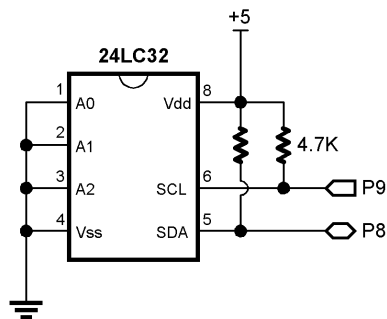




Experiment #32: I²C Communications

This experiment demonstrates the BASIC Stamp's ability to communicate with other devices through the use of the popular Philips I²C protocol. The experiment uses this protocol to write and read data to a serial EEPROM and the low-level I²C routines can be used to communicate with any I²C device.

Building The Circuit



```

=====
'
' File..... Ex32 - 24LC32.BS2
' Purpose... 24LC32 control via I2C
' Author.... Parallax
' E-mail.... stamptech@parallaxinc.com
' Started...
' Updated... 01 MAY 2002
'
' {$STAMP BS2}
'
=====

' -----
' Program Description
' -----

' This program demonstrates essential I2C routines and communication with the
' Microchip 24LC32 serial EEPROM.
'
' The connections for this program conform to the BS2p I2CIN and I2COUT

```

Experiment #32: I²C Communications

```
' commands. Use this program for the BS2, BS2e or BS2sx. There is a separate
' program for the BS2p.
```

```
' -----
' I/O Definitions
' -----
```

```
SDA          CON      8          ' I2C serial data line
SCL          CON      9          ' I2C serial clock line
```

```
' -----
' Constants
' -----
```

```
DevType      CON      %1010 << 4    ' device type
DevAddr      CON      %000 << 1     ' address = %000 -> %111
Wr2432       CON      DevType | DevAddr | 0 ' write to 24LC32
Rd2432       CON      DevType | DevAddr | 1 ' read from 24LC32

ACK          CON      0             ' acknowledge bit
NAK          CON      1             ' no ack bit

CrsrXY       CON      2             ' DEBUG Position Control
```

```
' -----
' Variables
' -----
```

```
i2cSDA       VAR      Nib           ' I2C serial data pin
i2cData      VAR      Byte          ' data to/from device
i2cWork      VAR      Byte          ' work byte for TX routine
i2cAck       VAR      Bit           ' ACK bit from device

eeAddr       VAR      Word          ' address: 0 - 4095
test         VAR      Nib           '
outVal       VAR      Byte          ' output to EEPROM
inVal        VAR      Byte          ' input from EEPROM
```

```
' -----
' Initialization
' -----
```

```
Initialize:
```

Experiment #32: I²C Communications

```
PAUSE 250                                ' let DEBUG open
DEBUG CLS, "24LC32 Demo", CR, CR          ' setup output screen
DEBUG "Address... ", CR
DEBUG "Output.... ", CR
DEBUG "Input..... ", CR

i2cSDA = SDA                              ' define SDA pin

' -----
' Program Code
' -----

Main:
  FOR eeAddr = 0 TO 4095                   ' test all locations
    DEBUG CrsrXY, 11, 2, DEC eeAddr, "  "
    FOR test = 0 TO 3                     ' use four patterns
      LOOKUP test, [$FF, $AA, $55, $00], outVal
      DEBUG CrsrXY, 11, 3, IHEX2 outVal
      i2cData = outVal
      GOSUB Write_Byte
      PAUSE 10
      GOSUB Read_Byte
      inVal = i2cData
      DEBUG CrsrXY, 11, 4, IHEX2 inVal, "  "
      IF (inVal <> outVal) THEN Bad_Addr
      DEBUG "Pass "
      GOTO Next_Addr
    NEXT test
  NEXT eeAddr

Bad_Addr:
  DEBUG "Fail "

Next_Addr:
  PAUSE 50
  NEXT
NEXT

DEBUG CR, CR, "Done!"
END

' -----
' Subroutines
' -----

' Byte to be written is passed in i2cData
' -- address passed in eeAddr
```

Experiment #32: I²C Communications

```
Write_Byte:
  GOSUB I2C_Start           ' send Start
  i2cWork = Wr2432         ' send write command
  GOSUB I2C_TX_Byte
  IF (i2cAck = NAK) THEN Write_Byte ' wait until not busy
  i2cWork = eeAddr / 256   ' send word address (1)
  GOSUB I2C_TX_Byte
  i2cWork = eeAddr // 256 ' send word address (0)
  GOSUB I2C_TX_Byte
  i2cWork = i2cData        ' send data
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN

' Byte read is returned in i2cData
' -- address passed in eeAddr

Read_Byte:
  GOSUB I2C_Start           ' send Start
  i2cWork = Wr2432         ' send write command
  GOSUB I2C_TX_Byte
  IF (i2cAck = NAK) THEN Write_Byte ' wait until not busy
  i2cWork = eeAddr / 256   ' send word address (1)
  GOSUB I2C_TX_Byte
  i2cWork = eeAddr // 256 ' send word address (0)
  GOSUB I2C_TX_Byte
  GOSUB I2C_Start
  i2cWork = Rd2432         ' send read command
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN

' -----
' Low Level I2C Subroutines
' -----

' --- Start ---

I2C_Start:
  INPUT i2cSDA           ' I2C start bit sequence
  INPUT SCL
  LOW i2cSDA             ' SDA -> low while SCL high
```

Experiment #32: I²C Communications

```
Clock_Hold:
  IF (Ins.LowBit(SCL) = 0) THEN Clock_Hold      ' device ready?
  RETURN

' --- Transmit ---

I2C_TX_Byte:
  SHIFTOUT i2cSDA, SCL, MSBFIRST, [i2cWork\8]  ' send byte to device
  SHIF TIN i2cSDA, SCL, MSBP RE, [i2cAck\1]    ' get acknowledge bit
  RETURN

' --- Receive ---

I2C_RX_Byte_Nak:
  i2cAck = NAK                                ' no ACK = high
  GOTO I2C_RX

I2C_RX_Byte:
  i2cAck = ACK                                ' ACK = low

I2C_RX:
  SHIF TIN i2cSDA, SCL, MSBP RE, [i2cWork\8]  ' get byte from device
  SHIF TOU T i2cSDA, SCL, LSBFIR ST, [i2cAck\1] ' send ack or nak
  RETURN

' --- Stop ---

I2C_Stop:                                     ' I2C stop bit sequence
  LOW i2cSDA
  INPUT SCL
  INPUT i2cSDA                                ' SDA --> high while SCL high
  RETURN
```

Behind the Scenes

The I²C-bus is a two-wire, synchronous bus that uses a Master-Slave relationship between components. The Master initiates communication with the Slave and is responsible for generating the clock signal. If requested to do so, the Slave can send data back to the Master. This means the data pin (SDA) is bi-directional and the clock pin (SCL) is [usually] controlled only by the Master.

Experiment #32: I²C Communications

The transfer of data between the Master and Slave works like this:

Master sending data

- Master initiates transfer
- Master addresses Slave
- Master sends data to Slave
- Master terminates transfer

Master receiving data

- Master initiates transfer
- Master addresses Slave
- Master receives data from Slave
- Master terminates transfer

The I²C specification actually allows for multiple Masters to exist on a common bus and provides a method for arbitrating between them. That's a bit beyond the scope of what we need to do so we're going to keep things simple. In our setup, the BS2 (or BS2e or BS2sx) will be the Master and anything connected to it will be a Slave.

You'll notice in I²C schematics that the SDA and SCL lines are pulled up to Vdd (usually through 4.7K). The specification calls for device bus pins to be open drain. To put a high on either line, the associated bus pin is made an input (floats) and the pull-up takes the line to Vdd. To make a line low, the bus pin pulls it to Vss (ground).

This scheme is designed to protect devices on the bus from a short to ground. Since neither line is driven high, there is no danger. We're going to cheat a bit. Instead of writing code to pull a line low or release it (certainly possible – I did it), we're going to use **SHIFTOUT** and **SHIFIN** to move data back and forth. Using **SHIFTOUT** and **SHIFIN** is faster and saves precious code space. If you're concerned about a bus short damaging the Stamp's SDA or SCL pins during **SHIFTOUT** and **SHIFIN**, you can protect each of them with a 220 ohm resistor. I've been careful with my wiring and code and haven't found this necessary.

Low Level I²C Code

At its lowest level, the I²C Master needs to do four things:

- Generate a Start condition
- Transmit 8-bit data to the Slave
- Receive 8-bit data from Slave – with or without Acknowledge
- Generate Stop condition

A Start condition is defined as a HIGH to LOW transition on the SDA line while the SCL line is HIGH. All transmissions begin with a Start condition. A Stop condition is defined as a LOW to HIGH transition of the SDA line while the clock line is HIGH. A Stop condition terminates a transfer and can be used to abort it as well.

There is a brief period when the Slave can take control of the SCL line. If a Slave is not ready to transmit or receive data, it can hold the SCL line low after the Start condition. The Master can monitor this to wait for the Slave to be ready. At the speed of the BS2, monitoring the clock line usually isn't necessary but I've built the clock-hold test into the I2C_Start subroutine just to be safe.

Data is transferred eight bits at a time, sending the MSB first. After each byte, the I²C specification calls for the receiving device to acknowledge the transmission by bringing the bus low for the ninth clock. The exception to this is when the Master is the receiver and is receiving the final byte from the Slave. In this case, there is no Acknowledge bit sent from Master to Slave.

Sending and receiving data from a specific slave always requires a Start condition, sending the Slave address and finally, the Stop condition. What happens between the Slave address and the Stop are dependent on the device and what we're doing.

What you'll need to do is get the data sheet for the I²C device you want to connect to. I have found, without exception, that data sheets for I²C-compatible parts have very clear protocol definitions – usually in graphic form – that makes implementing our low-level I²C routines very simple.

The experiment uses the low-level I²C routines to implement the **Write_Byte** and **Read_Byte** routines. The sequence for these routines was lifted right from the 24LC32 data sheet. Notice that each routine begins with an I²C Start condition and is terminated with the Stop condition. The code in between sends the device command/type code, the address to deal with and then actually deals with (writes or reads) the data. While this takes a few lines of code, it is actually very straightforward.

Experiment #32: I²C Communications

Most I²C routines follow a very similar structure; varying only in the internal address and for a few devices, the way the device code is transmitted (there are a few devices that carry an address setting in the device code byte).

Challenge

From the hundreds of I²C devices available, pick one that will be most useful for your projects and write the high-level code necessary to communicate with it.

Striking Out On Your Own

Going Beyond The Box

By now, your appetite for BASIC Stamp projects has probably grown well beyond what you ever expected. So where do you turn now? Don't worry, there are many BASIC Stamp and related resources available, both in print and on the Internet. Here's a list to get you started:

Books & Magazines

- *Microcontroller Application Cookbook* By Matt Gilliland
- *Microcontroller Projects with BASIC Stamps* By Al Williams
- *Programming and Customizing the BASIC Stamp Computer* By Scott Edwards
- *BASIC Stamp* By Claus Kühnel and Klaus Zahnert
- *Getting Started In Electronics* By Forrest Mims
- *Engineer's Notebook* By Forrest Mims
- *Nuts & Volts* Magazine "Stamp Applications" column

Internet Sites

www.parallaxinc.com

www.stampsinclass.com

www.al-williams.com/awce/index.htm

www.seetron.com

www.hth.com/losa

www.emesystems.com/BS2index.htm

Parallax main site

Parallax educational site

Al Williams web site

Scott Edwards Electronics web site

List of Stamp Applications – great idea source

Tracy Allen's Stamp resources – very technical



**Appendix A:
BASIC Stamp II Manual Version 2.0c**

Pages 198-344 of the BASIC Stamp Manual are included in this appendix. The entire manual (and future updates) is available for purchase or download from www.parallaxinc.com.