#include <Adafruit\_BusIO\_Register.h>

#if !defined(SPI\_INTERFACES\_COUNT) || \

(defined(SPI\_INTERFACES\_COUNT) && (SPI\_INTERFACES\_COUNT > 0))

/\*!

\* @brief Create a register we access over an I2C Device (which defines the

\* bus and address)

\* @param i2cdevice The I2CDevice to use for underlying I2C access

\* @param reg\_addr The address pointer value for the I2C/SMBus register, can

\* be 8 or 16 bits

\* @param width The width of the register data itself, defaults to 1 byte

\* @param byteorder The byte order of the register (used when width is > 1),

\* defaults to LSBFIRST

\* @param address\_width The width of the register address itself, defaults

\* to 1 byte

\*/

Adafruit\_BusIO\_Register::Adafruit\_BusIO\_Register(Adafruit\_I2CDevice \*i2cdevice,

uint16\_t reg\_addr,

uint8\_t width,

uint8\_t byteorder,

uint8\_t address\_width) {

\_i2cdevice = i2cdevice;

\_spidevice = nullptr;

\_addrwidth = address\_width;

\_address = reg\_addr;

\_byteorder = byteorder;

\_width = width;

}

/\*!

\* @brief Create a register we access over an SPI Device (which defines the

\* bus and CS pin)

\* @param spidevice The SPIDevice to use for underlying SPI access

\* @param reg\_addr The address pointer value for the SPI register, can

\* be 8 or 16 bits

\* @param type The method we use to read/write data to SPI (which is not

\* as well defined as I2C)

\* @param width The width of the register data itself, defaults to 1 byte

\* @param byteorder The byte order of the register (used when width is > 1),

\* defaults to LSBFIRST

\* @param address\_width The width of the register address itself, defaults

\* to 1 byte

\*/

Adafruit\_BusIO\_Register::Adafruit\_BusIO\_Register(Adafruit\_SPIDevice \*spidevice,

uint16\_t reg\_addr,

Adafruit\_BusIO\_SPIRegType type,

uint8\_t width,

uint8\_t byteorder,

uint8\_t address\_width) {

\_spidevice = spidevice;

\_spiregtype = type;

\_i2cdevice = nullptr;

\_addrwidth = address\_width;

\_address = reg\_addr;

\_byteorder = byteorder;

\_width = width;

}

/\*!

\* @brief Create a register we access over an I2C or SPI Device. This is a

\* handy function because we can pass in nullptr for the unused interface,

\* allowing libraries to mass-define all the registers

\* @param i2cdevice The I2CDevice to use for underlying I2C access, if

\* nullptr we use SPI

\* @param spidevice The SPIDevice to use for underlying SPI access, if

\* nullptr we use I2C

\* @param reg\_addr The address pointer value for the I2C/SMBus/SPI register,

\* can be 8 or 16 bits

\* @param type The method we use to read/write data to SPI (which is not

\* as well defined as I2C)

\* @param width The width of the register data itself, defaults to 1 byte

\* @param byteorder The byte order of the register (used when width is > 1),

\* defaults to LSBFIRST

\* @param address\_width The width of the register address itself, defaults

\* to 1 byte

\*/

Adafruit\_BusIO\_Register::Adafruit\_BusIO\_Register(

Adafruit\_I2CDevice \*i2cdevice, Adafruit\_SPIDevice \*spidevice,

Adafruit\_BusIO\_SPIRegType type, uint16\_t reg\_addr, uint8\_t width,

uint8\_t byteorder, uint8\_t address\_width) {

\_spidevice = spidevice;

\_i2cdevice = i2cdevice;

\_spiregtype = type;

\_addrwidth = address\_width;

\_address = reg\_addr;

\_byteorder = byteorder;

\_width = width;

}

/\*!

\* @brief Write a buffer of data to the register location

\* @param buffer Pointer to data to write

\* @param len Number of bytes to write

\* @return True on successful write (only really useful for I2C as SPI is

\* uncheckable)

\*/

bool Adafruit\_BusIO\_Register::write(uint8\_t \*buffer, uint8\_t len) {

uint8\_t addrbuffer[2] = {(uint8\_t)(\_address & 0xFF),

(uint8\_t)(\_address >> 8)};

if (\_i2cdevice) {

return \_i2cdevice->write(buffer, len, true, addrbuffer, \_addrwidth);

}

if (\_spidevice) {

if (\_spiregtype == ADDRESSED\_OPCODE\_BIT0\_LOW\_TO\_WRITE) {

// very special case!

// pass the special opcode address which we set as the high byte of the

// regaddr

addrbuffer[0] =

(uint8\_t)(\_address >> 8) & ~0x01; // set bottom bit low to write

// the 'actual' reg addr is the second byte then

addrbuffer[1] = (uint8\_t)(\_address & 0xFF);

// the address appears to be a byte longer

return \_spidevice->write(buffer, len, addrbuffer, \_addrwidth + 1);

}

if (\_spiregtype == ADDRBIT8\_HIGH\_TOREAD) {

addrbuffer[0] &= ~0x80;

}

if (\_spiregtype == ADDRBIT8\_HIGH\_TOWRITE) {

addrbuffer[0] |= 0x80;

}

if (\_spiregtype == AD8\_HIGH\_TOREAD\_AD7\_HIGH\_TOINC) {

addrbuffer[0] &= ~0x80;

addrbuffer[0] |= 0x40;

}

return \_spidevice->write(buffer, len, addrbuffer, \_addrwidth);

}

return false;

}

/\*!

\* @brief Write up to 4 bytes of data to the register location

\* @param value Data to write

\* @param numbytes How many bytes from 'value' to write

\* @return True on successful write (only really useful for I2C as SPI is

\* uncheckable)

\*/

bool Adafruit\_BusIO\_Register::write(uint32\_t value, uint8\_t numbytes) {

if (numbytes == 0) {

numbytes = \_width;

}

if (numbytes > 4) {

return false;

}

// store a copy

\_cached = value;

for (int i = 0; i < numbytes; i++) {

if (\_byteorder == LSBFIRST) {

\_buffer[i] = value & 0xFF;

} else {

\_buffer[numbytes - i - 1] = value & 0xFF;

}

value >>= 8;

}

return write(\_buffer, numbytes);

}

/\*!

\* @brief Read data from the register location. This does not do any error

\* checking!

\* @return Returns 0xFFFFFFFF on failure, value otherwise

\*/

uint32\_t Adafruit\_BusIO\_Register::read(void) {

if (!read(\_buffer, \_width)) {

return -1;

}

uint32\_t value = 0;

for (int i = 0; i < \_width; i++) {

value <<= 8;

if (\_byteorder == LSBFIRST) {

value |= \_buffer[\_width - i - 1];

} else {

value |= \_buffer[i];

}

}

return value;

}

/\*!

\* @brief Read cached data from last time we wrote to this register

\* @return Returns 0xFFFFFFFF on failure, value otherwise

\*/

uint32\_t Adafruit\_BusIO\_Register::readCached(void) { return \_cached; }

/\*!

\* @brief Read a buffer of data from the register location

\* @param buffer Pointer to data to read into

\* @param len Number of bytes to read

\* @return True on successful write (only really useful for I2C as SPI is

\* uncheckable)

\*/

bool Adafruit\_BusIO\_Register::read(uint8\_t \*buffer, uint8\_t len) {

uint8\_t addrbuffer[2] = {(uint8\_t)(\_address & 0xFF),

(uint8\_t)(\_address >> 8)};

if (\_i2cdevice) {

return \_i2cdevice->write\_then\_read(addrbuffer, \_addrwidth, buffer, len);

}

if (\_spidevice) {

if (\_spiregtype == ADDRESSED\_OPCODE\_BIT0\_LOW\_TO\_WRITE) {

// very special case!

// pass the special opcode address which we set as the high byte of the

// regaddr

addrbuffer[0] =

(uint8\_t)(\_address >> 8) | 0x01; // set bottom bit high to read

// the 'actual' reg addr is the second byte then

addrbuffer[1] = (uint8\_t)(\_address & 0xFF);

// the address appears to be a byte longer

return \_spidevice->write\_then\_read(addrbuffer, \_addrwidth + 1, buffer,

len);

}

if (\_spiregtype == ADDRBIT8\_HIGH\_TOREAD) {

addrbuffer[0] |= 0x80;

}

if (\_spiregtype == ADDRBIT8\_HIGH\_TOWRITE) {

addrbuffer[0] &= ~0x80;

}

if (\_spiregtype == AD8\_HIGH\_TOREAD\_AD7\_HIGH\_TOINC) {

addrbuffer[0] |= 0x80 | 0x40;

}

return \_spidevice->write\_then\_read(addrbuffer, \_addrwidth, buffer, len);

}

return false;

}

/\*!

\* @brief Read 2 bytes of data from the register location

\* @param value Pointer to uint16\_t variable to read into

\* @return True on successful write (only really useful for I2C as SPI is

\* uncheckable)

\*/

bool Adafruit\_BusIO\_Register::read(uint16\_t \*value) {

if (!read(\_buffer, 2)) {

return false;

}

if (\_byteorder == LSBFIRST) {

\*value = \_buffer[1];

\*value <<= 8;

\*value |= \_buffer[0];

} else {

\*value = \_buffer[0];

\*value <<= 8;

\*value |= \_buffer[1];

}

return true;

}

/\*!

\* @brief Read 1 byte of data from the register location

\* @param value Pointer to uint8\_t variable to read into

\* @return True on successful write (only really useful for I2C as SPI is

\* uncheckable)

\*/

bool Adafruit\_BusIO\_Register::read(uint8\_t \*value) {

if (!read(\_buffer, 1)) {

return false;

}

\*value = \_buffer[0];

return true;

}

/\*!

\* @brief Pretty printer for this register

\* @param s The Stream to print to, defaults to &Serial

\*/

void Adafruit\_BusIO\_Register::print(Stream \*s) {

uint32\_t val = read();

s->print("0x");

s->print(val, HEX);

}

/\*!

\* @brief Pretty printer for this register

\* @param s The Stream to print to, defaults to &Serial

\*/

void Adafruit\_BusIO\_Register::println(Stream \*s) {

print(s);

s->println();

}

/\*!

\* @brief Create a slice of the register that we can address without

\* touching other bits

\* @param reg The Adafruit\_BusIO\_Register which defines the bus/register

\* @param bits The number of bits wide we are slicing

\* @param shift The number of bits that our bit-slice is shifted from LSB

\*/

Adafruit\_BusIO\_RegisterBits::Adafruit\_BusIO\_RegisterBits(

Adafruit\_BusIO\_Register \*reg, uint8\_t bits, uint8\_t shift) {

\_register = reg;

\_bits = bits;

\_shift = shift;

}

/\*!

\* @brief Read 4 bytes of data from the register

\* @return data The 4 bytes to read

\*/

uint32\_t Adafruit\_BusIO\_RegisterBits::read(void) {

uint32\_t val = \_register->read();

val >>= \_shift;

return val & ((1 << (\_bits)) - 1);

}

/\*!

\* @brief Write 4 bytes of data to the register

\* @param data The 4 bytes to write

\* @return True on successful write (only really useful for I2C as SPI is

\* uncheckable)

\*/

bool Adafruit\_BusIO\_RegisterBits::write(uint32\_t data) {

uint32\_t val = \_register->read();

// mask off the data before writing

uint32\_t mask = (1 << (\_bits)) - 1;

data &= mask;

mask <<= \_shift;

val &= ~mask; // remove the current data at that spot

val |= data << \_shift; // and add in the new data

return \_register->write(val, \_register->width());

}

/\*!

\* @brief The width of the register data, helpful for doing calculations

\* @returns The data width used when initializing the register

\*/

uint8\_t Adafruit\_BusIO\_Register::width(void) { return \_width; }

/\*!

\* @brief Set the default width of data

\* @param width the default width of data read from register

\*/

void Adafruit\_BusIO\_Register::setWidth(uint8\_t width) { \_width = width; }

/\*!

\* @brief Set register address

\* @param address the address from register

\*/

void Adafruit\_BusIO\_Register::setAddress(uint16\_t address) {

\_address = address;

}

/\*!

\* @brief Set the width of register address

\* @param address\_width the width for register address

\*/

void Adafruit\_BusIO\_Register::setAddressWidth(uint16\_t address\_width) {

\_addrwidth = address\_width;

}

#endif // SPI exists