/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The MIT License

\*

\* Copyright (c) 2010, LeafLabs, LLC.

\*

\* Permission is hereby granted, free of charge, to any person

\* obtaining a copy of this software and associated documentation

\* files (the "Software"), to deal in the Software without

\* restriction, including without limitation the rights to use, copy,

\* modify, merge, publish, distribute, sublicense, and/or sell copies

\* of the Software, and to permit persons to whom the Software is

\* furnished to do so, subject to the following conditions:

\*

\* The above copyright notice and this permission notice shall be

\* included in all copies or substantial portions of the Software.

\*

\* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,

\* EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF

\* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND

\* NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS

\* BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN

\* ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN

\* CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

\* SOFTWARE.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Arduino srl - www.arduino.org

\* 2017 Feb 23: Edited by Francesco Alessi (alfran) - francesco@arduino.org

\*/

#ifndef \_SERVO\_H\_

#define \_SERVO\_H\_

#include "types.h"

#include "timer.h"

#include "wiring.h" /\* hack for IDE compile \*/

/\*

\* Note on Arduino compatibility:

\*

\* In the Arduino implementation, PWM is done "by hand" in the sense

\* that timer channels are hijacked in groups and an ISR is set which

\* toggles Servo::attach()ed pins using digitalWrite().

\*

\* While this scheme allows any pin to drive a servo, it chews up

\* cycles and complicates the programmer's notion of when a particular

\* timer channel will be in use.

\*

\* This implementation only allows Servo instances to attach() to pins

\* that already have a timer channel associated with them, and just

\* uses analogWrite() to drive the wave.

\*

\* This introduces an incompatibility: while the Arduino

\* implementation of attach() returns the affected channel on success

\* and 0 on failure, this one returns true on success and false on

\* failure.

\*

\* RC Servos expect a pulse every 20 ms. Since periods are set for

\* entire timers, rather than individual channels, attach()ing a Servo

\* to a pin can interfere with other pins associated with the same

\* timer. As always, your board's pin map is your friend.

\*/

// Pin number of unattached pins

#define NOT\_ATTACHED (-1)

#define \_Nbr\_16timers 14 // Number of STM32F469 Timers

#define SERVOS\_PER\_TIMER 4 // Number of timer channels

// Default min/max pulse widths (in microseconds) and angles (in

// degrees). Values chosen for Arduino compatibility. These values

// are part of the public API; DO NOT CHANGE THEM.

#define MIN\_ANGLE 0

#define MAX\_ANGLE 180

#define MIN\_PULSE\_WIDTH 544 // the shortest pulse sent to a servo

#define MAX\_PULSE\_WIDTH 2400 // the longest pulse sent to a servo

/\*\* Class for interfacing with RC servomotors. \*/

class Servo {

public:

/\*\*

\* @brief Construct a new Servo instance.

\*

\* The new instance will not be attached to any pin.

\*/

Servo();

/\*\*

\* @brief Associate this instance with a servomotor whose input is

\* connected to pin.

\*

\* If this instance is already attached to a pin, it will be

\* detached before being attached to the new pin. This function

\* doesn't detach any interrupt attached with the pin's timer

\* channel.

\*

\* @param pin Pin connected to the servo pulse wave input. This

\* pin must be capable of PWM output.

\*

\* @param minPulseWidth Minimum pulse width to write to pin, in

\* microseconds. This will be associated

\* with a minAngle degree angle. Defaults to

\* SERVO\_DEFAULT\_MIN\_PW = 544.

\*

\* @param maxPulseWidth Maximum pulse width to write to pin, in

\* microseconds. This will be associated

\* with a maxAngle degree angle. Defaults to

\* SERVO\_DEFAULT\_MAX\_PW = 2400.

\*

\* @param minAngle Target angle (in degrees) associated with

\* minPulseWidth. Defaults to

\* SERVO\_DEFAULT\_MIN\_ANGLE = 0.

\*

\* @param maxAngle Target angle (in degrees) associated with

\* maxPulseWidth. Defaults to

\* SERVO\_DEFAULT\_MAX\_ANGLE = 180.

\*

\* @sideeffect May set pinMode(pin, PWM).

\*

\* @return true if successful, false when pin doesn't support PWM.

\*/

bool attach(uint8 pin,

uint16 minPulseWidth=MIN\_PULSE\_WIDTH,

uint16 maxPulseWidth=MAX\_PULSE\_WIDTH,

int16 minAngle=MIN\_ANGLE,

int16 maxAngle=MAX\_ANGLE);

/\*\*

\* @brief Stop driving the servo pulse train.

\*

\* If not currently attached to a motor, this function has no effect.

\*

\* @return true if this call did anything, false otherwise.

\*/

bool detach();

/\*\*

\* @brief Set the servomotor target angle.

\*

\* @param angle Target angle, in degrees. If the target angle is

\* outside the range specified at attach() time, it

\* will be clamped to lie in that range.

\*

\* @see Servo::attach()

\*/

void write(int angle);

/\*\*

\* @brief Set the pulse width, in microseconds.

\*

\* @param pulseWidth Pulse width to send to the servomotor, in

\* microseconds. If outside of the range

\* specified at attach() time, it is clamped to

\* lie in that range.

\*

\* @see Servo::attach()

\*/

void writeMicroseconds(uint16 pulseWidth);

/\*\*

\* Get the servomotor's target angle, in degrees. This will

\* lie inside the range specified at attach() time.

\*

\* @see Servo::attach()

\*/

int read() const;

/\*\*

\* Get the current pulse width, in microseconds. This will

\* lie within the range specified at attach() time.

\*

\* @see Servo::attach()

\*/

uint16 readMicroseconds() const;

/\*\*

\* @brief Check if this instance is attached to a servo.

\* @return true if this instance is attached to a servo, false otherwise.

\* @see Servo::attachedPin()

\*/

bool attached() const { return this->pin != NOT\_ATTACHED; }

/\*\*

\* @brief Get the pin this instance is attached to.

\* @return Pin number if currently attached to a pin, NOT\_ATTACHED

\* otherwise.

\* @see Servo::attach()

\*/

int attachedPin() const { return this->pin; }

private:

int16 pin;

uint16 minPW;

uint16 maxPW;

int16 minAngle;

int16 maxAngle;

void resetFields(void);

};

#endif /\* \_SERVO\_H\_ \*/