#if defined(ARDUINO\_ARCH\_MBED)

#include <Arduino.h>

#include <Servo.h>

#include <mbed.h>

#if defined \_\_has\_include

# if \_\_has\_include ("pinDefinitions.h")

# include "pinDefinitions.h"

# endif

#endif

class ServoImpl {

mbed::DigitalOut \*pin;

mbed::Timeout timeout; // calls a callback once when a timeout expires

mbed::Ticker ticker; // calls a callback repeatedly with a timeout

public:

ServoImpl(PinName \_pin) {

pin = new mbed::DigitalOut(\_pin);

}

~ServoImpl() {

ticker.detach();

timeout.detach();

delete pin;

}

void start(uint32\_t duration\_us) {

duration = duration\_us;

ticker.attach(mbed::callback(this, &ServoImpl::call), 0.02f);

}

void call() {

timeout.attach(mbed::callback(this, &ServoImpl::toggle), duration / 1e6);

toggle();

}

void toggle() {

\*pin = !\*pin;

}

int32\_t duration = -1;

};

static ServoImpl\* servos[MAX\_SERVOS]; // static array of servo structures

uint8\_t ServoCount = 0; // the total number of attached servos

#define SERVO\_MIN() (MIN\_PULSE\_WIDTH - this->min) // minimum value in us for this servo

#define SERVO\_MAX() (MAX\_PULSE\_WIDTH - this->max) // maximum value in us for this servo

#define TRIM\_DURATION 15 //callback overhead (35 us) -> 15 us if toggle() is called after starting the timeout

Servo::Servo()

{

if (ServoCount < MAX\_SERVOS) {

this->servoIndex = ServoCount++;

} else {

this->servoIndex = INVALID\_SERVO; // too many servos

}

}

uint8\_t Servo::attach(int pin)

{

return this->attach(pin, MIN\_PULSE\_WIDTH, MAX\_PULSE\_WIDTH);

}

uint8\_t Servo::attach(int pin, int min, int max)

{

pinMode(pin, OUTPUT); // set servo pin to output

servos[this->servoIndex] = new ServoImpl(digitalPinToPinName(pin));

this->min = (MIN\_PULSE\_WIDTH - min);

this->max = (MAX\_PULSE\_WIDTH - max);

return this->servoIndex;

}

void Servo::detach()

{

delete servos[this->servoIndex];

servos[this->servoIndex] = NULL;

}

void Servo::write(int value)

{

// treat values less than 544 as angles in degrees (valid values in microseconds are handled as microseconds)

if (value < MIN\_PULSE\_WIDTH)

{

if (value < 0)

value = 0;

else if (value > 180)

value = 180;

value = map(value, 0, 180, SERVO\_MIN(), SERVO\_MAX());

}

writeMicroseconds(value);

}

void Servo::writeMicroseconds(int value)

{

if (!servos[this->servoIndex]) {

return;

}

// calculate and store the values for the given channel

byte channel = this->servoIndex;

if( (channel < MAX\_SERVOS) ) // ensure channel is valid

{

if (value < SERVO\_MIN()) // ensure pulse width is valid

value = SERVO\_MIN();

else if (value > SERVO\_MAX())

value = SERVO\_MAX();

value = value - TRIM\_DURATION;

if (servos[this->servoIndex]->duration == -1) {

servos[this->servoIndex]->start(value);

}

servos[this->servoIndex]->duration = value;

}

}

int Servo::read() // return the value as degrees

{

return map(readMicroseconds(), SERVO\_MIN(), SERVO\_MAX(), 0, 180);

}

int Servo::readMicroseconds()

{

if (!servos[this->servoIndex]) {

return 0;

}

return servos[this->servoIndex]->duration;

}

bool Servo::attached()

{

return servos[this->servoIndex] != NULL;

}

#endif