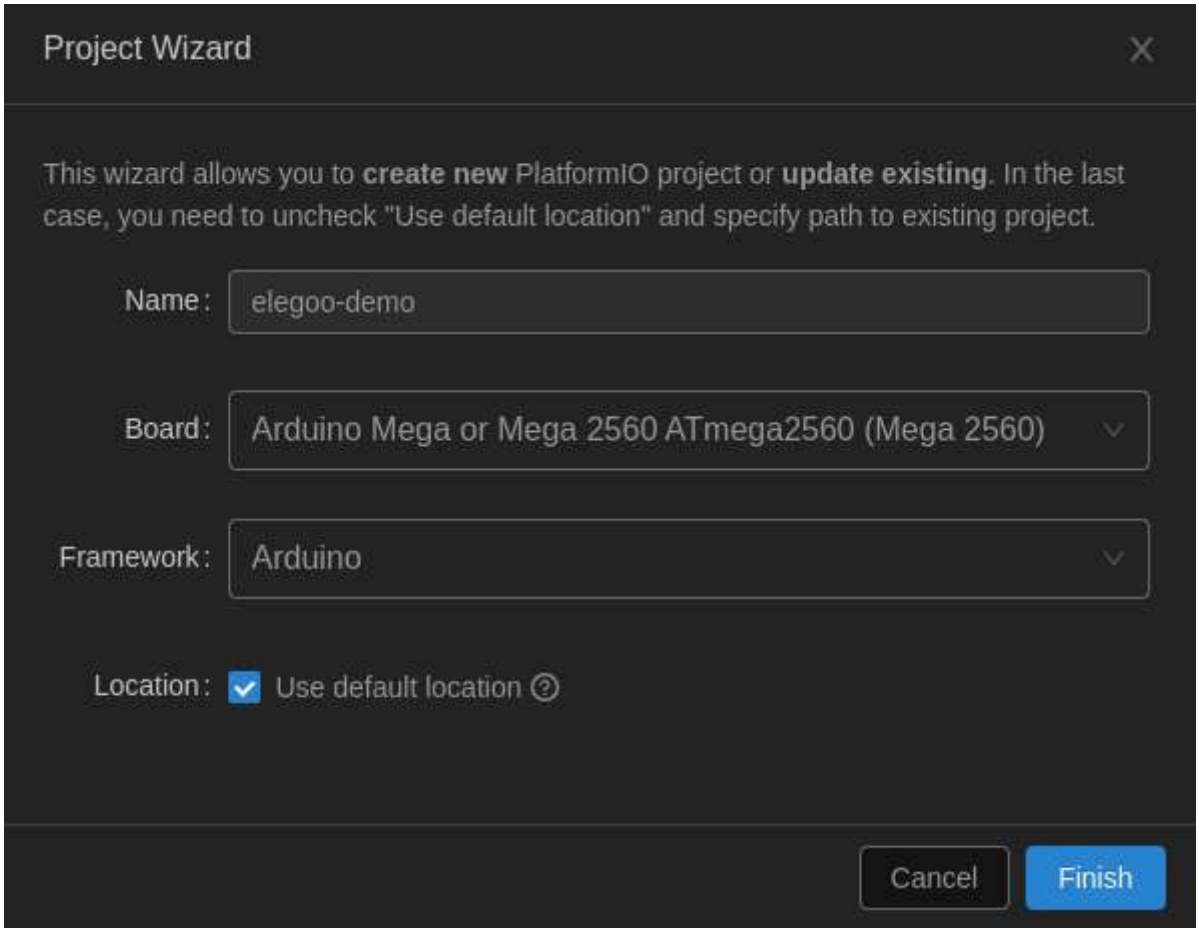
Open VSCode once you have installed it, and search for platformio in the extensions tab.



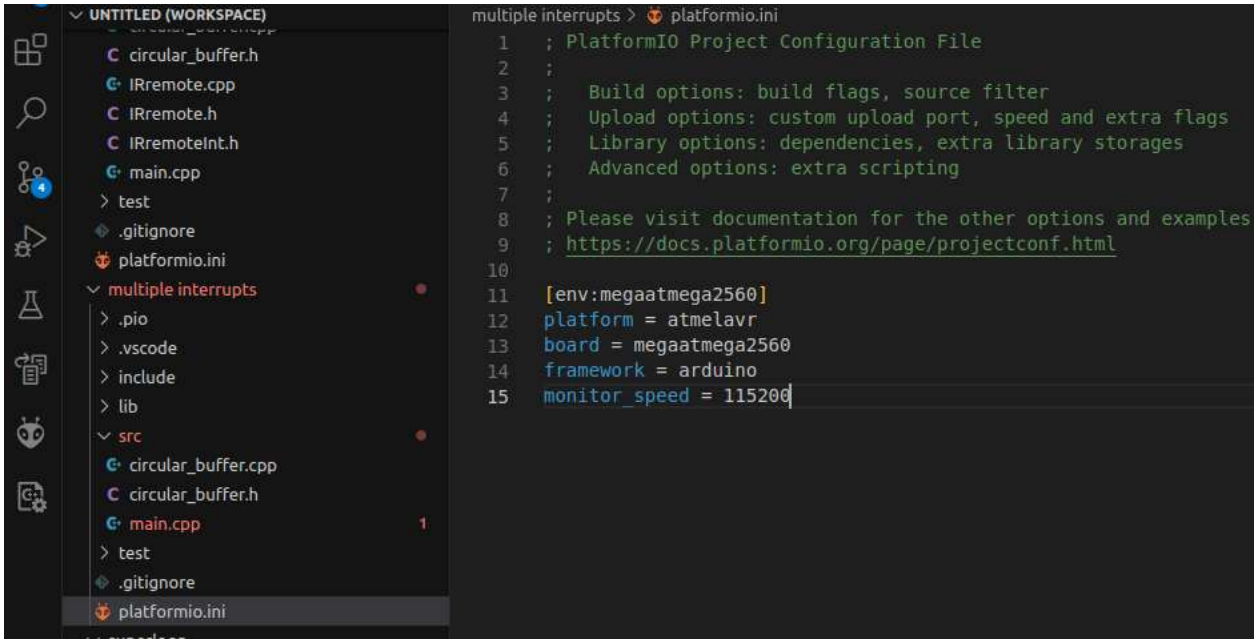
Once you have installed the extension, you can create a New project under PlatformIO Home. You can also open the project examples provided above.



Enter a name for your project, and set the board to “Arduino Mega or Mega 2560 ATMega2560 (Mega 2560)”. The Arduino framework will be selected automatically. Press Finish when ready.



You can set the baud rate for the serial monitor by adding “monitor_speed” to the platforio.ini file.



You can upload your project to the board by pressing the -> arrow button on the bottom toolbar. The plug-in button will open a serial monitor.

Installing libraries

The Elegoo folder can be downloaded here:
https://github.com/Iyury1/elegoo/archive/refs/heads/main.zip

The Elegoo kit comes with code examples for each breakout board in the kid. These code examples are located in:
“elegoo\ELEGOO The Most Complete Starter Kit for MEGA V1.0.2022.03.24\English\code”

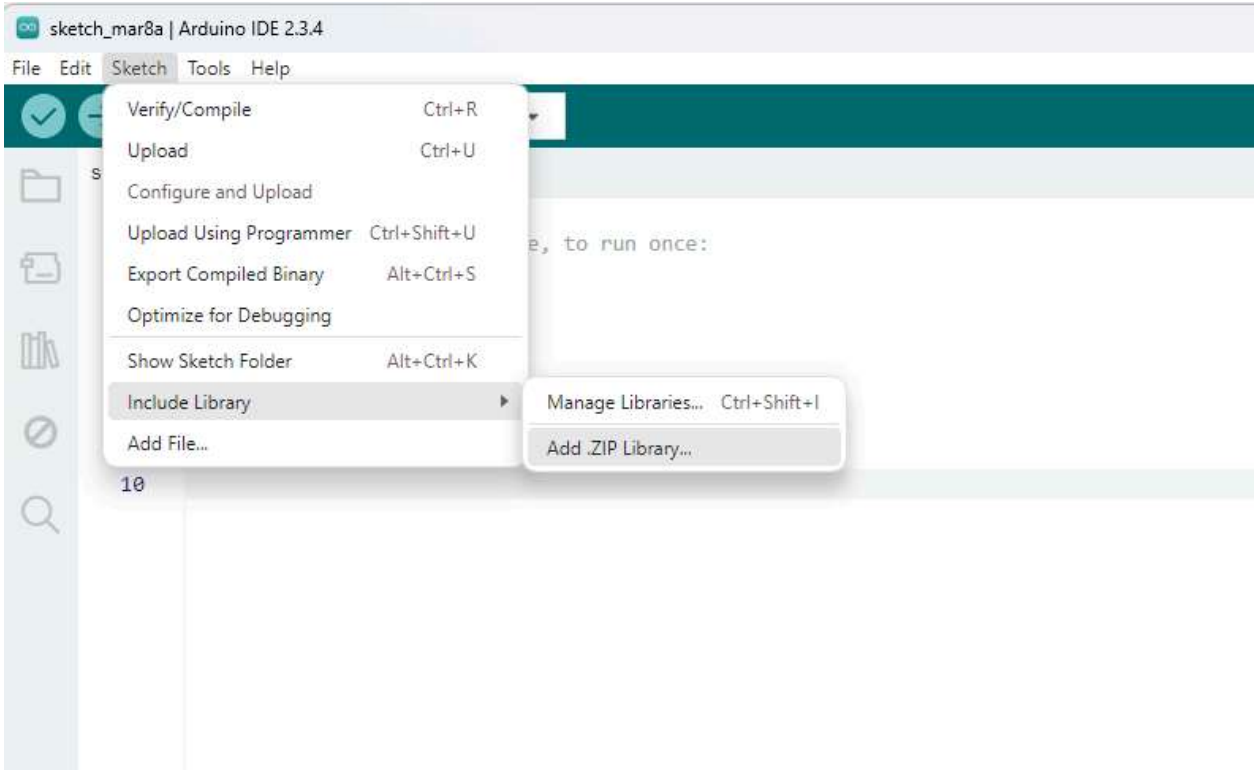
However, you must also install the library for the component you want to use.

The Elegoo libraries are located in the folder:
“elegoo\ELEGOO The Most Complete Starter Kit for MEGA V1.0.2022.03.24\English\Libraries

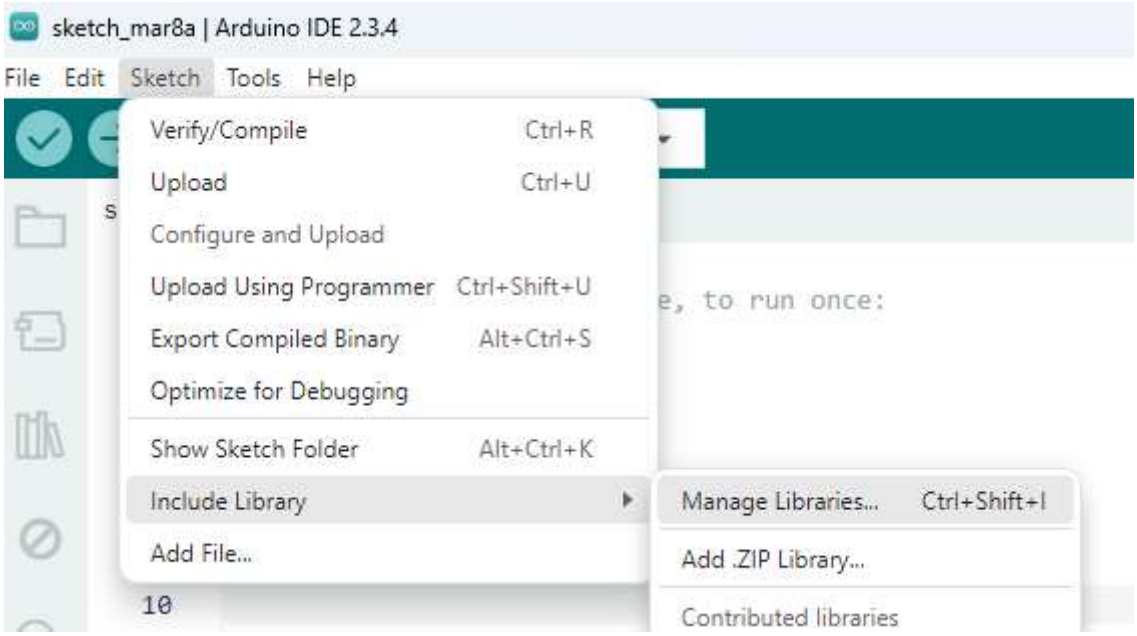
Arduino libraries

If you are using the Arduino IDE, you can install a library by clicking
“Sketch -> Include Library -> Add .ZIP Library…”

Then, select the ZIP file for the library you want to use from the Elegoo folder. After the library is installed, the code example should function correctly.

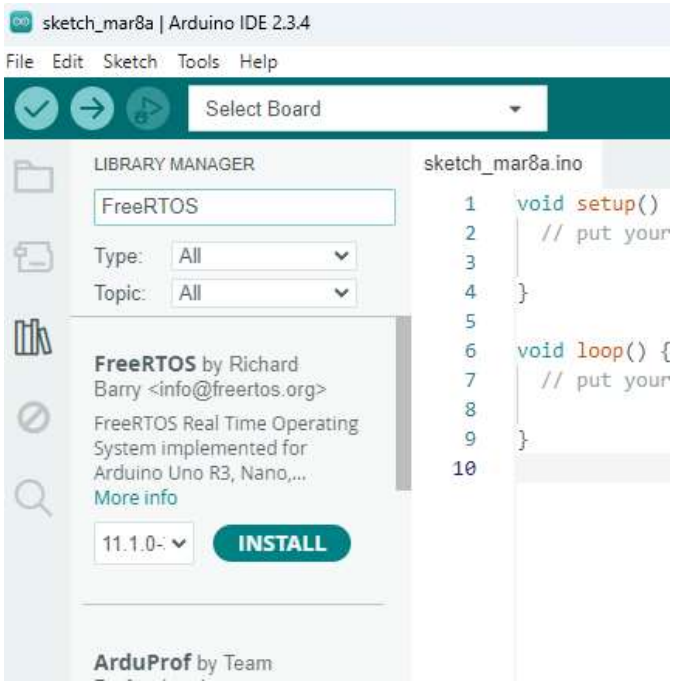


You can also install libraries from the Arduino library manager by clicking “Sketch -> Include Library -> Manage Libraries…”



You can then search for libraries and install them from the Library Manager toolbar.

NOTE: Make sure you use the exact libraries from the Elegoo folder to use the breakout boards included in the kit. New library versions might not work with the hardware you have.



L298N Motor Driver Example

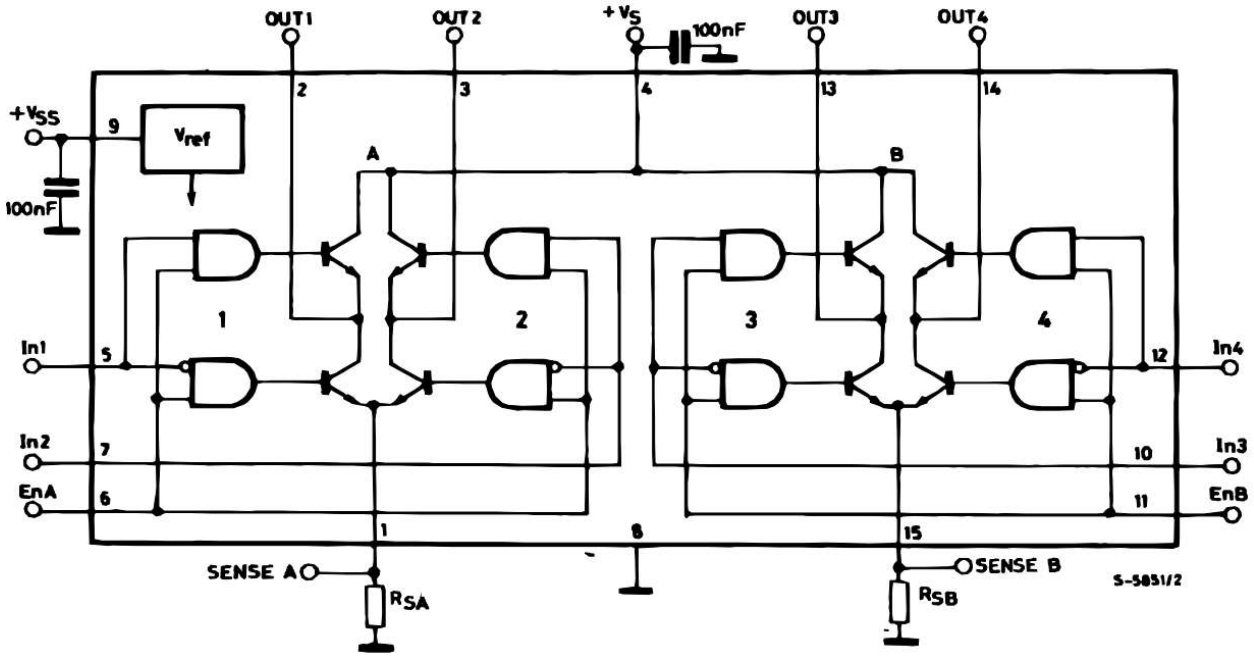


Figure 4: Datasheet for STM32 L298 Dual full-bridge driver

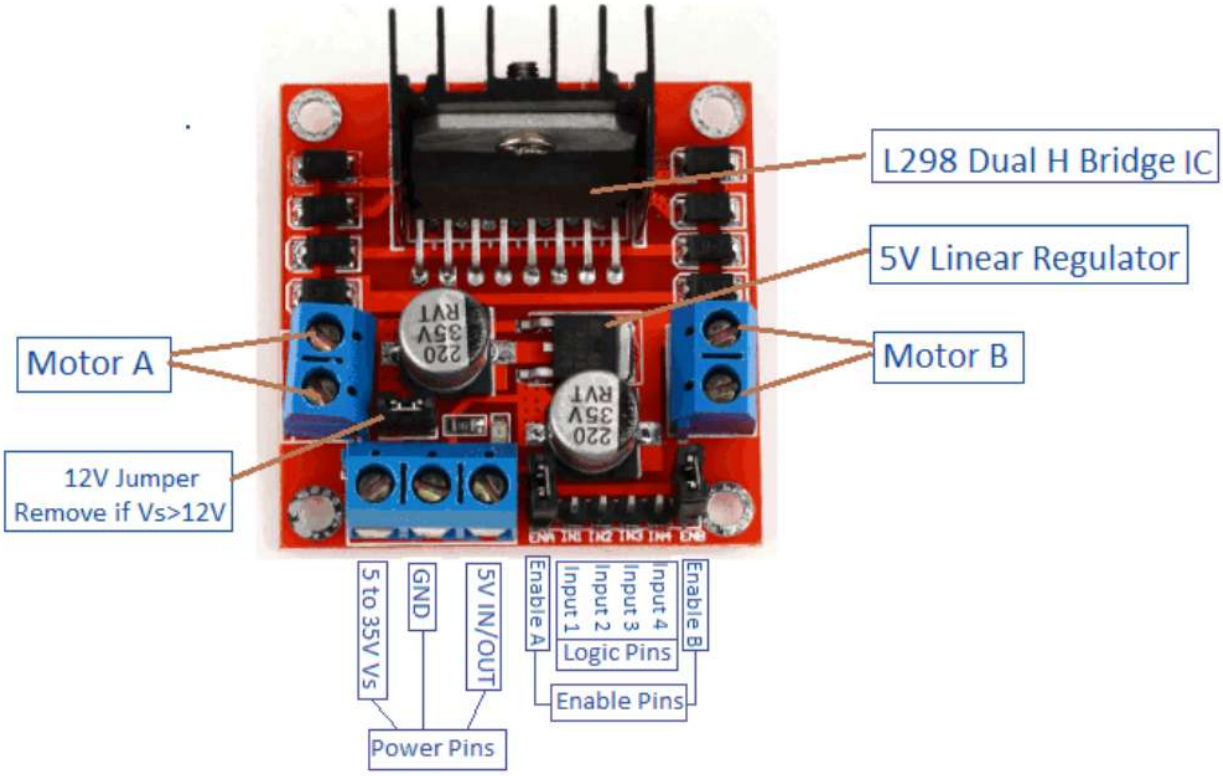


Figure 5: Top view of L298N breakout board

Our battery pack can hold 4 AA batteries, which can have 1.2 - 1.5V each, so we can leave the 12V jumper in place. The positive wire of the battery pack should be connected to the “4 to 35V Vs” power pin of the L298 breakout board, and the negative battery cable should connect to “GND”.

If we leave the “Enable” jumpers in place, then the enable pins are set to 5V. This will make sure the H-bridge is always enabled. A PWM could be applied to the enable pin if the jumper is removed.

Make sure that both the battery and Arduino share a common ground.

```
int dir1PinA = 2;
int dir2PinA = 3;

pinMode(dir1PinA,OUTPUT);
pinMode(dir2PinA,OUTPUT);

// Forward
digitalWrite(dir1PinA, LOW);
digitalWrite(dir2PinA, HIGH);
```

```
// Stop
digitalWrite(dir1PinA, LOW);
digitalWrite(dir2PinA, LOW);

// Reverse
digitalWrite(dir1PinA, HIGH);
digitalWrite(dir2PinA, LOW);
```

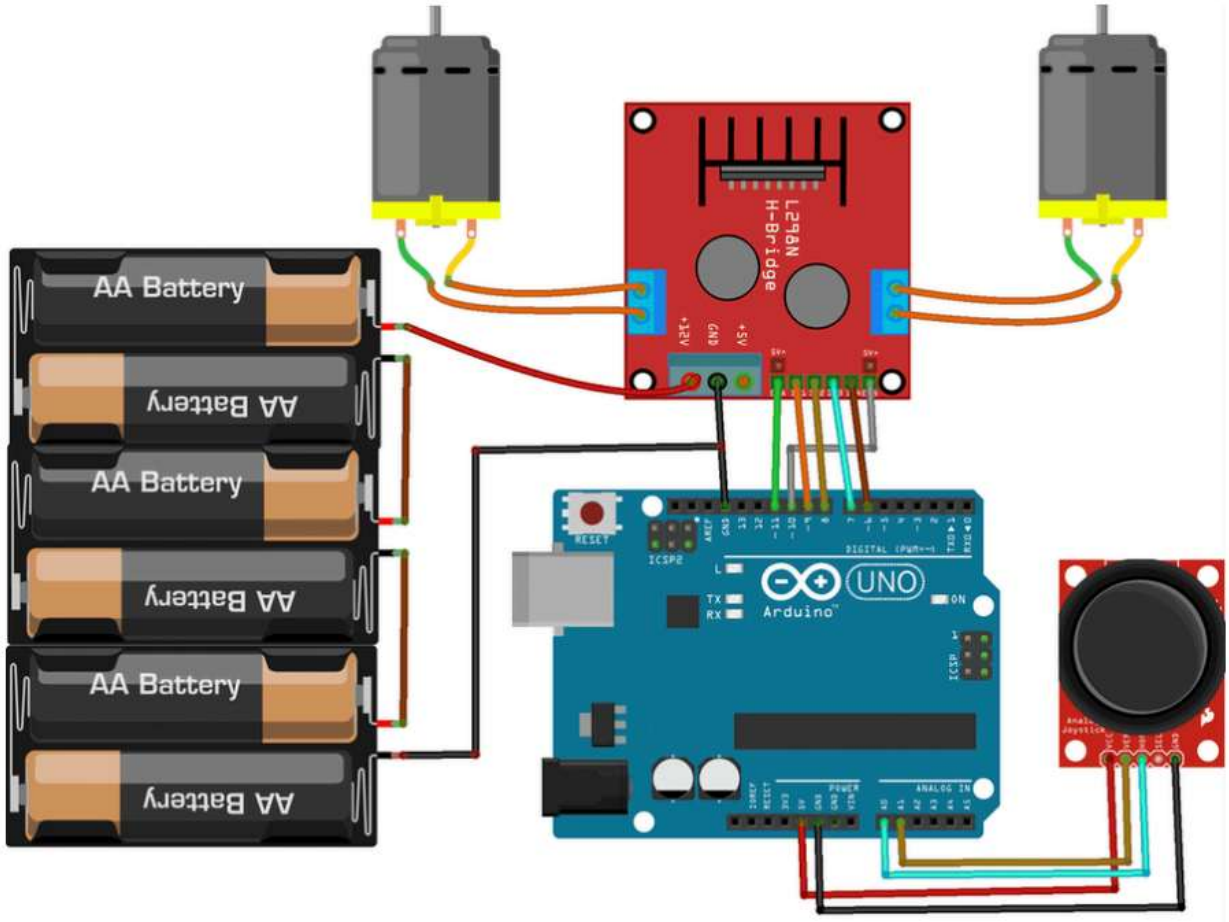
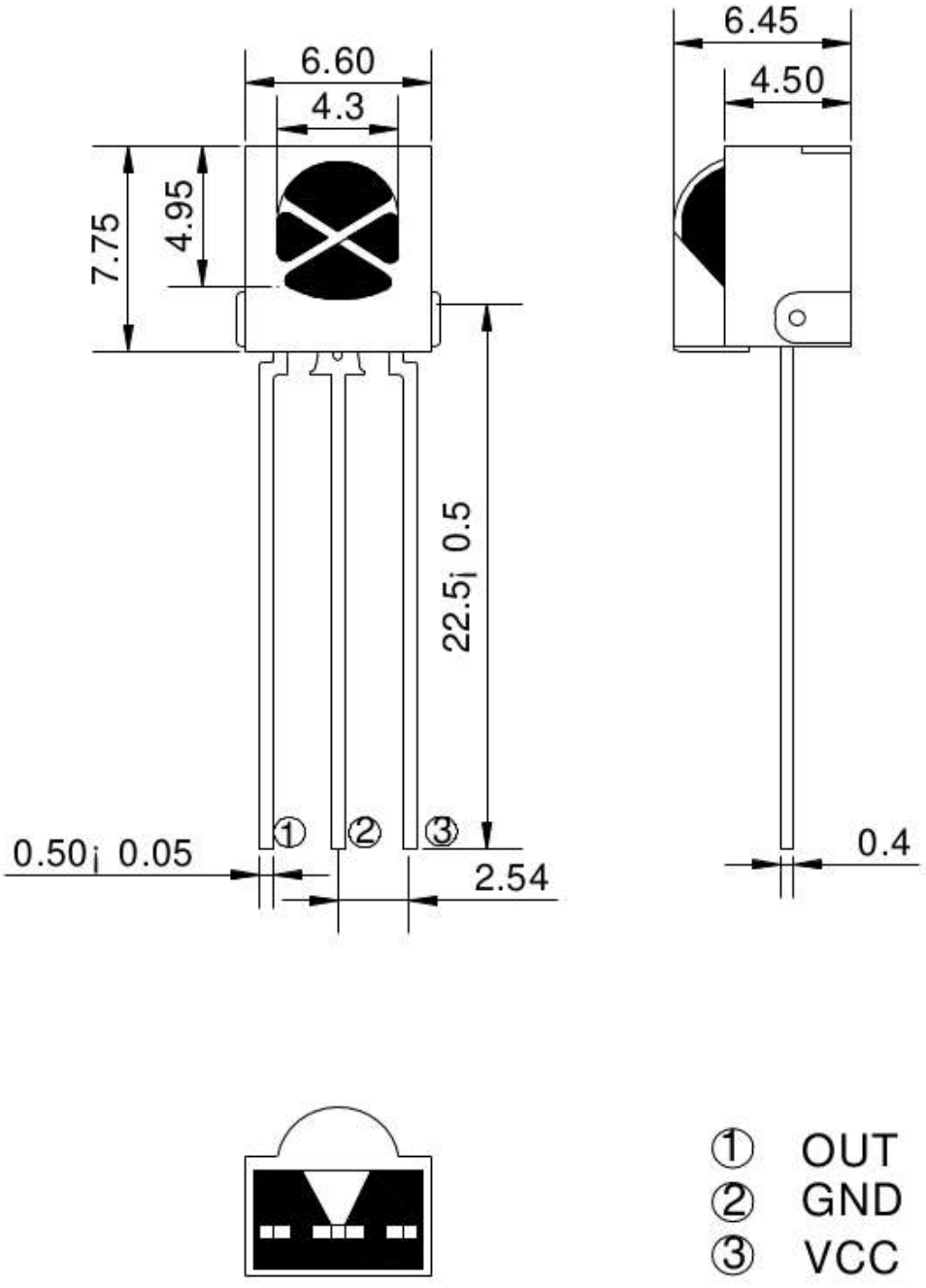


Figure 6: Wiring instructions for L298

https://www.st.com/resource/en/datasheet/l298.pdf
https://www.instructables.com/Running-DC-Motor-With-Arduino-Using-L298N-Motor-Dr/
https://www.instructables.com/Arduino-Modules-L298N-Dual-H-Bridge-Motor-Controll/
https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/

IR Remote Example



How can we make the motor respond to the IR Remote?

We can read the button sent by the IR Remote in one task, and pass a message to another task that will control the motor.

The “harmonic loop periods” project has an example of how to do this:

```
void task1()
{
 Serial.println("Task 1");
 if (irrecv.decode(&results)) // have we received an IR signal?
 {
   irrecv.resume(); // receive the next value
   translateIR();
 }
}
```

```
void task3()
{
 Serial.println("Task 3");
 switch(motor_state)
 {
   case FORWARD:
     motor_forward();
   break;
   case REVERSE:
     motor_reverse();
   break;
   case STOP:
     motor_stop();
   break;
   default:
   break;
 }
 if (next_motor_state != motor_state) {
   motor_state = next_motor_state;
 }
}
```

```
// describing Remote IR codes
{
 switch(results.value)

 {
 case 0xFFA25D: Serial.println("POWER"); break;
 case 0xFFE21D: Serial.println("FUNC/STOP"); break;
 case 0xFF629D: Serial.println("VOL+"); break;
  case 0xFF22DD:
   Serial.println("FAST BACK");
   if(motor_state == FORWARD)
     next_motor_state = STOP;
   else
     next_motor_state = REVERSE;
   break;

 case 0xFF02FD:
   Serial.println("PAUSE");
   next_motor_state = STOP;
   break;

 case 0xFFC23D:
   Serial.println("FAST FORWARD");
   if(motor_state == REVERSE)
     next_motor_state = STOP;
   else
     next_motor_state = FORWARD;
   break;

 case 0xFFE01F: Serial.println("DOWN");    break;
 case 0xFFA857: Serial.println("VOL-");    break;
 case 0xFF906F: Serial.println("UP");    break;
 case 0xFF9867: Serial.println("EQ");    break;
 case 0xFFB04F: Serial.println("ST/REPT");    break;
 case 0xFF6897: Serial.println("0");    break;
 case 0xFF30CF: Serial.println("1");    break;
 case 0xFF18E7: Serial.println("2");    break;
 case 0xFF7A85: Serial.println("3");    break;
 case 0xFF10EF: Serial.println("4");    break;
 case 0xFF38C7: Serial.println("5");    break;
 case 0xFF5AA5: Serial.println("6");    break;
 case 0xFF42BD: Serial.println("7");    break;
 case 0xFF4AB5: Serial.println("8");    break;
 case 0xFF52AD: Serial.println("9");    break;
 case 0xFFFFFFFF: Serial.println(" REPEAT");break;

 default:
```

```
    Serial.println(" other button   ");

  }// End Case
  key_value = results.value;
  // delay(500); // Do not get immediate repeat


 } //END translateIR
```

digitalRead()
digitalWrite()
analogRead()