#include "Adafruit\_I2CDevice.h"

//#define DEBUG\_SERIAL Serial

/\*!

\* @brief Create an I2C device at a given address

\* @param addr The 7-bit I2C address for the device

\* @param theWire The I2C bus to use, defaults to &Wire

\*/

Adafruit\_I2CDevice::Adafruit\_I2CDevice(uint8\_t addr, TwoWire \*theWire) {

\_addr = addr;

\_wire = theWire;

\_begun = false;

#ifdef ARDUINO\_ARCH\_SAMD

\_maxBufferSize = 250; // as defined in Wire.h's RingBuffer

#elif defined(ESP32)

\_maxBufferSize = I2C\_BUFFER\_LENGTH;

#else

\_maxBufferSize = 32;

#endif

}

/\*!

\* @brief Initializes and does basic address detection

\* @param addr\_detect Whether we should attempt to detect the I2C address

\* with a scan. 99% of sensors/devices don't mind, but once in a while they

\* don't respond well to a scan!

\* @return True if I2C initialized and a device with the addr found

\*/

bool Adafruit\_I2CDevice::begin(bool addr\_detect) {

\_wire->begin();

\_begun = true;

if (addr\_detect) {

return detected();

}

return true;

}

/\*!

\* @brief De-initialize device, turn off the Wire interface

\*/

void Adafruit\_I2CDevice::end(void) {

// Not all port implement Wire::end(), such as

// - ESP8266

// - AVR core without WIRE\_HAS\_END

// - ESP32: end() is implemented since 2.0.1 which is latest at the moment.

// Temporarily disable for now to give time for user to update.

#if !(defined(ESP8266) || \

(defined(ARDUINO\_ARCH\_AVR) && !defined(WIRE\_HAS\_END)) || \

defined(ARDUINO\_ARCH\_ESP32))

\_wire->end();

\_begun = false;

#endif

}

/\*!

\* @brief Scans I2C for the address - note will give a false-positive

\* if there's no pullups on I2C

\* @return True if I2C initialized and a device with the addr found

\*/

bool Adafruit\_I2CDevice::detected(void) {

// Init I2C if not done yet

if (!\_begun && !begin()) {

return false;

}

// A basic scanner, see if it ACK's

\_wire->beginTransmission(\_addr);

#ifdef DEBUG\_SERIAL

DEBUG\_SERIAL.print(F("Address 0x"));

DEBUG\_SERIAL.print(\_addr);

#endif

if (\_wire->endTransmission() == 0) {

#ifdef DEBUG\_SERIAL

DEBUG\_SERIAL.println(F(" Detected"));

#endif

return true;

}

#ifdef DEBUG\_SERIAL

DEBUG\_SERIAL.println(F(" Not detected"));

#endif

return false;

}

/\*!

\* @brief Write a buffer or two to the I2C device. Cannot be more than

\* maxBufferSize() bytes.

\* @param buffer Pointer to buffer of data to write. This is const to

\* ensure the content of this buffer doesn't change.

\* @param len Number of bytes from buffer to write

\* @param prefix\_buffer Pointer to optional array of data to write before

\* buffer. Cannot be more than maxBufferSize() bytes. This is const to

\* ensure the content of this buffer doesn't change.

\* @param prefix\_len Number of bytes from prefix buffer to write

\* @param stop Whether to send an I2C STOP signal on write

\* @return True if write was successful, otherwise false.

\*/

bool Adafruit\_I2CDevice::write(const uint8\_t \*buffer, size\_t len, bool stop,

const uint8\_t \*prefix\_buffer,

size\_t prefix\_len) {

if ((len + prefix\_len) > maxBufferSize()) {

// currently not guaranteed to work if more than 32 bytes!

// we will need to find out if some platforms have larger

// I2C buffer sizes :/

#ifdef DEBUG\_SERIAL

DEBUG\_SERIAL.println(F("\tI2CDevice could not write such a large buffer"));

#endif

return false;

}

\_wire->beginTransmission(\_addr);

// Write the prefix data (usually an address)

if ((prefix\_len != 0) && (prefix\_buffer != nullptr)) {

if (\_wire->write(prefix\_buffer, prefix\_len) != prefix\_len) {

#ifdef DEBUG\_SERIAL

DEBUG\_SERIAL.println(F("\tI2CDevice failed to write"));

#endif

return false;

}

}

// Write the data itself

if (\_wire->write(buffer, len) != len) {

#ifdef DEBUG\_SERIAL

DEBUG\_SERIAL.println(F("\tI2CDevice failed to write"));

#endif

return false;

}

#ifdef DEBUG\_SERIAL

DEBUG\_SERIAL.print(F("\tI2CWRITE @ 0x"));

DEBUG\_SERIAL.print(\_addr, HEX);

DEBUG\_SERIAL.print(F(" :: "));

if ((prefix\_len != 0) && (prefix\_buffer != nullptr)) {

for (uint16\_t i = 0; i < prefix\_len; i++) {

DEBUG\_SERIAL.print(F("0x"));

DEBUG\_SERIAL.print(prefix\_buffer[i], HEX);

DEBUG\_SERIAL.print(F(", "));

}

}

for (uint16\_t i = 0; i < len; i++) {

DEBUG\_SERIAL.print(F("0x"));

DEBUG\_SERIAL.print(buffer[i], HEX);

DEBUG\_SERIAL.print(F(", "));

if (i % 32 == 31) {

DEBUG\_SERIAL.println();

}

}

if (stop) {

DEBUG\_SERIAL.print("\tSTOP");

}

#endif

if (\_wire->endTransmission(stop) == 0) {

#ifdef DEBUG\_SERIAL

DEBUG\_SERIAL.println();

// DEBUG\_SERIAL.println("Sent!");

#endif

return true;

} else {

#ifdef DEBUG\_SERIAL

DEBUG\_SERIAL.println("\tFailed to send!");

#endif

return false;

}

}

/\*!

\* @brief Read from I2C into a buffer from the I2C device.

\* Cannot be more than maxBufferSize() bytes.

\* @param buffer Pointer to buffer of data to read into

\* @param len Number of bytes from buffer to read.

\* @param stop Whether to send an I2C STOP signal on read

\* @return True if read was successful, otherwise false.

\*/

bool Adafruit\_I2CDevice::read(uint8\_t \*buffer, size\_t len, bool stop) {

size\_t pos = 0;

while (pos < len) {

size\_t read\_len =

((len - pos) > maxBufferSize()) ? maxBufferSize() : (len - pos);

bool read\_stop = (pos < (len - read\_len)) ? false : stop;

if (!\_read(buffer + pos, read\_len, read\_stop))

return false;

pos += read\_len;

}

return true;

}

bool Adafruit\_I2CDevice::\_read(uint8\_t \*buffer, size\_t len, bool stop) {

#if defined(TinyWireM\_h)

size\_t recv = \_wire->requestFrom((uint8\_t)\_addr, (uint8\_t)len);

#elif defined(ARDUINO\_ARCH\_MEGAAVR)

size\_t recv = \_wire->requestFrom(\_addr, len, stop);

#else

size\_t recv = \_wire->requestFrom((uint8\_t)\_addr, (uint8\_t)len, (uint8\_t)stop);

#endif

if (recv != len) {

// Not enough data available to fulfill our obligation!

#ifdef DEBUG\_SERIAL

DEBUG\_SERIAL.print(F("\tI2CDevice did not receive enough data: "));

DEBUG\_SERIAL.println(recv);

#endif

return false;

}

for (uint16\_t i = 0; i < len; i++) {

buffer[i] = \_wire->read();

}

#ifdef DEBUG\_SERIAL

DEBUG\_SERIAL.print(F("\tI2CREAD @ 0x"));

DEBUG\_SERIAL.print(\_addr, HEX);

DEBUG\_SERIAL.print(F(" :: "));

for (uint16\_t i = 0; i < len; i++) {

DEBUG\_SERIAL.print(F("0x"));

DEBUG\_SERIAL.print(buffer[i], HEX);

DEBUG\_SERIAL.print(F(", "));

if (len % 32 == 31) {

DEBUG\_SERIAL.println();

}

}

DEBUG\_SERIAL.println();

#endif

return true;

}

/\*!

\* @brief Write some data, then read some data from I2C into another buffer.

\* Cannot be more than maxBufferSize() bytes. The buffers can point to

\* same/overlapping locations.

\* @param write\_buffer Pointer to buffer of data to write from

\* @param write\_len Number of bytes from buffer to write.

\* @param read\_buffer Pointer to buffer of data to read into.

\* @param read\_len Number of bytes from buffer to read.

\* @param stop Whether to send an I2C STOP signal between the write and read

\* @return True if write & read was successful, otherwise false.

\*/

bool Adafruit\_I2CDevice::write\_then\_read(const uint8\_t \*write\_buffer,

size\_t write\_len, uint8\_t \*read\_buffer,

size\_t read\_len, bool stop) {

if (!write(write\_buffer, write\_len, stop)) {

return false;

}

return read(read\_buffer, read\_len);

}

/\*!

\* @brief Returns the 7-bit address of this device

\* @return The 7-bit address of this device

\*/

uint8\_t Adafruit\_I2CDevice::address(void) { return \_addr; }

/\*!

\* @brief Change the I2C clock speed to desired (relies on

\* underlying Wire support!

\* @param desiredclk The desired I2C SCL frequency

\* @return True if this platform supports changing I2C speed.

\* Not necessarily that the speed was achieved!

\*/

bool Adafruit\_I2CDevice::setSpeed(uint32\_t desiredclk) {

#if defined(\_\_AVR\_ATmega328\_\_) || \

defined(\_\_AVR\_ATmega328P\_\_) // fix arduino core set clock

// calculate TWBR correctly

if ((F\_CPU / 18) < desiredclk) {

#ifdef DEBUG\_SERIAL

Serial.println(F("I2C.setSpeed too high."));

#endif

return false;

}

uint32\_t atwbr = ((F\_CPU / desiredclk) - 16) / 2;

if (atwbr > 16320) {

#ifdef DEBUG\_SERIAL

Serial.println(F("I2C.setSpeed too low."));

#endif

return false;

}

if (atwbr <= 255) {

atwbr /= 1;

TWSR = 0x0;

} else if (atwbr <= 1020) {

atwbr /= 4;

TWSR = 0x1;

} else if (atwbr <= 4080) {

atwbr /= 16;

TWSR = 0x2;

} else { // if (atwbr <= 16320)

atwbr /= 64;

TWSR = 0x3;

}

TWBR = atwbr;

#ifdef DEBUG\_SERIAL

Serial.print(F("TWSR prescaler = "));

Serial.println(pow(4, TWSR));

Serial.print(F("TWBR = "));

Serial.println(atwbr);

#endif

return true;

#elif (ARDUINO >= 157) && !defined(ARDUINO\_STM32\_FEATHER) && \

!defined(TinyWireM\_h)

\_wire->setClock(desiredclk);

return true;

#else

(void)desiredclk;

return false;

#endif

}