BASCOM-AVR I2C SLAVE Library

The I2C-Slave library is intended to create I2C slave chips.
You can control these slave chips with the BASCOM I2C master statements like I2CINIT, I2CSEND, I2CRECEIVE, I2CWBYTE, etc. Please consult the BASCOM Help file for using I2C in master mode.

Before you begin
Copy all LIB and LBX files to the BASCOM-AVR\LIB folder.
The LIB files contain the ASM source. The LBX files contain the compiled ASM source.
The following libraries are available :

- i2cslave.lib (for chips without TWI)
- i2c_twi-slave (for chips with TWI)
- i2c_twi-slave-acknack (for chips with TWI)
- xmega-twi-slave (for XMEGA)
- I2C_USI_SLAVE.LIB (for chips with USI)

TWI is Atmels name for I2C. It is exactly the same.
There are various libraries. When using Xmega, you should use the XMEGA-TWI-SLAVE library. This library works in interrupt mode.

When using a processor with USI interface you should use I2C_USI_SLAVE.LIB. This library works in interrupt mode.

When using a processor with TWI interface you should use i2c_twi-slave library. It uses the TWI in interrupt mode and uses little resources compared to the i2cslave.lib

The i2cslave.lib is a software i2c slave library. It can only be used on a number of processors.

All libraries work in a similar way. They wait for a start condition and check the slave address and read/write bit.

SOFTWARE EMULATED SLAVE

Slave address
Every I2C device must have a slave address so it can be addressed by the master I2C routines.
When you write to an I2C-slave chip the least significant bit (bit0) is used to specify if we want to read from the chip or that we want to write to the chip.
When you specify the slave address, do not use bit 0 in the address!
Bascom uses the 8 bit address for I2C.
Manufacturers often specify the 7 bit address. For example address 7 bit address **&B**1 becomes **&B**10 in bascom since bit 0 is used for read/write. When you shift the 7-bit address from the manufacturer 1 place to the left you will get the 8-bit address that bascom uses.
When creating I2C slave devices you do not need to care about it as long as you do not use bit 0. This means that your slave address is always an even number like 0, 2, 4, etc.

For example a PCF8574 has address &H40. To write to the chip use &H40, to read from the chip, use &H41. When emulating a PCF8574 we would specify address &H40.

(c) MCS Electronics

The I2C slave routines in bascom are simple to use. You use CONFIG I2CSLAVE/TWISLAVE/TWICSLAVE/USE to setup the slave.

Example for software emulated I2C slave:
Use the CONFIG statement to specify the slave address:

```
Config I2cslave = &B01000000                    ' same as &H40
```

Optional use : CONFIG I2CSLAVE = address, INT= int , TIMER = tmr
Where INT is INT0, INT1 etc. and TIMER is TIMER0, TIMER1 etc.
When using other interrupts or timers, you need to change the library source!

This directive will enable the global interrupt.

After you have configured the slave address, you can insert your code.
A do-loop would be best:

```
Do
   ' your code here
Loop
```

This is a simple never ending loop. You can use a GOTO with a label or a While Wend loop too but ensure that the program will never end.

After your main program code you need to insert two labels:

When the master needs to read a byte, the following label is always called
You must put the data you want to send to the master in variable _a1 which is register R16

```
I2c_master_needs_data:
  'when your code is short, you need to put in a waitms statement
  'Take in mind that during this routine, a wait state is active and the master will wait
  'After the return, the waitstate is ended
  Config Portb = Input                          ' make it an input
  _a1 = Pinb                                    ' Get input from portB and assign it
Return
```

When the master writes a byte, the following label is always called
It is your task to retrieve variable _A1 and do something with it
_A1 is register R16 that could be destroyed/altered by BASIC statements
For that reason it is important that you first save this variable

```
I2c_master_has_data:
  'when your code is short, you need to put in a waitms statement
  'Take in mind that during this routine, a wait state is active and the master will wait
  'After the return, the waitstate is ended

  Bfake = _a1                                   ' this is not needed but it shows how you can store
_A1 in a byte
  'after you have stored the received data into bFake, you can alter R16
```

```
 Config Portb = Output                ' make it an output since it could be an input
 Portb = _a1                          'assign _A1 (R16)
Return
```

The additional code inside the labels is copied from the i2cslave_pcf8574.bas example.

When the slave address is asked to send data, the label I2C_MASTER_NEEDS_DATA is called by the library. It is you task to assign a value to _A1. This value will be received by the master read code.

When the slave library is instructed that data will follow, the label I2C_MASTER_HAS_DATA will be called by the library. In this routine you can retrieve the data by reading _A1.
Assign _A1 to a variable before you use BASIC commands since some commands might alter _A1. _A is register R16.

Do not forget to use 4K7 pull up resistors on the SCL and SDA lines.

The SCL and SDA pins are fixed since they use INT0 and TIMER/COUNTER0.

Chip	SCL	SDA
90S2313	PORTD.4	PORTD.2
90S2333	PORTD.4	PORTD.2
90S4433	PORTD.4	PORTD.2
Mega8	PORTD.4	PORTD.2
90S2323	PORTB.2	PORTB.1
90S2343	PORTB.2	PORTB.1

BASCOM sample project.

```
'-----------------------------------------------------------------------
'                            I2C SLAVE LIBRARY DEMO
'                                PCF8574 emulator
'                           (c) 2002 MCS Electronics
'-----------------------------------------------------------------------
'This program shows how you could use the I2C slave library to create a
PCF8574
'The PCF8574 is an IO extender chip that has 8 pins.
'The pins can be set to a logic level by writing the address followed by a
value
'In order to read from the pins you need to make them '1' first

'This program uses a AT90S2313, PORTB is used as the PCF8574 PORT
'The slave library needs INT0 and TIMER0 in order to work.
'SCL is PORTD.4 (T0)
'SDA is PORTD.2 (INT0)
'Use 10K pull up resistors for both SCL and SDA

'The Slave library will only work for chips that have T0 and INT0 connected
to the same PORT.
'These chips are : 2313,2323, 2333,2343,4433,tiny22, tiny12,tiny15, M8
'The other chips have build in hardware I2C(slave) support.



'specify the XTAL connected to the chip
$crystal = 3684000

'specify the used chip
$regfile = "2313def.dat"

'specify the slave address. This is &H40 for the PCF8574
'You always need to specify the address used for write. In this case &H40 ,

'The config i2cslave command will enable the global interrupt enable flag !
Config I2cslave = &B01000000                                  ' same as &H40

'A byte named _i2c_slave_address_received is generated by the compiler.
'This byte will hold the received address.

'A byte named _i2c_slave_address is generated by the compiler.
'This byte must be assigned with the slave address of your choice

'the following constants will be created that are used by the slave
library:

' _i2c_pinmask = &H14
' _i2c_slave_port = Portd
' _i2c_slave_pin = Pind
' _i2c_slave_ddr = Ddrd
' _i2c_slave_scl = 4
' _i2c_slave_sda = 2

'These values are adjusted automatic depending on the selected chip.
'You do not need to worry about it, only provided as additional info

'by default the PCF8574 port is set to input
Config Portb = Input
Portb = 255                                                  'all pins high
```

```
by default

'DIM a byte that is not needed but shows how you can store/write the I2C
DATA
Dim Bfake As Byte


'empty loop
 Do
   ' you could put your other program code here
   'In any case, do not use END since it will disable interrupts

 Loop


'here you can write your other program code
'But do not forget, do not use END. Use STOP when needed

'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
'             The following labels are called from the slave library
'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

'When the master wants to read a byte, the following label is always called
'You must put the data you want to send to the master in variable _a1 which
is register R16
I2c_master_needs_data:
  'when your code is short, you need to put in a waitms statement
  'Take in mind that during this routine, a wait state is active and the
master will wait
  'After the return, the waitstate is ended
  Config Portb = Input              ' make it an input
  _a1 = Pinb                        ' Get input from portB and assign it
Return


'When the master writes a byte, the following label is allways called
'It is your task to retrieve variable _A1 and do something with it
'_A1 is register R16 that could be destroyed/altered by BASIC statements
'For that reason it is important that you first save this variable

I2c_master_has_data:
  'when your code is short, you need to put in a waitms statement
  'Take in mind that during this routine, a wait state is active and the
master will wait
  'After the return, the waitstate is ended

  Bfake = _a1 ' this is not needed but it shows how you can store _A1 in a
byte
  'after you have stored the received data into bFake, you can alter R16
  Config Portb = Output ' make it an output since it could be an input
  Portb = _a1           'assign _A1 (R16)
Return


'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

'You could simply extend this sample so it will use 3 pins of PORT D for
the address selection
'For example portD.1 , portd.2 and portD.3 could be used for the address
selection
'Then after the CONFIG I2CSLAVE = &H40 statement, you can put code like:
```

```
'Dim switches as Byte    ' dim byte
'switches = PIND          ' get dip switch value
'switches = switches and &H1110 ' we only need the lower nibble without
'_i2c_slave_address = &H40 + switches ' set the proper address
```

The following information is available in the bascom-help as well.

Syntax

CONFIG I2CSLAVE = address , INT = interrupt , TIMER = tmr
(This function is part of the I2C-Slave library. This is an add-on library that is not included in Bascom-AVR by default. It is a commercial add on library. It is available from MCS Electronics)

Remarks

Address	The slave address you want to assign to the I2C slave chip. This is an address that must be even like &H60. So &H61 cannot be used. I2C uses a 7 bit address from bit 1 to bit 7. Bit 0 is used to specify a read/write operation. In BASCOM the byte transmission address is used for I2C. This means that an I2C address of 1 becomes &B10 = 2. And we say the address is 2. This is done so you can copy the address from the data sheets which are in the same format in most cases.
Interrupt	The interrupt that must be used. This is INT0 by default.
Tmr	The timer that must be used. This is TIMER0 by default.

The library was written for TIMER0 and INT0.

While the interrupt can be specified, you need to change the library code when you use a non-default interrupt. For example when you like to use INT1 instead of the default INT0.

The same applies to the TIMER. You need to change the library when you like to use another timer.

You can not use these interrupts yourself. It also means that the SCL and SDA pins are fixed.

CONFIG I2CSLAVE will enable the global interrupts.

Timer0 and INT0 Pin's of Various AVR's

The I2C slave routines use the TIMER0 and INT0.

The following table lists the pins for the various chips

Chip	SCL	SDA
AT90S1200	PORTD.4	PORTD.2
AT90S2313	PORTD.4	PORTD.2
AT90S2323	PORTB.2	PORTB.1
AT90S2333	PORTD.4	PORTD.2
AT90S2343	PORTB.2	PORTB.1
AT90S4433	PORTD.4	PORTD.2
ATTINY22	PORTB.2	PORTB.1
ATTINY13	PORTB.2	PORTB.1

ATTINY2313	PORTD.4	PORTD.2
ATMEGA1280	PORTD.7	PORTD.0
ATMEGA128CAN	PORTD.7	PORTD.0
ATMEGA168	PORTD.4	PORTD.2
ATMEGA2560	PORTD.7	PORTD.0
ATMEGA2561	PORTD.7	PORTD.0
ATMEGA48	PORTD.4	PORTD.2
ATMEGA88	PORTD.4	PORTD.2
ATMEGA8	PORTD.4	PORTD.2

After you have configured the slave address, you can insert your code.

A do-loop would be best:

```
Do
  ' your code here
Loop
```

After your main program you need to insert two labels with a return:

When the master needs to read a byte, the following label is always called.
You must put the data you want to send to the master in variable _a1 which is register R16

```
I2c_master_needs_data:
'when your code is short, you need to put in a waitms statement
'Take in mind that during this routine, a wait state is active and the
master will wait
'After the return, the waitstate is ended
Config Portb = Input ' make it an input

_a1 = Pinb ' Get input from portB and assign it
Return
```

When the master writes a byte, the following label is always called.
It is your task to retrieve variable _A1 and do something with it
_A1 is register R16 that could be destroyed/altered by BASIC statements
For that reason it is important that you first save this variable.

```
I2c_master_has_data:
  'when your code is short, you need to put in a waitms statement
  'Take in mind that during this routine, a wait state is active and the
master will wait
  'After the return, the waitstate is ended

  Bfake = _a1                    ' this is not needed but it shows how you
can store _A1 in a byte
  'after you have stored the received data into bFake, you can alter R16
  Config Portb = Output       ' make it an output since it could be an input
  Portb = _a1                    'assign _A1 (R16)
Return
```

See Also

CONFIG TWI , CONFIG TWISLAVE, I2C TWI Slave

Debugging Hint's

If you encounter a problem first check:

- Do you use the correct Pin's for SDA and SCL ?
- Pull-up Resistor from SDA and SCL to Vcc ?
- Try to reduce clockrate from I2C Master
- Try to use **waitms XX** between the **I2CWBYTE** in the I2C Master AVR
- Try to reduce code in the interrupt routine

Example

```
'-----------------------------------------------------------------------
---------------
'name                     : i2c_pcf8574.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : shows how you could use the I2C slave library
to create a PCF8574
'micro                    : AT90S2313
'suited for demo          : NO, ADDON NEEDED
'commercial addon needed  : yes
'-----------------------------------------------------------------------
---------------

$regfile = "2313def.dat"                                    ' specify the
used micro
$crystal = 3684000                                          ' used crystal
frequency
$baud = 19200                                               ' use baud rate
$hwstack = 32                                               ' default use
32 for the hardware stack
$swstack = 10                                               ' default use
10 for the SW stack
$framesize = 40                                             ' default use
40 for the frame space

'This program shows how you could use the I2C slave library to create a
PCF8574
'The PCF8574 is an IO extender chip that has 8 pins.
'The pins can be set to a logic level by writing the address followed by a
value
'In order to read from the pins you need to make them '1' first

'This program uses a AT90S2313, PORTB is used as the PCF8574 PORT
'The slave library needs INT0 and TIMER0 in order to work.
'SCL is PORTD.4 (T0)
'SDA is PORTD.2 (INT0)
'Use 10K pull up resistors for both SCL and SDA

'The Slave library will only work for chips that have T0 and INT0 connected
to the same PORT.
'These chips are : 2313,2323, 2333,2343,4433,tiny22, tiny12,tiny15, M8
'The other chips have build in hardware I2C(slave) support.

'specify the slave address. This is &H40 for the PCF8574
'You always need to specify the address used for write. In this case &H40 ,

'The config i2cslave command will enable the global interrupt enable flag !
Config I2cslave = &B01000000                                ' same as &H40
'Config I2cslave = &H40 , Int = Int0 , Timer = Timer0
'A byte named _i2c_slave_address_received is generated by the compiler.
```

```
'This byte will hold the received address.

'A byte named _i2c_slave_address is generated by the compiler.
'This byte must be assigned with the slave address of your choice

'the following constants will be created that are used by the slave
library:

' _i2c_pinmask = &H14
' _i2c_slave_port = Portd
' _i2c_slave_pin = Pind
' _i2c_slave_ddr = Ddrd
' _i2c_slave_scl = 4
' _i2c_slave_sda = 2

'These values are adjusted automatic depending on the selected chip.
'You do not need to worry about it, only provided as additional info

'by default the PCF8574 port is set to input
Config Portb = Input
Portb = 255                                                 'all pins high
by default

'DIM a byte that is not needed but shows how you can store/write the I2C
DATA
Dim Bfake As Byte


'empty loop
Do
   ' you could put your other program code here
   'In any case, do not use END since it will disable interrupts

Loop


'here you can write your other program code
'But do not forget, do not use END. Use STOP when needed

'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!
'             The following labels are called from the slave library
'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!

'When the master wants to read a byte, the following label is allways
called
'You must put the data you want to send to the master in variable _a1 which
is register R16
I2c_master_needs_data:
  'when your code is short, you need to put in a waitms statement
  'Take in mind that during this routine, a wait state is active and the
master will wait
  'After the return, the waitstate is ended
  Config Portb = Input                                      ' make it an
input
  _a1 = Pinb                                                ' Get input
from portB and assign it
Return
```

```
'When the master writes a byte, the following label is always called
'It is your task to retrieve variable _A1 and do something with it
'_A1 is register R16 that could be destroyed/altered by BASIC statements
'For that reason it is important that you first save this variable

I2c_master_has_data:
  'when your code is short, you need to put in a waitms statement
  'Take in mind that during this routine, a wait state is active and the
master will wait
  'After the return, the waitstate is ended

  Bfake = _a1                                                ' this is not
needed but it shows how you can store _A1 in a byte
  'after you have stored the received data into bFake, you can alter R16
  Config Portb = Output                                      ' make it an
output since it could be an input
  Portb = _a1                                                'assign _A1
(R16)
Return


'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!

'You could simply extend this sample so it will use 3 pins of PORT D for
the address selection
'For example portD.1 , portd.2 and portD.3 could be used for the address
selection
'Then after the CONFIG I2CSLAVE = &H40 statement, you can put code like:
'Dim switches as Byte    ' dim byte
'switches = PIND         ' get dip switch value
'switches = switches and &H1110 ' we only need the lower nibble without the
LS bit
'_i2c_slave_address = &H40 + switches ' set the proper address
```

The **TWI** slave library works similar but requires a different configuration.

Action
Configure the TWI Slave address and bit rate

Syntax
CONFIG TWISLAVE = address , BTR = value , BITRATE = value , SAVE=option [,GENCALL=value] [,USERACK=ack]

(I2C TWI Slave is part of the I2C-Slave library. This is an add-on library that is not included in Bascom-AVR by default. It is a commercial add on library. It is available from MCS Electronics)

See also: I2C TWI Slave, USING I2C Protocol, Using USI, CONFIG I2CSLAVE

Remarks

Address	The slave address that is assigned to the slave chip. This must be an Even number. Bit 0 of the address is used to activate the general call address. The GENCAL option will set this bit automatic. I2C uses a 7 bit address from bit 1 to bit 7. Bit 0 is used to specify a read/write operation. In BASCOM the byte transmission address is used for I2C. This means that an I2C 7-bit address of 1 becomes &B10 = 2. And we say the address is 2. This is done so you can copy the address from the data sheets which are in the same format in most cases. So if you work with 7 bit address, you need to multiply the address by 2.
BTR	Bytes to receive. With this constant you specify how many bytes will be expected when the master reads data from the slave. And thus how many bytes will be sent to the master.
Bit rate	This is the I2C/TWI clock frequency. Most chips support 400 KHz (400000) but all I2C chips support 100000.
SAVE	SAVE = NOSAVE : this can be used when you do not change a lot of registers in the interrupt. SAVE = SAVE : this is best to be used when you do not use ASM in the TWI interrupt. See the explanation below. When you do not specify SAVE, the default will be SAVE=SAVE.
GENCALL	General call address activated or not. When you specify 1 or YES, the General call address will be activated which mean that the slave will respond not only to it's own address, but also to the general call address 0. When you omit the option or specify 0 or NO, the general call address will not be honored.
USERACK	Default is OFF. When you use ON, an alternative library will be used. This library will create a variable named TWI_ACK. Each time your code is called this variable is filled with the value 255. If you do not alter the value, the slave will send an ACK as it is supposed to. If you reset the value to 0, the slave will send a NACK. You can use this to send data with variable length to the slave. In this case, BTR only serves as an index. You must make sure to reset TWI_ACK when you have send the last byte to the master.

The variables Twi , Twi_btr and Twi_btw are created by the compiler. These are all bytes
The TWI interrupt is enabled but you need to enabled the global interrupt

The TWI Slave code is running as an interrupt process. Each time there is a TWI interrupt some slave code is executed. Your BASIC code is called from the low level slave code under a number of events. You must include all these labels in your Slave application. You do not need to write code in all these sub routines.

Label	Event
Twi_stop_rstart_received	The Master sent a stop(i2CSTOP) or repeated start. Typical you do not need to do anything here.
Twi_addressed_goread	The master has addressed the slave and will now continue to send data to the slave. You do not need to take action here.
Twi_addressed_gowrite	The master has addressed the slave and will now continue to receive data from the slave. You do not need to take action here.
Twi_gotdata	The master has sent data. The variable **TWI** holds the received value. The byte **TWI_BTW** is an index that holds the value of the number of received bytes. The first received byte will have an index value of 1.
Twi_master_needs_byte	The master reads from the slave and needs a value. The variable TWI_BTR can be inspected to see which index byte was needed. With the CONFIG **BTR**, you specify how many bytes the master will read.

In most cases your main application is just an empty DO LOOP. But when you write a slave that performs other tasks on the background these other tasks are interrupted by the TWI traffic.
Take in mind that the interrupt with the lowest address has the highest priority.
So do NOT write blocking code inside an interrupt. While servicing another interrupt, the TWI interrupt can not be serviced.

The TWI Slave code will save all used registers.
But since it will call your BASIC application when the TWI interrupt occurs, your BASIC code could be in the middle of say a PRINT statement.
When you then execute another PRINT statement , you will destroy registers.
So keep the code in the sub routines to a minimum, and use SAVE option to save all registers. This is the default.
While two printing commands will give odd results (print 12345 and 456 in the middle of the first print will give 1234545) at least no register is destroyed.

A typical configuration is shown below.

To test the above hardware, use the samples : twi-master.bas and twi-slave.bas
Optional you can use i2cscan.bas to test the general call address.

When you want to change the address of the slave at run time you need to write to the **TWAR** register.
The TWAR register contains the slave address. Bit 0 which is used to indicate a read or write transaction should be cleared. When you set it, the slave will also recognize the general call address. The GENCALL option just sets bit 0 of the slave.

See also

CONFIG TWI , CONFIG SCL , CONFIG SDA , I2C TWI Slave, Using the I2C protocol

Example1(master)

```
'-------------------------------------------------------------------------
-----
'                             (c) 2004 MCS Electronics
'                         This demo shows an example of the TWI
'                        Not all AVR chips have TWI (hardware I2C)
'-------------------------------------------------------------------------
-----

'The chip will work in TWI/I2C master mode
'Connected is a PCF8574A 8-bits port extender


$regfile = "M88def.dat"                                     ' the used chip
$crystal = 8000000                                          ' frequency
used
$baud = 19200                                               ' baud rate
$hwstack = 32                                               ' default use
32 for the hardware stack
$swstack = 10                                               ' default use
10 for the SW stack
$framesize = 40                                             ' default use
40 for the frame space


$lib "i2c_twi.lbx"                                          ' we do not use
software emulated I2C but the TWI

Config Scl = Portc.5                                        ' we need to
provide the SCL pin name
Config Sda = Portc.4                                        ' we need to
provide the SDA pin name

'On the Mega88,          On the PCF8574A
'scl=PC5 , pin 28            pin 14
'sda=PC4 , pin 27            pin 15


I2cinit                                                     ' we need to
set the pins in the proper state


Config Twi = 100000                                         ' wanted clock
frequency
'will set TWBR and TWSR
'Twbr = 12                                                   'bit rate
register
'Twsr = 0                                                   'pre scaler
bits

Dim B As Byte , X As Byte
Print "TWI master"
Do
  Incr B                                                    ' increase
value
  I2csend &H0 , B                                           ' send the
value to general call address
```

```
  I2csend &H70 , B                                          ' send the
value
  Print "Error : " ; Err                                    ' show error
status
  I2creceive &H70 , X                                       ' get a byte
  Print X ; " " ; Err                                       ' show error
  Waitms 500                                                'wait a bit
Loop
End
```

Example2(slave)

```
'-------------------------------------------------------------------------
-----
'                            (c) 2004 MCS Electronics
'                This demo shows an example of the TWI in SLAVE mode
'                       Not all AVR chips have TWI (hardware I2C)
' IMPORTANT : this example ONLY works when you have the TWI slave library
'             which is a commercial add on library, not part of BASCOM
'Use this sample in combination with i2cscan.bas and/or twi-master.bas
'-------------------------------------------------------------------------
-----
$regfile = "M88def.dat"                                     ' the chip we
use
$crystal = 8000000                                          ' crystal
oscillator value
$baud = 19200                                               ' baud rate
$hwstack = 32                                               ' default use
32 for the hardware stack
$swstack = 10                                               ' default use
10 for the SW stack
$framesize = 40                                             ' default use
40 for the frame space

Print "MCS Electronics TWI-slave demo"

Config Twislave = &H70 , Btr = 1 , Bitrate = 100000 , Gencall = 1
'In i2c the address has 7 bits. The LS bit is used to indicate read or
write
'When the bit is 0, it means a write and a 1 means a read
'When you address a slave with the master in bascom, the LS bit will be
set/reset automatic.
'The TWAR register in the AVR is 8 bit with the slave address also in the
most left 7 bits
'This means that when you setup the slave address as &H70, TWAR will be set
to &H0111_0000
'And in the master you address the slave with address &H70 too.
'The AVR TWI can also recognize the general call address 0. You need to
either set bit 0 for example
'by using &H71 as a slave address, or by using GENCALL=1

'as you might need other interrupts as well, you need to enable them all
manual
Enable Interrupts

'this is just an empty loop but you could perform other tasks there
Do
  nop
Loop
End

'A master can send or receive bytes.
```

```
'A master protocol can also send some bytes, then receive some bytes
'The master and slave must match.

'the following labels are called from the library
Twi_stop_rstart_received:
  Print "Master sent stop or repeated start"
Return


Twi_addressed_goread:
  Print "We were addressed and master will send data"
Return


Twi_addressed_gowrite:
  Print "We were addressed and master will read data"
Return


'this label is called when the master sends data and the slave has received
the byte
'the variable TWI holds the received value
Twi_gotdata:
   Print "received : " ; Twi
Return

'this label is called when the master receives data and needs a byte
'the variable twi_btr is a byte variable that holds the index of the needed
byte
'so when sending multiple bytes from an array, twi_btr can be used for the
index
Twi_master_needs_byte:
  Print "Master needs byte : " ; Twi_btr
  Twi = 65                                                  ' twi must be
filled with a value
Return


'when the mast has all bytes received this label will be called
Twi_master_need_nomore_byte:
  Print "Master does not need anymore bytes"
Return
```

For **XMEGA** the configuration is similar as for TWI.

CONFIG TWICSLAVE = address , BTR = value [,GENCALL=value]

```
'-------------------------------------------------------------------------
----
'name                     : xmega-twi-slave.bas
'copyright                : (c) 1995-2013, MCS Electronics
'purpose                  : demonstrates Xmega TWI slave add on
'micro                    : Xmega128A1
'suited for demo          : yes
'commercial addon needed  : yes
'-------------------------------------------------------------------------
----

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40


'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits = 1 ,
Databits = 8
'Config Serialin = Buffered , Size = 50

'Enable Interrupts
Open "COM1:" For Binary As #1

Open "TWID" For Binary As #4                                ' or use
TWIC,TWIE oR TWIF
Config Twid = 100000                                        'CONFIG TWI
will ENABLE the TWI master interface
'you can also use TWIC, TWID, TWIE of TWIF
'!!!!!!!!!!!   WITHOUT a channel identifier, TWIC will be used
!!!!!!!!!!!!!
'SCL is on pin 1
'SDA is on pin 0
'This demo uses TWID as master and TWIC as SLAVE
'Thus portc.0 connects with portD.0 and
'     portc.1 connects with portD.1

'The TWIC when used as a slave has megaAVR compatible labels
'The TWID,TWIE and TWIF have unique new labelnames
'These labels are the labels in your code which are called from the slave
ISR.
'For example : Twi_addressed_gowrite  is named TwiD_addressed_gowrite for
TWID


Dim Twi_start As Byte , j as byte , b as byte
I2cinit #4                      'init the master
config TWIcslave = &H70 , btr = 2  'use address &H70 which is &H38 in 7-bit
i2c notation
```

```
Enable INTERRUPTS 'for the slave to work we must enable global interrupts

do
   Print #1 , "test xmega"

   For J = 0 To 120 Step 1  'notice that we scan odd and even addresses
      I2cstart #4           'send start
      I2cwbyte J , #4       'send value of J
      If Err = 0 Then       ' no errors
         Print #1 , "FOUND : " ; Hex(j)
         if j.0 = 0 then  'ONLY if R/W bit is not set we may write data !
            I2cwbyte 100 , #4   'just write to values to the slave
            I2cwbyte 101 , #4
         else                   'read
            I2crbyte b , Ack , #4 : print #1 , "GOT : " ; b 'read 2 bytes
            I2crbyte b , nAck , #4 : print #1 , "GOT : " ; b
         end if
      End If
      I2cstop #4                'done
   Next
   waitms 2000                  'wait some time
loop


'the following labels are called from the library  when master send stop or
start
'notice that these label names are valid for TWIC.
'for TWID the name would be TWID_stop_rstart_received:
Twi_stop_rstart_received:
   Print #1 , "Master sent stop or repeated start"
Return

'master sent our slave address and will not send data
Twi_addressed_goread:
   Print #1 , "We were addressed and master will send data"
Return


Twi_addressed_gowrite:
   Print #1 , "We were addressed and master will read data"
Return

'this label is called when the master sends data and the slave has received
the byte
'the variable TWIx holds the received value
'The x is the TWI interface letter
Twi_gotdata:
   Print #1 , "received : " ; Twic ; " byte no : " ; Twic_btw
   'here you would do something with the received data
   '   Select Case Twic_btw
   '     Case 1 : Portb = Twi                                    ' first
byte
   '     Case 2:                                                 'you can
set another port here for example
   '   End Select
Return

'this label is called when the master receives data and needs a byte
'the variable twix_btr is a byte variable that holds the index of the
needed byte
```

```
'so when sending multiple bytes from an array, twix_btr can be used for the
index
'again the variable name depends on the twi interface
Twi_master_needs_byte:
   Print #1 , "Master needs byte : " ; Twic_btr
   Select Case Twic_btr
      Case 1:                                                    ' first byte
         twic = 66                                          'we assign a value
but this could be any value you want
      Case 2                                                     ' send second
byte
         twic = 67
   End Select
Return


'when the mast has all bytes received this label will be called
Twi_master_need_nomore_byte:
   Print #1 , "Master does not need anymore bytes"
Return

End
```

When writing your own slave device it is simplest to copy one of the examples and use it as a template or starting point for your own slave.

USI as SLAVE

Syntax
CONFIG USI=mode , Address=adr

Remarks
The USI(universal serial Interface) is found in most atTiny processors. It can be used for various tasks. At the moment only the TWI slave mode is supported. The other modes you need to configure/code yourself.
The CONFIG USI = TWISLAVE mode requires a library that is part of the i2c slave add on which is a commercial add on.

Mode	The supported mode is : TWISLAVE. This will set the USI in TWI slave mode. The USI works in interrupt mode on the background.
Address	This is the I2C/TWI slave address. Notice that bascom uses the 8-bit address notation.

When USI is used in TWI/I2C mode, it does require that SCL and SDA have pull up resistors. You can not freely choose the SCL and SDA pins : you must use the fixed SCL en SDA pins.

The variables TWI_USI_OVS , TWI_slaveAddress, Twi , Twi_btr , TWI_bflag and Twi_btw are created by the compiler. These are all bytes
The USI interrupts are enabled but you need to enabled the global interrupt using ENABLE INTERRUPTS

The USI Slave code is running as an interrupt process. Each time there is an USI interrupt some slave code is executed. Your BASIC code is called from the low level slave code at a number of events.
You must include all these labels in your Slave application. You do not need to write code in all these sub routines.

Label	Event
Twi_stop_rstart_received	The Master sent a stop(i2CSTOP) or repeated start. Typical you do not need to do anything here.
Twi_addressed_goread	The master has addressed the slave and will now continue to send data to the slave. You do not need to take action here.
Twi_addressed_gowrite	The master has addressed the slave and will now continue to receive data from the slave. You do not need to take action here.
Twi_gotdata	The master has sent data. The variable **TWI** holds the received value. The byte **TWI_BTW** is an index that holds the value of the number of received bytes. The first received byte will have an index value of 1.
Twi_master_needs_byte	The master reads from the slave and needs a value. The variable TWI_BTR can be inspected to see which index byte was needed.

See also

Using USI , CONFIG TWISLAVE, CONFIG TWIXSLAVE

Example

```
'-------------------------------------------------------------------------
-----
'                             (c) 2004-2014 MCS Electronics
'          This demo demonstrates the USI I2C slave
'          Not all AVR chips have an USI !!!!
'-------------------------------------------------------------------------
-----

$regfile = "attiny2313.dat"

$crystal = 8000000
$hwstack = 40
$swstack = 16
$framesize = 24

const cPrint = 0                                            'make 0 for
chips that have NO UART, make 1 when the micro has a UART and you want to
show data on the terminal

#if cPrint
   $baud = 19200                                            'only when the
processor has a UART
#endif

config usi = twislave , address = &H40                      'bascom uses 8
bit i2c address (7 bit shifted to the left with one bit)

#if cPrint
   print "USI DEMO"
#endif

'do not forget to enable global interrupts since USI is used in interrupt
mode
enable interrupts                                           'it is
important you enable interrupts

do
   !  nop                                                   ; nothing to do
here
loop



'The following labels are called from the library. You need to insert code
in these subroutines
'Notice that the PRINT commands are remarked.
'You can unmark them and see what happens, but it will increase code size
'The idea is that you write your code in the called labels. And this code
must execute in as little time
'as possible. So when you slave must read the A/D converter, you can best
do it in the main program
'then the data is available when the master requires it, and you do not
need to do the conversion which cost time.
```

```
'A master can send or receive bytes.
'A master protocol can also send some bytes, then receive some bytes
'The master and slave address must match.

'the following labels are called from the library  when master send stop or
start
Twi_stop_rstart_received:
   '  Print "Master sent stop or repeated start"
Return

'master sent our slave address and will not send data
Twi_addressed_goread:
   ' Print "We were addressed and master will send data"
Return


Twi_addressed_gowrite:
   '  Print "We were addressed and master will read data"
Return

'this label is called when the master sends data and the slave has received
the byte
'the variable TWI holds the received value
Twi_gotdata:
   '   Print "received : " ; Twi ; " byte no : " ; Twi_btw
   Select Case Twi_btw
      Case 1 :                                                  'Portd = Twi
' first byte
      Case 2:                                                   'you can set
another port here for example
   End Select
Return

'this label is called when the master receives data and needs a byte
'the variable twi_btr is a byte variable that holds the index of the needed
byte
'so when sending multiple bytes from an array, twi_btr can be used for the
index
Twi_master_needs_byte:
   '  Print "Master needs byte : " ; Twi_btr
   Select Case Twi_btr
      Case 1 : twi = 68                                          ' first byte
      Case 2 : twi = 69                                          ' send second
byte
   End Select                                                   'you could
also return the state of a port pin or A/D converter
Return
```