/\*

|| @author Brett Hagman <bhagman@wiring.org.co>

|| @contribution Fotis Papadopoulos <fpapadopou@gmail.com>

|| @url http://wiring.org.co/

|| @url http://roguerobotics.com/

||

|| @description

|| | A Software Digital Square Wave Tone Generation Library

|| |

|| | Written by Brett Hagman

|| | http://www.roguerobotics.com/

|| | bhagman@roguerobotics.com, bhagman@wiring.org.co

|| |

|| | This is a Wiring Framework (Arduino) library to produce square-wave

|| | tones on an arbitrary pin.

|| |

|| | You can make multiple instances of the Tone object, to create tones on

|| | different pins.

|| |

|| | The number of tones that can be generated at the same time is limited

|| | by the number of hardware timers available on the hardware.

|| | (e.g. ATmega328 has 3 available timers, and the ATmega1280 has 6 timers)

|| |

|| | A simplified (single tone) version of this library has been included

|| | in the Wiring Framework since Wiring 0025 and in the Arduino distribution

|| | since Arduino 0018.

|| |

|| #

||

|| @license Please see the accompanying LICENSE.txt file for this project.

||

|| @name Software PWM Library

|| @type Library

|| @target Atmel AVR 8 Bit

||

|| @version 1.0.0

||

\*/

#if defined(WIRING)

#include <Wiring.h>

#elif ARDUINO >= 100

#include <Arduino.h>

#else

#include <WProgram.h>

#endif

#include "Tone.h"

#if defined(\_\_AVR\_ATmega8\_\_)

#define TCCR2A TCCR2

#define TCCR2B TCCR2

#define COM2A1 COM21

#define COM2A0 COM20

#define OCR2A OCR2

#define TIMSK2 TIMSK

#define OCIE2A OCIE2

#define TIMER2\_COMPA\_vect TIMER2\_COMP\_vect

#define TIMSK1 TIMSK

#endif

// timerx\_toggle\_count:

// > 0 - duration specified

// = 0 - stopped

// < 0 - infinitely (until stop() method called, or new play() called)

#if !defined(\_\_AVR\_ATmega8\_\_)

volatile int32\_t timer0\_toggle\_count;

volatile uint8\_t \*timer0\_pin\_port;

volatile uint8\_t timer0\_pin\_mask;

#endif

volatile int32\_t timer1\_toggle\_count;

volatile uint8\_t \*timer1\_pin\_port;

volatile uint8\_t timer1\_pin\_mask;

volatile int32\_t timer2\_toggle\_count;

volatile uint8\_t \*timer2\_pin\_port;

volatile uint8\_t timer2\_pin\_mask;

#if defined(\_\_AVR\_ATmega1280\_\_)

volatile int32\_t timer3\_toggle\_count;

volatile uint8\_t \*timer3\_pin\_port;

volatile uint8\_t timer3\_pin\_mask;

volatile int32\_t timer4\_toggle\_count;

volatile uint8\_t \*timer4\_pin\_port;

volatile uint8\_t timer4\_pin\_mask;

volatile int32\_t timer5\_toggle\_count;

volatile uint8\_t \*timer5\_pin\_port;

volatile uint8\_t timer5\_pin\_mask;

#endif

#if defined(\_\_AVR\_ATmega1280\_\_)

#define AVAILABLE\_TONE\_PINS 6

// Leave timers 1, and zero to last.

const uint8\_t PROGMEM tone\_pin\_to\_timer\_PGM[] = { 2, 3, 4, 5, 1, 0 };

#elif defined(\_\_AVR\_ATmega8\_\_)

#define AVAILABLE\_TONE\_PINS 2

const uint8\_t PROGMEM tone\_pin\_to\_timer\_PGM[] = { 2, 1 };

#else

#define AVAILABLE\_TONE\_PINS 3

// Leave timer 0 to last.

const uint8\_t PROGMEM tone\_pin\_to\_timer\_PGM[] = { 2, 1, 0 };

#endif

// Initialize our pin count

uint8\_t Tone::\_tone\_pin\_count = 0;

// Interrupt routines

#if !defined(\_\_AVR\_ATmega8\_\_)

#ifdef WIRING

void Tone\_Timer0\_Interrupt(void)

#else

ISR(TIMER0\_COMPA\_vect)

#endif

{

if (timer0\_toggle\_count != 0)

{

// toggle the pin

\*timer0\_pin\_port ^= timer0\_pin\_mask;

if (timer0\_toggle\_count > 0)

timer0\_toggle\_count--;

}

else

{

TIMSK0 &= ~(1 << OCIE0A); // disable the interrupt

\*timer0\_pin\_port &= ~(timer0\_pin\_mask); // keep pin low after stop

}

}

#endif

#ifdef WIRING

void Tone\_Timer1\_Interrupt(void)

#else

ISR(TIMER1\_COMPA\_vect)

#endif

{

if (timer1\_toggle\_count != 0)

{

// toggle the pin

\*timer1\_pin\_port ^= timer1\_pin\_mask;

if (timer1\_toggle\_count > 0)

timer1\_toggle\_count--;

}

else

{

TIMSK1 &= ~(1 << OCIE1A); // disable the interrupt

\*timer1\_pin\_port &= ~(timer1\_pin\_mask); // keep pin low after stop

}

}

#ifdef WIRING

void Tone\_Timer2\_Interrupt(void)

#else

ISR(TIMER2\_COMPA\_vect)

#endif

{

int32\_t temp\_toggle\_count = timer2\_toggle\_count;

if (temp\_toggle\_count != 0)

{

// toggle the pin

\*timer2\_pin\_port ^= timer2\_pin\_mask;

if (temp\_toggle\_count > 0)

temp\_toggle\_count--;

}

else

{

TIMSK2 &= ~(1 << OCIE2A); // disable the interrupt

\*timer2\_pin\_port &= ~(timer2\_pin\_mask); // keep pin low after stop

}

timer2\_toggle\_count = temp\_toggle\_count;

}

#if defined(\_\_AVR\_ATmega1280\_\_)

#ifdef WIRING

void Tone\_Timer3\_Interrupt(void)

#else

ISR(TIMER3\_COMPA\_vect)

#endif

{

if (timer3\_toggle\_count != 0)

{

// toggle the pin

\*timer3\_pin\_port ^= timer3\_pin\_mask;

if (timer3\_toggle\_count > 0)

timer3\_toggle\_count--;

}

else

{

TIMSK3 &= ~(1 << OCIE3A); // disable the interrupt

\*timer3\_pin\_port &= ~(timer3\_pin\_mask); // keep pin low after stop

}

}

#ifdef WIRING

void Tone\_Timer4\_Interrupt(void)

#else

ISR(TIMER4\_COMPA\_vect)

#endif

{

if (timer4\_toggle\_count != 0)

{

// toggle the pin

\*timer4\_pin\_port ^= timer4\_pin\_mask;

if (timer4\_toggle\_count > 0)

timer4\_toggle\_count--;

}

else

{

TIMSK4 &= ~(1 << OCIE4A); // disable the interrupt

\*timer4\_pin\_port &= ~(timer4\_pin\_mask); // keep pin low after stop

}

}

#ifdef WIRING

void Tone\_Timer5\_Interrupt(void)

#else

ISR(TIMER5\_COMPA\_vect)

#endif

{

if (timer5\_toggle\_count != 0)

{

// toggle the pin

\*timer5\_pin\_port ^= timer5\_pin\_mask;

if (timer5\_toggle\_count > 0)

timer5\_toggle\_count--;

}

else

{

TIMSK5 &= ~(1 << OCIE5A); // disable the interrupt

\*timer5\_pin\_port &= ~(timer5\_pin\_mask); // keep pin low after stop

}

}

#endif

void Tone::begin(uint8\_t tonePin)

{

if (\_tone\_pin\_count < AVAILABLE\_TONE\_PINS)

{

\_pin = tonePin;

\_timer = pgm\_read\_byte(tone\_pin\_to\_timer\_PGM + \_tone\_pin\_count);

\_tone\_pin\_count++;

// Set timer specific stuff

// All timers in CTC mode

// 8 bit timers will require changing prescalar values,

// whereas 16 bit timers are set to either ck/1 or ck/64 prescalar

switch (\_timer)

{

#if !defined(\_\_AVR\_ATmega8\_\_)

case 0:

// 8 bit timer

TCCR0A = 0;

TCCR0B = 0;

bitWrite(TCCR0A, WGM01, 1);

bitWrite(TCCR0B, CS00, 1);

timer0\_pin\_port = portOutputRegister(digitalPinToPort(\_pin));

timer0\_pin\_mask = digitalPinToBitMask(\_pin);

#ifdef WIRING

Timer0.attachInterrupt(INTERRUPT\_COMPARE\_MATCH\_A, Tone\_Timer0\_Interrupt);

#endif

break;

#endif

case 1:

// 16 bit timer

TCCR1A = 0;

TCCR1B = 0;

bitWrite(TCCR1B, WGM12, 1);

bitWrite(TCCR1B, CS10, 1);

timer1\_pin\_port = portOutputRegister(digitalPinToPort(\_pin));

timer1\_pin\_mask = digitalPinToBitMask(\_pin);

#ifdef WIRING

Timer1.attachInterrupt(INTERRUPT\_COMPARE\_MATCH\_A, Tone\_Timer1\_Interrupt);

#endif

break;

case 2:

// 8 bit timer

TCCR2A = 0;

TCCR2B = 0;

bitWrite(TCCR2A, WGM21, 1);

bitWrite(TCCR2B, CS20, 1);

timer2\_pin\_port = portOutputRegister(digitalPinToPort(\_pin));

timer2\_pin\_mask = digitalPinToBitMask(\_pin);

#ifdef WIRING

Timer2.attachInterrupt(INTERRUPT\_COMPARE\_MATCH\_A, Tone\_Timer2\_Interrupt);

#endif

break;

#if defined(\_\_AVR\_ATmega1280\_\_)

case 3:

// 16 bit timer

TCCR3A = 0;

TCCR3B = 0;

bitWrite(TCCR3B, WGM32, 1);

bitWrite(TCCR3B, CS30, 1);

timer3\_pin\_port = portOutputRegister(digitalPinToPort(\_pin));

timer3\_pin\_mask = digitalPinToBitMask(\_pin);

#ifdef WIRING

Timer3.attachInterrupt(INTERRUPT\_COMPARE\_MATCH\_A, Tone\_Timer3\_Interrupt);

#endif

break;

case 4:

// 16 bit timer

TCCR4A = 0;

TCCR4B = 0;

bitWrite(TCCR4B, WGM42, 1);

bitWrite(TCCR4B, CS40, 1);

timer4\_pin\_port = portOutputRegister(digitalPinToPort(\_pin));

timer4\_pin\_mask = digitalPinToBitMask(\_pin);

#ifdef WIRING

Timer4.attachInterrupt(INTERRUPT\_COMPARE\_MATCH\_A, Tone\_Timer4\_Interrupt);

#endif

break;

case 5:

// 16 bit timer

TCCR5A = 0;

TCCR5B = 0;

bitWrite(TCCR5B, WGM52, 1);

bitWrite(TCCR5B, CS50, 1);

timer5\_pin\_port = portOutputRegister(digitalPinToPort(\_pin));

timer5\_pin\_mask = digitalPinToBitMask(\_pin);

#ifdef WIRING

Timer5.attachInterrupt(INTERRUPT\_COMPARE\_MATCH\_A, Tone\_Timer5\_Interrupt);

#endif

break;

#endif

}

}

else

{

// disabled

\_timer = -1;

}

}

// frequency (in hertz) and duration (in milliseconds).

void Tone::play(uint16\_t frequency, uint32\_t duration)

{

uint8\_t prescalarbits = 0b001;

int32\_t toggle\_count = 0;

uint32\_t ocr = 0;

if (\_timer >= 0)

{

// Set the pinMode as OUTPUT

pinMode(\_pin, OUTPUT);

// if we are using an 8 bit timer, scan through prescalars to find the best fit

if (\_timer == 0 || \_timer == 2)

{

ocr = F\_CPU / frequency / 2 - 1;

prescalarbits = 0b001; // ck/1: same for both timers

if (ocr > 255)

{

ocr = F\_CPU / frequency / 2 / 8 - 1;

prescalarbits = 0b010; // ck/8: same for both timers

if (\_timer == 2 && ocr > 255)

{

ocr = F\_CPU / frequency / 2 / 32 - 1;

prescalarbits = 0b011;

}

if (ocr > 255)

{

ocr = F\_CPU / frequency / 2 / 64 - 1;

prescalarbits = \_timer == 0 ? 0b011 : 0b100;

if (\_timer == 2 && ocr > 255)

{

ocr = F\_CPU / frequency / 2 / 128 - 1;

prescalarbits = 0b101;

}

if (ocr > 255)

{

ocr = F\_CPU / frequency / 2 / 256 - 1;

prescalarbits = \_timer == 0 ? 0b100 : 0b110;

if (ocr > 255)

{

// can't do any better than /1024

ocr = F\_CPU / frequency / 2 / 1024 - 1;

prescalarbits = \_timer == 0 ? 0b101 : 0b111;

}

}

}

}

#if !defined(\_\_AVR\_ATmega8\_\_)

if (\_timer == 0)

TCCR0B = (TCCR0B & 0b11111000) | prescalarbits;

else

#endif

TCCR2B = (TCCR2B & 0b11111000) | prescalarbits;

}

else

{

// two choices for the 16 bit timers: ck/1 or ck/64

ocr = F\_CPU / frequency / 2 - 1;

prescalarbits = 0b001;

if (ocr > 0xffff)

{

ocr = F\_CPU / frequency / 2 / 64 - 1;

prescalarbits = 0b011;

}

if (\_timer == 1)

TCCR1B = (TCCR1B & 0b11111000) | prescalarbits;

#if defined(\_\_AVR\_ATmega1280\_\_)

else if (\_timer == 3)

TCCR3B = (TCCR3B & 0b11111000) | prescalarbits;

else if (\_timer == 4)

TCCR4B = (TCCR4B & 0b11111000) | prescalarbits;

else if (\_timer == 5)

TCCR5B = (TCCR5B & 0b11111000) | prescalarbits;

#endif

}

// Calculate the toggle count

if (duration > 0)

{

toggle\_count = 2 \* frequency \* duration / 1000;

}

else

{

toggle\_count = -1;

}

// Set the OCR for the given timer,

// set the toggle count,

// then turn on the interrupts

switch (\_timer)

{

#if !defined(\_\_AVR\_ATmega8\_\_)

case 0:

OCR0A = ocr;

timer0\_toggle\_count = toggle\_count;

bitWrite(TIMSK0, OCIE0A, 1);

break;

#endif

case 1:

OCR1A = ocr;

timer1\_toggle\_count = toggle\_count;

bitWrite(TIMSK1, OCIE1A, 1);

break;

case 2:

OCR2A = ocr;

timer2\_toggle\_count = toggle\_count;

bitWrite(TIMSK2, OCIE2A, 1);

break;

#if defined(\_\_AVR\_ATmega1280\_\_)

case 3:

OCR3A = ocr;

timer3\_toggle\_count = toggle\_count;

bitWrite(TIMSK3, OCIE3A, 1);

break;

case 4:

OCR4A = ocr;

timer4\_toggle\_count = toggle\_count;

bitWrite(TIMSK4, OCIE4A, 1);

break;

case 5:

OCR5A = ocr;

timer5\_toggle\_count = toggle\_count;

bitWrite(TIMSK5, OCIE5A, 1);

break;

#endif

}

}

}

void Tone::stop()

{

switch (\_timer)

{

#if !defined(\_\_AVR\_ATmega8\_\_)

case 0:

TIMSK0 &= ~(1 << OCIE0A);

break;

#endif

case 1:

TIMSK1 &= ~(1 << OCIE1A);

break;

case 2:

TIMSK2 &= ~(1 << OCIE2A);

break;

#if defined(\_\_AVR\_ATmega1280\_\_)

case 3:

TIMSK3 &= ~(1 << OCIE3A);

break;

case 4:

TIMSK4 &= ~(1 << OCIE4A);

break;

case 5:

TIMSK5 &= ~(1 << OCIE5A);

break;

#endif

}

digitalWrite(\_pin, 0);

}

bool Tone::isPlaying(void)

{

bool returnvalue = false;

switch (\_timer)

{

#if !defined(\_\_AVR\_ATmega8\_\_)

case 0:

returnvalue = (TIMSK0 & (1 << OCIE0A));

break;

#endif

case 1:

returnvalue = (TIMSK1 & (1 << OCIE1A));

break;

case 2:

returnvalue = (TIMSK2 & (1 << OCIE2A));

break;

#if defined(\_\_AVR\_ATmega1280\_\_)

case 3:

returnvalue = (TIMSK3 & (1 << OCIE3A));

break;

case 4:

returnvalue = (TIMSK4 & (1 << OCIE4A));

break;

case 5:

returnvalue = (TIMSK5 & (1 << OCIE5A));

break;

#endif

}

return returnvalue;

}