// ---------------------------------------------------------------------------

// NewPing Library - v1.9.7 - 02/16/2023

//

// AUTHOR/LICENSE:

// Created by Tim Eckel - eckel.tim@gmail.com

// Copyright 2023 License: Forks and derivitive works are NOT permitted without

// permission. Permission is only granted to use as-is for private and

// commercial use. If you wish to contribute, make changes, or enhancements,

// please create a pull request. I get a lot of support issues from this

// library, much of which comes from old versions, buggy forks or sites without

// proper documentation. Trying to wrangle this project together, thanks!

//

// LINKS:

// Project home: https://bitbucket.org/teckel12/arduino-new-ping/wiki/Home

// Support: https://bitbucket.org/teckel12/arduino-new-ping/issues?status=new&status=open

//

// DISCLAIMER:

// This software is furnished "as is", without technical support, and with no

// warranty, express or implied, as to its usefulness for any purpose.

//

// BACKGROUND:

// Initially, I wasn't happy with how poorly ultrasonic sensors performed. I

// soon realized the problem wasn't the sensor, it was the available ping and

// ultrasonic libraries causing the problem. The NewPing library totally fixes

// these problems, adds many new features, and breathes new life into these

// very affordable distance sensors.

//

// FEATURES:

// \* Works with many different ultrasonic sensors: SR04, SRF05, SRF06, DYP-ME007, URM37 & Parallax PING)))™.

// \* Compatible with the entire Arduino line-up (and clones), Teensy family (including $19.80 96Mhz 32 bit Teensy 3.2) and non-AVR microcontrollers.

// \* Interface with all but the SRF06 sensor using only one Arduino pin.

// \* Doesn't lag for a full second if no ping/echo is received.

// \* Ping sensors consistently and reliably at up to 30 times per second.

// \* Timer interrupt method for event-driven sketches.

// \* Built-in digital filter method ping\_median() for easy error correction.

// \* Uses port registers for a faster pin interface and smaller code size.

// \* Allows you to set a maximum distance where pings beyond that distance are read as no ping "clear".

// \* Ease of using multiple sensors (example sketch with 15 sensors).

// \* More accurate distance calculation (cm, inches & uS).

// \* Doesn't use pulseIn, which is slow and gives incorrect results with some ultrasonic sensor models.

// \* Actively developed with features being added and bugs/issues addressed.

//

// CONSTRUCTOR:

// NewPing sonar(trigger\_pin, echo\_pin [, max\_cm\_distance])

// trigger\_pin & echo\_pin - Arduino pins connected to sensor trigger and echo.

// NOTE: To use the same Arduino pin for trigger and echo, specify the same pin for both values.

// max\_cm\_distance - [Optional] Maximum distance you wish to sense. Default=500cm.

//

// METHODS:

// sonar.ping([max\_cm\_distance]) - Send a ping and get the echo time (in microseconds) as a result. [max\_cm\_distance] allows you to optionally set a new max distance.

// sonar.ping\_in([max\_cm\_distance]) - Send a ping and get the distance in whole inches. [max\_cm\_distance] allows you to optionally set a new max distance.

// sonar.ping\_cm([max\_cm\_distance]) - Send a ping and get the distance in whole centimeters. [max\_cm\_distance] allows you to optionally set a new max distance.

// sonar.ping\_median(iterations [, max\_cm\_distance]) - Do multiple pings (default=5), discard out of range pings and return median in microseconds. [max\_cm\_distance] allows you to optionally set a new max distance.

// NewPing::convert\_in(echoTime) - Convert echoTime from microseconds to inches (rounds to nearest inch).

// NewPing::convert\_cm(echoTime) - Convert echoTime from microseconds to centimeters (rounds to nearest cm).

// sonar.ping\_timer(function [, max\_cm\_distance]) - Send a ping and call function to test if ping is complete. [max\_cm\_distance] allows you to optionally set a new max distance.

// sonar.check\_timer() - Check if ping has returned within the set distance limit.

// NewPing::timer\_us(frequency, function) - Call function every frequency microseconds.

// NewPing::timer\_ms(frequency, function) - Call function every frequency milliseconds.

// NewPing::timer\_stop() - Stop the timer.

//

// HISTORY:

// 02/16/2023 v1.9.7 - ONE\_PIN\_ENABLED mode is now automatic, based on if you

// use the same trigger and echo pins. Echo TRIGGER\_WIDTH can now be modified

// if your sensors are out of spec (defaults to 12uS, previously 10uS).

//

// 12/15/2022 v1.9.6 - Reverted timer overflow commit as it contained bugs.

//

// 12/13/2022 v1.9.5 - Resolved micros() timer overflow bug. Changed example

// sketch extensions so they load in latest release of the Arduino IDE.

// Changed private members to protected for easier 3rd party customization.

//

// 01/26/2022 v1.9.4 - Added esp32 to the compatible architectures (note:

// non-timer methods only)

//

// 01/20/2022 v1.9.3 - Default to disable timer methods for non-AVR

// microcontrollers and on the ATmega4809 used in the Nano Every.

//

// 07/15/2018 v1.9.1 - Added support for ATtiny441 and ATtiny841

// microcontrollers.

//

// 12/09/2017 v1.9.0 - Added support for the ARM-based Particle devices. If

// other ARM-based microcontrollers adopt a similar timer method that the

// Particle and Teensy 3.x share, support for other ARM-based microcontrollers

// could be possible. Attempt to add to Arduino library manager. License

// changed.

//

// 07/30/2016 v1.8 - Added support for non-AVR microcontrollers. For non-AVR

// microcontrollers, advanced ping\_timer() timer methods are disabled due to

// inconsistencies or no support at all between platforms. However, standard

// ping methods are all supported. Added new optional variable to ping(),

// ping\_in(), ping\_cm(), ping\_median(), and ping\_timer() methods which allows

// you to set a new maximum distance for each ping. Added support for the

// ATmega16, ATmega32 and ATmega8535 microcontrollers. Changed convert\_cm()

// and convert\_in() methods to static members. You can now call them without

// an object. For example: cm = NewPing::convert\_cm(distance);

//

// 09/29/2015 v1.7 - Removed support for the Arduino Due and Zero because

// they're both 3.3 volt boards and are not 5 volt tolerant while the HC-SR04

// is a 5 volt sensor. Also, the Due and Zero don't support pin manipulation

// compatibility via port registers which can be done (see the Teensy 3.2).

//

// 06/17/2014 v1.6 - Corrected delay between pings when using ping\_median()

// method. Added support for the URM37 sensor (must change URM37\_ENABLED from

// false to true). Added support for Arduino microcontrollers like the $20

// 32 bit ARM Cortex-M4 based Teensy 3.2. Added automatic support for the

// Atmel ATtiny family of microcontrollers. Added timer support for the

// ATmega8 microcontroller. Rounding disabled by default, reduces compiled

// code size (can be turned on with ROUNDING\_ENABLED switch). Added

// TIMER\_ENABLED switch to get around compile-time "\_\_vector\_7" errors when

// using the Tone library, or you can use the toneAC, NewTone or

// TimerFreeTone libraries: https://bitbucket.org/teckel12/arduino-toneac/

// Other speed and compiled size optimizations.

//

// 08/15/2012 v1.5 - Added ping\_median() method which does a user specified

// number of pings (default=5) and returns the median ping in microseconds

// (out of range pings ignored). This is a very effective digital filter.

// Optimized for smaller compiled size (even smaller than sketches that

// don't use a library).

//

// 07/14/2012 v1.4 - Added support for the Parallax PING)))� sensor. Interface

// with all but the SRF06 sensor using only one Arduino pin. You can also

// interface with the SRF06 using one pin if you install a 0.1uf capacitor

// on the trigger and echo pins of the sensor then tie the trigger pin to

// the Arduino pin (doesn't work with Teensy). To use the same Arduino pin

// for trigger and echo, specify the same pin for both values. Various bug

// fixes.

//

// 06/08/2012 v1.3 - Big feature addition, event-driven ping! Uses Timer2

// interrupt, so be mindful of PWM or timing conflicts messing with Timer2

// may cause (namely PWM on pins 3 & 11 on Arduino, PWM on pins 9 and 10 on

// Mega, and Tone library). Simple to use timer interrupt functions you can

// use in your sketches totally unrelated to ultrasonic sensors (don't use if

// you're also using NewPing's ping\_timer because both use Timer2 interrupts).

// Loop counting ping method deleted in favor of timing ping method after

// inconsistent results kept surfacing with the loop timing ping method.

// Conversion to cm and inches now rounds to the nearest cm or inch. Code

// optimized to save program space and fixed a couple minor bugs here and

// there. Many new comments added as well as line spacing to group code

// sections for better source readability.

//

// 05/25/2012 v1.2 - Lots of code clean-up thanks to Arduino Forum members.

// Rebuilt the ping timing code from scratch, ditched the pulseIn code as it

// doesn't give correct results (at least with ping sensors). The NewPing

// library is now VERY accurate and the code was simplified as a bonus.

// Smaller and faster code as well. Fixed some issues with very close ping

// results when converting to inches. All functions now return 0 only when

// there's no ping echo (out of range) and a positive value for a successful

// ping. This can effectively be used to detect if something is out of range

// or in-range and at what distance. Now compatible with Arduino 0023.

//

// 05/16/2012 v1.1 - Changed all I/O functions to use low-level port registers

// for ultra-fast and lean code (saves from 174 to 394 bytes). Tested on both

// the Arduino Uno and Teensy 2.0 but should work on all Arduino-based

// platforms because it calls standard functions to retrieve port registers

// and bit masks. Also made a couple minor fixes to defines.

//

// 05/15/2012 v1.0 - Initial release.

// ---------------------------------------------------------------------------

#ifndef NewPing\_h

#define NewPing\_h

#if defined(ARDUINO) && ARDUINO >= 100

#include <Arduino.h>

#else

#include <WProgram.h>

#if defined(PARTICLE)

#include <SparkIntervalTimer.h>

#else

#include <pins\_arduino.h>

#endif

#endif

#if defined(\_\_AVR\_\_)

#include <avr/io.h>

#include <avr/interrupt.h>

#endif

// Shouldn't need to change these values unless you have a specific need to do so.

#define MAX\_SENSOR\_DISTANCE 500 // Maximum sensor distance can be as high as 500cm, no reason to wait for ping longer than sound takes to travel this distance and back. Default=500

#define US\_ROUNDTRIP\_CM 57 // Microseconds (uS) it takes sound to travel round-trip 1cm (2cm total), uses integer to save compiled code space. Default=57

#define US\_ROUNDTRIP\_IN 146 // Microseconds (uS) it takes sound to travel round-trip 1 inch (2 inches total), uses integer to save compiled code space. Default=146

#define ROUNDING\_ENABLED false // Set to "true" to enable distance rounding which also adds 64 bytes to binary size. Default=false

#define URM37\_ENABLED false // Set to "true" to enable support for the URM37 sensor in PWM mode. Default=false

#define TRIGGER\_WIDTH 12 // Microseconds (uS) notch to trigger sensor to start ping. Sensor specs state notch should be 10uS, defaults to 12uS for out of spec sensors. Default=12

// Probably shouldn't change these values unless you really know what you're doing.

#define NO\_ECHO 0 // Value returned if there's no ping echo within the specified MAX\_SENSOR\_DISTANCE or max\_cm\_distance. Default=0

#define MAX\_SENSOR\_DELAY 5800 // Maximum uS it takes for sensor to start the ping. Default=5800

#define ECHO\_TIMER\_FREQ 24 // Frequency (in microseconds) to check for a ping echo (every 24uS is about 0.4cm accuracy). Default=24

#define PING\_MEDIAN\_DELAY 30000 // Microsecond delay between pings in the ping\_median method. Default=30000

#if URM37\_ENABLED == true

#undef US\_ROUNDTRIP\_CM

#undef US\_ROUNDTRIP\_IN

#define US\_ROUNDTRIP\_CM 50 // Every 50uS PWM signal is low indicates 1cm distance. Default=50

#define US\_ROUNDTRIP\_IN 127 // If 50uS is 1cm, 1 inch would be 127uS (50 x 2.54 = 127). Default=127

#endif

// Conversion from uS to distance (round result to nearest cm or inch).

#define NewPingConvert(echoTime, conversionFactor) (max(((unsigned int)echoTime + conversionFactor / 2) / conversionFactor, (echoTime ? 1 : 0)))

// Detect non-AVR microcontrollers (Teensy 3.x, Arduino DUE, etc.) and don't use port registers or timer interrupts as required.

#if (defined(\_\_arm\_\_) && (defined(TEENSYDUINO) || defined(PARTICLE)))

#define PING\_OVERHEAD 1

#define PING\_TIMER\_OVERHEAD 1

#define TIMER\_ENABLED true

#define DO\_BITWISE true

#elif defined(\_\_AVR\_\_)

#define PING\_OVERHEAD 5 // Ping overhead in microseconds (uS). Default=5

#define PING\_TIMER\_OVERHEAD 13 // Ping timer overhead in microseconds (uS). Default=13

#define TIMER\_ENABLED true

#define DO\_BITWISE true

#else

#define PING\_OVERHEAD 1

#define PING\_TIMER\_OVERHEAD 1

#define TIMER\_ENABLED false

#define DO\_BITWISE false

#endif

// Disable the timer interrupts when using ATmega128, ATmega4809 and all ATtiny microcontrollers.

#if defined(\_\_AVR\_ATmega128\_\_) || defined(\_\_AVR\_ATmega4809\_\_) || defined(\_\_AVR\_ATtiny24\_\_) || defined(\_\_AVR\_ATtiny44\_\_) || defined(\_\_AVR\_ATtiny441\_\_) || defined(\_\_AVR\_ATtiny84\_\_) || defined(\_\_AVR\_ATtiny841\_\_) || defined(\_\_AVR\_ATtiny25\_\_) || defined(\_\_AVR\_ATtiny45\_\_) || defined(\_\_AVR\_ATtiny85\_\_) || defined(\_\_AVR\_ATtiny261\_\_) || defined(\_\_AVR\_ATtiny461\_\_) || defined(\_\_AVR\_ATtiny861\_\_) || defined(\_\_AVR\_ATtiny43U\_\_) || defined(\_\_AVR\_ATtiny1614\_\_)

#undef TIMER\_ENABLED

#define TIMER\_ENABLED false

#endif

// Define timers when using ATmega8, ATmega16, ATmega32 and ATmega8535 microcontrollers.

#if defined(\_\_AVR\_ATmega8\_\_) || defined(\_\_AVR\_ATmega16\_\_) || defined(\_\_AVR\_ATmega32\_\_) || defined(\_\_AVR\_ATmega8535\_\_)

#define OCR2A OCR2

#define TIMSK2 TIMSK

#define OCIE2A OCIE2

#endif

class NewPing {

public:

NewPing(uint8\_t trigger\_pin, uint8\_t echo\_pin, unsigned int max\_cm\_distance = MAX\_SENSOR\_DISTANCE);

unsigned int ping(unsigned int max\_cm\_distance = 0);

unsigned long ping\_cm(unsigned int max\_cm\_distance = 0);

unsigned long ping\_in(unsigned int max\_cm\_distance = 0);

unsigned long ping\_median(uint8\_t it = 5, unsigned int max\_cm\_distance = 0);

static unsigned int convert\_cm(unsigned int echoTime);

static unsigned int convert\_in(unsigned int echoTime);

#if TIMER\_ENABLED == true

void ping\_timer(void (\*userFunc)(void), unsigned int max\_cm\_distance = 0);

boolean check\_timer();

unsigned long ping\_result;

static void timer\_us(unsigned int frequency, void (\*userFunc)(void));

static void timer\_ms(unsigned long frequency, void (\*userFunc)(void));

static void timer\_stop();

#endif

protected:

boolean ping\_trigger();

void set\_max\_distance(unsigned int max\_cm\_distance);

#if TIMER\_ENABLED == true

boolean ping\_trigger\_timer(unsigned int trigger\_delay);

boolean ping\_wait\_timer();

static void timer\_setup();

static void timer\_ms\_cntdwn();

#endif

#if DO\_BITWISE == true

uint8\_t \_triggerBit;

uint8\_t \_echoBit;

#if defined(PARTICLE)

#if !defined(portModeRegister)

#if defined(STM32F10X\_MD)

#define portModeRegister(port) ( &(port->CRL) )

#elif defined(STM32F2XX)

#define portModeRegister(port) ( &(port->MODER) )

#endif

#endif

volatile uint32\_t \*\_triggerOutput;

volatile uint32\_t \*\_echoInput;

volatile uint32\_t \*\_triggerMode;

#else

volatile uint8\_t \*\_triggerOutput;

volatile uint8\_t \*\_echoInput;

volatile uint8\_t \*\_triggerMode;

#endif

#else

uint8\_t \_triggerPin;

uint8\_t \_echoPin;

#endif

unsigned int \_maxEchoTime;

unsigned long \_max\_time;

bool \_one\_pin\_mode;

};

#endif