#ifndef HX711\_h

#define HX711\_h

#define PD\_SCK\_PORT PORTD // Power Down and Serial Clock Input Port

#define PD\_SCK\_DDR DDRD // Power Down and Serial Clock DDR

#define PD\_SCK\_PIN PD5 // Power Down and Serial Clock Pin

#define PD\_SCK\_SET\_OUTPUT PD\_SCK\_DDR |= (1<<PD\_SCK\_PIN)

#define PD\_SCK\_SET\_HIGH PD\_SCK\_PORT |= (1<<PD\_SCK\_PIN)

#define PD\_SCK\_SET\_LOW PD\_SCK\_PORT &= ~(1<<PD\_SCK\_PIN)

#define DOUT\_PORT PORTD // Serial Data Output Port

#define DOUT\_DDR DDRD // Serial Data Output DDR

#define DOUT\_INPUT PIND // Serial Data Output Input

#define DOUT\_PIN PD6 // Serial Data Output Pin

#define DOUT\_READ (DOUT\_INPUT & (1<<DOUT\_PIN)) // Serial Data Output Read Pin

#define DOUT\_SET\_HIGH DOUT\_PORT |= (1<<DOUT\_PIN)

#define DOUT\_SET\_LOW DOUT\_PORT &= ~(1<<DOUT\_PIN)

#define DOUT\_SET\_INPUT DOUT\_DDR &= ~(1<<DOUT\_PIN); DOUT\_SET\_HIGH

#define DOUT\_SET\_OUTPUT DOUT\_DDR |= (1<<DOUT\_PIN); DOUT\_SET\_LOW

uint8\_t GAIN; // amplification factor

int32\_t OFFSET; // used for tare weight

float SCALE; // used to return weight in grams, kg, ounces, whatever

// define clock and data pin, channel, and gain factor

// channel selection is made by passing the appropriate gain: 128 or 64 for channel A, 32 for channel B

// gain: 128 or 64 for channel A; channel B works with 32 gain factor only

void HX711\_init(uint8\_t gain);

// check if HX711 is ready

// from the datasheet: When output data is not ready for retrieval, digital output pin DOUT is high. Serial clock

// input PD\_SCK should be low. When DOUT goes to low, it indicates data is ready for retrieval.

bool HX711\_is\_ready();

// set the gain factor; takes effect only after a call to read()

// channel A can be set for a 128 or 64 gain; channel B has a fixed 32 gain

// depending on the parameter, the channel is also set to either A or B

void HX711\_set\_gain(uint8\_t gain);

// waits for the chip to be ready and returns a reading

int32\_t HX711\_read();

// returns an average reading; times = how many times to read

int32\_t HX711\_read\_average(uint8\_t times);

// returns (read\_average() - OFFSET), that is the current value without the tare weight; times = how many readings to do

double HX711\_get\_value(uint8\_t times);

// returns get\_value() divided by SCALE, that is the raw value divided by a value obtained via calibration

// times = how many readings to do

float HX711\_get\_units(uint8\_t times);

// set the OFFSET value for tare weight; times = how many times to read the tare value

void HX711\_tare(uint8\_t times);

// set the SCALE value; this value is used to convert the raw data to "human readable" data (measure units)

void HX711\_set\_scale(float scale);

// get the current SCALE

float HX711\_get\_scale();

// set OFFSET, the value that's subtracted from the actual reading (tare weight)

void HX711\_set\_offset(int32\_t offset);

// get the current OFFSET

int32\_t HX711\_get\_offset();

// puts the chip into power down mode

void HX711\_power\_down();

// wakes up the chip after power down mode

void HX711\_power\_up();

// Sends/receives data. Modified from Arduino source

uint8\_t shiftIn(void);

#endif /\* HX711\_h \*/

#include <stdbool.h>

#include <stdint.h>

#include <avr/io.h>

#include <util/delay.h>

#include "HX711.h"

void HX711\_init(uint8\_t gain)

{

PD\_SCK\_SET\_OUTPUT;

DOUT\_SET\_INPUT;

HX711\_set\_gain(gain);

}

bool HX711\_is\_ready()

{

return (DOUT\_INPUT & (1 << DOUT\_PIN)) == 0;

}

void HX711\_set\_gain(uint8\_t gain)

{

switch (gain)

{

case 128: // channel A, gain factor 128

GAIN = 1;

break;

case 64: // channel A, gain factor 64

GAIN = 3;

break;

case 32: // channel B, gain factor 32

GAIN = 2;

break;

}

PD\_SCK\_SET\_LOW;

HX711\_read();

}

int32\_t HX711\_read()

{

// wait for the chip to become ready

while (!HX711\_is\_ready());

uint32\_t value = 0;

uint8\_t data[3] = { 0 };

uint8\_t filler = 0x00;

// pulse the clock pin 24 times to read the data

data[2] = shiftIn();

data[1] = shiftIn();

data[0] = shiftIn();

// set the channel and the gain factor for the next reading using the clock pin

for (uint8\_t i = 0; i < GAIN; i++)

{

PD\_SCK\_SET\_HIGH;

PD\_SCK\_SET\_LOW;

}

// Datasheet indicates the value is returned as a two's complement value

// Flip all the bits

data[2] = ~data[2];

data[1] = ~data[1];

data[0] = ~data[0];

// Replicate the most significant bit to pad out a 32-bit signed integer

if ( data[2] & 0x80 )

{

filler = 0xFF;

} else if ((0x7F == data[2]) && (0xFF == data[1]) && (0xFF == data[0]))

{

filler = 0xFF;

} else

{

filler = 0x00;

}

// Construct a 32-bit signed integer

value = ( (uint32\_t)(filler) << 24

| (uint32\_t)(data[2]) << 16

| (uint32\_t)(data[1]) << 8

| (uint32\_t)(data[0]) );

// ... and add 1

return (int32\_t)(++value);

}

int32\_t HX711\_read\_average(uint8\_t times)

{

int32\_t sum = 0;

for (uint8\_t i = 0; i < times; i++)

{

sum += HX711\_read();

// TODO: See if yield will work | yield();

}

return sum / times;

}

double HX711\_get\_value(uint8\_t times)

{

return HX711\_read\_average(times) - OFFSET;

}

float HX711\_get\_units(uint8\_t times)

{

return HX711\_get\_value(times) / SCALE;

}

void HX711\_tare(uint8\_t times)

{

double sum = HX711\_read\_average(times);

HX711\_set\_offset(sum);

}

void HX711\_set\_scale(float scale)

{

SCALE = scale;

}

float HX711\_get\_scale()

{

return SCALE;

}

void HX711\_set\_offset(int32\_t offset)

{

OFFSET = offset;

}

int32\_t HX711\_get\_offset()

{

return OFFSET;

}

void HX711\_power\_down()

{

PD\_SCK\_SET\_LOW;

PD\_SCK\_SET\_HIGH;

\_delay\_us(70);

}

void HX711\_power\_up()

{

PD\_SCK\_SET\_LOW;

}

uint8\_t shiftIn(void)

{

uint8\_t value = 0;

for (uint8\_t i = 0; i < 8; ++i)

{

PD\_SCK\_SET\_HIGH;

value |= DOUT\_READ << (7 - i);

PD\_SCK\_SET\_LOW;

}

return value;

}

#include <stdint.h>

#include "lib/hx711.h"

static FILE mystdout = FDEV\_SETUP\_STREAM(usart\_putchar\_printf, NULL, \_FDEV\_SETUP\_WRITE);

int32\_t current\_weight;

int main()

{

stdout = &mystdout;

/// Setup

HX711\_init(128);

HX711\_set\_scale(11500.f);

HX711\_tare(10);

char acc\_conv\_out\_x[10]; // Output for dtostrf (+1 for null char)

/// Main Loop

while(1)

{

// Testing HX711 output

current\_weight = HX711\_get\_units(10);

dtostrf(current\_weight, 5, 1, acc\_conv\_out\_x);

printf("Weight: %s\n", acc\_conv\_out\_x);

\_delay\_ms(500);

}

}