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

\* 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.

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

#if defined(ARDUINO\_ARCH\_STM32F4)

#include "ServoTimers.h"

#include "boards.h"

#include "io.h"

#include "pwm.h"

#include "math.h"

// 20 millisecond period config. For a 1-based prescaler,

//

// (prescaler \* overflow / CYC\_MSEC) msec = 1 timer cycle = 20 msec

// => prescaler \* overflow = 20 \* CYC\_MSEC

//

// This picks the smallest prescaler that allows an overflow < 2^16.

#define MAX\_OVERFLOW ((1 << 16) - 1)

#define CYC\_MSEC (1000 \* CYCLES\_PER\_MICROSECOND)

#define TAU\_MSEC 20

#define TAU\_USEC (TAU\_MSEC \* 1000)

#define TAU\_CYC (TAU\_MSEC \* CYC\_MSEC)

#define SERVO\_PRESCALER (TAU\_CYC / MAX\_OVERFLOW + 1)

#define SERVO\_OVERFLOW ((uint16)round((double)TAU\_CYC / SERVO\_PRESCALER))

// Unit conversions

#define US\_TO\_COMPARE(us) ((uint16)map((us), 0, TAU\_USEC, 0, SERVO\_OVERFLOW))

#define COMPARE\_TO\_US(c) ((uint32)map((c), 0, SERVO\_OVERFLOW, 0, TAU\_USEC))

#define ANGLE\_TO\_US(a) ((uint16)(map((a), this->minAngle, this->maxAngle, \

this->minPW, this->maxPW)))

#define US\_TO\_ANGLE(us) ((int16)(map((us), this->minPW, this->maxPW, \

this->minAngle, this->maxAngle)))

Servo::Servo() {

this->resetFields();

}

bool Servo::attach(uint8 pin, uint16 minPW, uint16 maxPW, int16 minAngle, int16 maxAngle)

{

// SerialUSB.begin(115200);

// SerialUSB.println(MAX\_OVERFLOW);

timer\_dev \*tdev = PIN\_MAP[pin].timer\_device;

analogWriteResolution(16);

int prescaler = 6;

int overflow = 65400;

int minPW\_correction = 300;

int maxPW\_correction = 300;

pinMode(pin, OUTPUT);

if (tdev == NULL) {

// don't reset any fields or ASSERT(0), to keep driving any

// previously attach()ed servo.

return false;

}

if ( (tdev == TIMER1) || (tdev == TIMER8) || (tdev == TIMER10) || (tdev == TIMER11))

{

prescaler = 54;

overflow = 65400;

minPW\_correction = 40;

maxPW\_correction = 50;

}

if ( (tdev == TIMER2) || (tdev == TIMER3) || (tdev == TIMER4) || (tdev == TIMER5) )

{

prescaler = 6;

overflow = 64285;

minPW\_correction = 370;

maxPW\_correction = 350;

}

if ( (tdev == TIMER6) || (tdev == TIMER7) )

{

prescaler = 6;

overflow = 65400;

minPW\_correction = 0;

maxPW\_correction = 0;

}

if ( (tdev == TIMER9) || (tdev == TIMER12) || (tdev == TIMER13) || (tdev == TIMER14) )

{

prescaler = 6;

overflow = 65400;

minPW\_correction = 30;

maxPW\_correction = 0;

}

if (this->attached()) {

this->detach();

}

this->pin = pin;

this->minPW = (minPW + minPW\_correction);

this->maxPW = (maxPW + maxPW\_correction);

this->minAngle = minAngle;

this->maxAngle = maxAngle;

timer\_pause(tdev);

timer\_set\_prescaler(tdev, prescaler); // prescaler is 1-based

timer\_set\_reload(tdev, overflow);

timer\_generate\_update(tdev);

timer\_resume(tdev);

return true;

}

bool Servo::detach() {

if (!this->attached()) {

return false;

}

timer\_dev \*tdev = PIN\_MAP[this->pin].timer\_device;

uint8 tchan = PIN\_MAP[this->pin].timer\_channel;

timer\_set\_mode(tdev, tchan, TIMER\_DISABLED);

this->resetFields();

return true;

}

void Servo::write(int degrees) {

degrees = constrain(degrees, this->minAngle, this->maxAngle);

this->writeMicroseconds(ANGLE\_TO\_US(degrees));

}

int Servo::read() const {

int a = US\_TO\_ANGLE(this->readMicroseconds());

// map() round-trips in a weird way we mostly correct for here;

// the round-trip is still sometimes off-by-one for write(1) and

// write(179).

return a == this->minAngle || a == this->maxAngle ? a : a + 1;

}

void Servo::writeMicroseconds(uint16 pulseWidth) {

if (!this->attached()) {

ASSERT(0);

return;

}

pulseWidth = constrain(pulseWidth, this->minPW, this->maxPW);

analogWrite(this->pin, US\_TO\_COMPARE(pulseWidth));

}

uint16 Servo::readMicroseconds() const {

if (!this->attached()) {

ASSERT(0);

return 0;

}

stm32\_pin\_info pin\_info = PIN\_MAP[this->pin];

uint16 compare = timer\_get\_compare(pin\_info.timer\_device,

pin\_info.timer\_channel);

return COMPARE\_TO\_US(compare);

}

void Servo::resetFields(void) {

this->pin = NOT\_ATTACHED;

this->minAngle = MIN\_ANGLE;

this->maxAngle = MAX\_ANGLE;

this->minPW = MIN\_PULSE\_WIDTH;

this->maxPW = MAX\_PULSE\_WIDTH;

}

#endif