Using the $I^2C$ Bus with a Rabbit Microprocessor

This document describes how to use the library, $I^2C\text{.LIB}$, to enable the Rabbit 2000 microprocessor to communicate with $I^2C$ peripheral devices. The library covers the basic $I^2C$ protocol. You must have a Rabbit 2000 or newer Rabbit chip to use the $I^2C$ library. Most Dynamic C libraries are forward compatible; to verify this for a particular library read the comments at the top of the library file.

The term “$I^2C$” comes from IIC, an acronym for Inter IC. The $I^2C$ bus is a popular way for a processor to communicate with peripheral chips on a board. Its simple and flexible nature make it a good choice for low-speed peripherals. There are a number of these that can be used by the Rabbit, including serial EEPROM chips, ADCs, and even more exotic types such as video switches and USB controllers.

Physical Properties of the $I^2C$ Bus

$I^2C$ is a synchronous serial bus that can be shared by several peripheral devices. The physical connection is made up of two wires (SCL and SDA) pulled up to a logic high level by resistors. All nodes on the bus use open-collector style drivers when controlling SCL or SDA. This prevents the possibility of contention between nodes.

![Figure 1.](image)

$I^2C$ Protocol

The master (i.e., the Rabbit) normally controls the clock line (SCL) and determines when a data transfer will occur. Peripherals are not able to interrupt the master. A data bit on the SDA line is normally latched before the clock goes into a high state. The master can signal a start (S) or stop (P) condition by violating this and creating a transition while the clock is high. A falling edge indicates a start, while a rising edge indicates a stop. This is followed by bits being clocked out normally, with the most significant bit first. At the end of each byte, the master generates an extra clock pulse, and the receiver of the byte (slave or master) generates an acknowledge bit (ACK). The ACK is a low state on SDA while SCL is high.

Since SDA is pulled up, an ACK will only be generated if the receiver intentionally pulls the line low.
Which Parallel Port to Use

There are 5 parallel ports on the Rabbit 2000. The I²C library uses port D because it has open drain capabilities, which makes clock stretching easy to detect. Clock stretching is when a slave device needs more time to prepare to send a byte of data and so pulls SCL low to indicate that it is not yet ready.

Port D is hardcoded in i2c.lib with #ifndef directives. To use a different parallel port you must #define a few things in your application code, as well as change the function i2c_init(). Refer to i2c.lib, and specifically the comment “Define these to change basic bit handling,” to see what needs to be done.

I²C Library API

The functions in I2C.LIB handle the generic aspects of an I²C interface. Drivers for specific I²C peripheral devices can easily be built with these functions. The basic operations for a Rabbit I²C master are:

- **Initialize the I²C interface pins**
  ```
  i2c_init() // Sets up the SCL and SDA port pins for open-drain output
  // Also initializes delay constant
  ```

- **Send a start condition**
  ```
  i2c_start_tx() // initiates I²C transmission by sending S (START)
  i2c_startw_tx() // initiates I²C transmission by sending S
  // inserts delay after S pulse.
  ```

- **Send a byte of data**
  ```
  i2c_write_char() // Sends 8 bits to slave
  i2c_wr_wait() // Retries char write until slave responds
  ```

- **Listen for an Acknowledgement**
  ```
  i2c_check_ack() // Checks if slave pulls data low for ACK on clock pulse
  ```

- **Receive a byte of data**
  ```
  i2c_read_char() // Reads 8 bits from slave
  ```

- **Send an Acknowledgement**
  ```
  i2c_send_ack() // Sends ACK sequence to slave
  i2c_send_nak() // Sends NAK sequence to slave
  ```

- **Send a stop condition**
  ```
  i2c_stop_tx() // Sends P (STOP) to slave
  ```
These actions are combined in driver functions to communicate with a specific I²C device. For additional information on this API, refer to the Dynamic C Function Reference Manual or use the “Function Lookup/Insert” feature on Dynamic C’s Help menu.

**Device Specific Functions**

To write a set of functions for a given device, you will need to consult its datasheet and write functions for each of the operations you wish to perform. This is simple, since the master’s tasks breaks down into a series of byte writes and byte reads. Examples of higher-level API functions written for specific devices are in I2C_DEVICES.LIB. This library contains implementations for a 24LC16 EEPROM and a MAX518 DAC.

After writing a byte, the master usually listens for an acknowledgement bit and acts appropriately depending on if one is received or not. Not receiving an ACK from the slave does not always indicate an error. Several types of I²C devices stop acknowledging while they are busy to indicate that they are not ready. EEPROMs doing a write operation and ADCs performing a conversion are good examples of this.

If the I²C device only receives commands, then it is a simple matter sending it a Start, writing the command bytes to it, listening for an ACK after each one, and finally ending with a Stop. If the device sends data back to the master, the operation is more complex.

In this case, the master starts the transfer by sending a Start followed by one or more bytes. The slave should ACK each of these. At some point the master will begin reading bytes from the device and sending an ACK bit after each one. The master usually ends the transfer by not acknowledging the last byte and then sending a Stop condition.

Note that even while reading, the master still controls the clock (SCL), but the device is controlling SDA.

**Sample Programs**

Sample programs are available in the directory where Dynamic C was installed, in the folder Samples\I2C. The program, i2c_test.c, writes a string to the beginning of the memory space of a Microchip 24LC16 serial EEPROM, and then reads it back. The program, i2c_dac_sample.c, tests the connection of the Rabbit to a MAX517/518/519 chip. This sample will cycle through DAC values zero through 255, creating a sawtooth output.

**Summary**

Communicating with devices over an I²C bus requires a tailored implementation for each device type. After reading this document and looking at the sample programs, you should be able to create an implementation for whatever device you will be using. The best resource for further information is the datasheet for the chip you are interfacing to the Rabbit. The better datasheets include an overview of I²C in addition to specifics about communicating with the chip. The I²C specification was developed by Philips Semiconductor and is available on their website: www.semiconductors.philips.com.