```
{{
  ____________________________________________________________________________
  *************************************************
  *           BS2 Function Library               *
  *  Functional Equivalents of many BS2 Commands  *
  *                Version 1.0.0                  *
  *                  3/14/06                      *
  *           Contact: Martin Hebel               *
  *  martin@selmaware.com  or  mhebel@siu.edu     *
  *************************************************
  *  --- Distribute Freely Unmodified ---         *
  *  Feel free to modify, but please rename       *
  *  prior to distributing modified versions      *
  *  (append your own initials).                  *
  *  Denote orginal author(s) where appropriate   *
  *************************************************

    To use include in code:
  ------------------------------------------------
  CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000

  VAR
    Long stack1[50]                  ' Stack for 2nd BS2 Cog

  OBJ

    BS2 : "BS2_Functions"          ' Create BS2 Object

  PUB Start | x
      BS2.start (31,30)              ' Initialize BS2 Object, Rx and Tx pins for DEBUG
      cognew(Read,@Stack1)           ' Start a cog to interact with 1st
  ------------------------------------------------

    Use Functions as:
       BS2.FREQOUT(pin,duration,frequency)
       or
       MyVariable = BS2.PULSIN(pin,state)

  ____________________________________________________________________________

  Note: SHIFTIN/OUT, DEBUG/IN and SERIN/OUT are only recommended at 80MHz
        Maximum Baud Rate is 9600.
        For more options, use the"FullDuplexSerial" Library.

}}


var
  long s, ms, us,Last_Freq
  Byte DataIn[50],DEBUG_PIN,DEBUGIN_PIN

con
  ' SHIFTIN Constants
  MSBPRE   = 0
  LSBPRE   = 1
  MSBPOST  = 2
  LSBPOST  = 3
  ' SHIFTOUT Constants
  LSBFIRST = 0
  MSBFIRST = 1
  ' SEROUT/SERIN Constants
  NInv     = 1
  Inv      = 0

  cntMin     = 400      ' Minimum waitcnt value to prevent lock-up
  DEBUG_BAUD = 9600     ' DEBUG Serial speed - maximum!
  DEBUG_MODE = 1        ' Non-Inverted

PUB Start (Debug_rx, Debug_tx)
'' Initialize variables and pins for DEBUGIN and DEBUG, typically:
''   BS2.Start(31,30)

  Debug_Pin := Debug_tx                       ' DEBUG Tx Pin
  DebugIn_Pin := Debug_rx                     ' DEBUG Rx Pin
  s:= clkfreq                                 ' Clock cycles for 1 s
```

```
  ms:= clkfreq / 1_000                     ' Clock cycles for 1 ms
  us:= clkfreq / 1_000_000                 ' Clock cycles for 1 us
  Last_Freq := 0                           ' Holds last setting for FREQOUT_SET

PUB COUNT (Pin, Duration) : Value
{{
  Counts rising-edge pulses on Pin for Duration in mS
  Maximum count is around 30MHz
  Example Code:
    x := BS2.count(5,100)              ' Measure count for 100 mSec
    BS2.Debug_Dec(x)                   ' DEBUG value
    BS2.Debug_Char(13)                 ' CR
}}

  dira[PIN]~                                               ' Set as input
       ctra := 0                                           ' Clear any value in ctra
                                                           ' set up counter, pos edge trigger
       ctra := (%01010 << 26 ) | (%001 << 23) | (0 << 9) | (PIN)
       frqa := 1                                           ' 1 count/trigger
       phsa:=0                                             ' Clear phase - holds accumulated count
       pause(duration)                                     ' Allow to count for duration in mS
       Value := phsa                                       ' Reurn total count

PUB DEBUG_CHAR(Char)
{{
  Sends chracter (byte) data at 9600 Baud to DEBUG TX pin.
    BS2.Debug_Char(13)  ' CR
    BS2.Debug_Char(65)  ' Letter A
    BS2.Debug_Char("A") ' Letter A
 }}

   SEROUT_CHAR(Debug_Pin,char,DEBUG_Baud,DEBUG_MODE,8)      ' Send character using SEROUT

PUB DEBUG_BIN(value, Digits)
{{
  Sends value as binary value without %, for up to 32 Digits
    BS2.DEBUG_BIN(x,16)
      Code adapted from "FullDuplexSerial"
}}
  value <<= 32 - digits                                    ' Shift bits for number of digits
  repeat digits                                            ' Repeat for number of digital
    DEBUG_CHAR((value <-= 1) & 1 + "0")                    ' Shift value and test each for 1 + 0 ASCII

PUB DEBUG_DEC(Value)
{{
  Sends value as decimal value.
    BS2.DEBUG_DEC(x)
}}

  SEROUT_DEC(Debug_Pin,Value,Debug_Baud,DEBUG_MODE,8)       ' Send using SEROUT_DEC code

PUB DEBUG_HEX(value, digits)
{{
  Sends value as binary value without $ for number of digits defined
    BS2.DEBUG_HEX(x,4)
      Code adapted from "FullDuplexSerial"
}}
  value <<= (8 - digits) << 2                              ' Shiftover for number of hex digits
  repeat digits                                            ' lookup ASCII for nibble, sub and shift
     Debug_CHAR(lookupz((value <-= 4) & $F : "0".."9", "A".."F"))

PUB DEBUG_STR(stringPtr)
{{
  Sends a string for DEBUGing
    BS2.Debug_Str(string("Spin-Up World!",13))
    BS2.Debug_Str(@myStr)

}}
    SEROUT_Str(Debug_Pin,stringPtr,Debug_Baud,DEBUG_MODE,8)' send using serout_str


PUB DEBUG_IBIN(value, digits)
{{
  Sends value as binary value with %, for up to 32 Digits
    BS2.DEBUG_IBIN(x,16)
}}
```

```
    debug_CHAR("%")                                       ' Send leading %
    DEBUG_BIN(value,digits)                               ' Send value as binary


PUB DEBUG_IHEX(Value,Digits)
{{
  Sends value as binary value without $ for number of digits defined
    BS2.Debug_IHEX(x,4)
}}
    DEBUG_CHAR("$")                                       ' Send leading $
    DEBUG_HEX(value,digits)                               ' Send value as Hex


PUB DEBUGIN_CHAR : ByteVal
{{
  Accepts a single serial character (byte) on DEBUGIN_Pin at 9600 Baud
  Will cause cog-lockup while waiting without a cog-watchdog (see example)
    x := BS2.DEBUGIN_Char
    BS2.DEBUG_Char(x)
}}
    ByteVal := SERIN_CHAR(DEBUGIN_PIN,DEBUG_BAUD,DEBUG_MODE,8)' Send character using SEROUT_Char


PUB DEBUGIN_DEC : Value
{{
  Accepts a decimal value on DEBUGIN_Pin at 9600 Baud, up through a CR
  Will cause cog-lockup while waiting without a cog-watchdog (see example)
  Values may be +/-, no error checking for garbage.
    x := BS2.DEBUGIN_Dec
    BS2.DEBUG_Dec(x)
}}
   Value := SERIN_DEC(DEBUGIN_PIN,DEBUG_BAUD,DEBUG_MODE,8)     ' Get using Serin_DEC

PUB DEBUGIN_STR (stringptr)
{{
  Accepts a character string on DEBUGIN_Pin at 9600 Baud, up through a CR
  Maximum is 49 character, such as "abc, 123, you and me!"
  Will cause cog-lockup while waiting without a cog-watchdog (see example)
  NOTE: There is NO buffer overflow protection!

VAR
    Byte myString[50]

    Repeat
        BS2.DebugIn_Str(@myString)    ' Accept string passing pointer for variable
        BS2.Debug_Str(@myString)      ' display string at pointer
        BS2.Debug_Char(13)            ' CR
        BS2.Debug_Char(myString[5])   ' show 5th character

}}

   SERIN_Str(DEBUGIN_Pin, stringPtr, DEBUG_Baud, DEBUG_MODE, 8)    ' Get using SERIN_Str

PUB FREQOUT(Pin,Duration, Frequency)
{{
  Plays frequency defines on pin for duration in mS, does NOT support dual frequencies.
    BS2.Freqout(5,500,2500)     ' Produces 2500Hz on Pin 5 for 500 mSec
}}
   Update(Pin,Frequency,0)                                ' Set tone using FREQOUT_Set
   Pause(Duration)                                        ' duration pause
   Update(Pin,0,0)                                        ' stop tone


PUB FREQOUT_SET(Pin, Frequency)
{{
   Plays frequency defined on pin INDEFINATELY does NOT support dual frequencies.
   Use Frequency of 0 to stop.
     FREQOUT_Set(5, 2500)       ' Produces 2500Hz on Pin 5 forever
     FREQOUT_Set(5,0)           ' Turns off frequency
}}
   If Frequency <> Last_Freq                              ' Check to see if freq change
      Update(Pin,Frequency,0)                             ' update tone
      Last_Freq := Frequency                              ' save last


PUB FREQIN (pin, duration) : Frequency
{{
```

```
  Measure frequency on pin defined for duration defined.
  Positive edge triggered
    x:= BS2.FreqIn(5)
}}
  dira[PIN]~
      ctra := 0                                              ' Clear ctra settings
                                                             ' trigger to count rising edge on pin
      ctra := (%01010 << 26 ) | (%001 << 23) | (0 << 9) | (PIN)
      frqa := 1000                                           ' count 1000 each trigger
        phsa:=0                                              ' clear accumulated value
        pause(duration)                                      ' pause for duration
        Frequency := phsa / duration                         ' calculate freq based on duration

PUB PAUSE(Duration) | clkCycles
{{
   Causes a pause for the duration in mS
   Smallest value is 2 at clkfreq = 5Mhz, higher frequencies may use 1
   Largest value is around 50 seconds at 80Mhz.
     BS2.Pause(1000)   ' 1 second pause
}}

   clkCycles := Duration * ms-2300 #> cntMin                 ' duration * clk cycles for ms
                                                             ' - inst. time, min cntMin
   waitcnt( clkCycles + cnt )                                ' wait until clk gets there


PUB PAUSE_uS(Duration) | clkCycles
{{
   Causes a pause for the duration in uS
   Smallest value is 20 at clkfreq = 80Mhz
   Largest value is around 50 seconds at 80Mhz.
     BS2.Pause_uS(1000)   ' 1 mS pause
}}
   clkCycles := Duration * uS #> cntMin                      ' duration * clk cycles for us
                                                             ' - inst. time, min cntMin
   waitcnt(clkcycles + cnt)                                  ' wait until clk gets there



PUB PULSOUT(Pin,Duration)  | clkcycles
{{
   Produces an opposite pulse on the pin for the duration in 2uS increments
   Smallest value is 10 at clkfreq = 80Mhz
   Largest value is around 50 seconds at 80Mhz.
     BS2.Pulsout(500)   ' 1 mS pulse
}}
  ClkCycles := (Duration * us * 2 - 1250) #> cntMin          ' duration * clk cycles for 2us
                                                             ' - inst. time, min cntMin
  dira[pin]~~                                                ' Set to output

  !outa[pin]                                                 ' set to opposite state
  waitcnt(clkcycles + cnt)                                   ' wait until clk gets there
  !outa[pin]                                                 ' return to orig. state
PUB PULSOUT_uS(Pin,Duration) | ClkCycles
{{
   Produces an opposite pulse on the pin for the duration in 1uS increments
   Smallest value is 10 at clkfreq = 80Mhz
   Largest value is around 50 seconds at 80Mhz.
     BS2.Pulsout_uS(500)   ' 0.5 mS pulse
}}
  ClkCycles := (Duration * us-1050) #> cntMin                ' duration * clk cycles for us
                                                             ' - inst. time, min cntMin
  dira[pin]~~                                                ' Set to output

  !outa[pin]                                                 ' set to opposite state

  waitcnt(clkcycles + cnt)                                   ' wait until clk gets there

  !outa[pin]                                                 ' return to orig. state



PUB PULSIN (Pin, State) : Duration
{{
  Reads duration of Pulse on pin defined for state, returns duration in 2uS resolution
  Shortest measureable pulse is around 20uS
  Note: Absence of pulse can cause cog lockup if watchdog is not used - See distributed example
```

```
    x := BS2.Pulsin(5,1)
    BS2.Debug_Dec(x)
}}

   Duration := PULSIN_Clk(Pin, State) / us / 2 + 1          ' Use PulsinClk and calc for 2uS increments


PUB PULSIN_uS (Pin, State) : Duration | ClkStart, clkStop, timeout
{{
  Reads duration of Pulse on pin defined for state, returns duration in 1uS resolution
  Note: Absence of pulse can cause cog lockup if watchdog is not used - See distributed example
    x := BS2.Pulsin_uS(5,1)
    BS2.Debug_Dec(x)
}}

   Duration := PULSIN_Clk(Pin, State) / us + 1              ' Use PulsinClk and calc for 1uS increments

PUB PULSIN_Clk(Pin, State) : Duration
{{
  Reads duration of Pulse on pin defined for state, returns duration in 1/clkFreq increments - 12.5nS at
80MHz
  Note: Absence of pulse can cause cog lockup if watchdog is not used - See distributed example
    x := BS2.Pulsin_Clk(5,1)
    BS2.Debug_Dec(x)
}}

  DIRA[pin]~
  ctra := 0
  if state == 1
    ctra := (%11010 << 26 ) | (%001 << 23) | (0 << 9) | (PIN) ' set up counter, A level count
  else
    ctra := (%10101 << 26 ) | (%001 << 23) | (0 << 9) | (PIN) ' set up counter, !A level count
  frqa := 1
  waitpne(State << pin, |< Pin, 0)                          ' Wait for opposite state ready
  phsa:=0                                                   ' Clear count
  waitpeq(State << pin, |< Pin, 0)                          ' wait for pulse
  waitpne(State << pin, |< Pin, 0)                          ' Wait for pulse to end
  Duration := phsa                                          ' Return duration as counts
  ctra :=0                                                  ' stop counter

PUB PWM(Pin, Duty, Duration) | htime, ltime, Loop_Dur
{{
   Produces 400hz PWM on pin at 0-255 for duration in mS
   allowable range 2 - 252 at 80MHz clock freq, is 3-252 at 20MHz, 12-242 at 1Mhz
     PWM(5,128,1000)
}}
    htime := us * 10 * Duty #> cntMin                       ' Calculate high time
    ltime := us * 10 * (255-Duty) #> cntMin                 ' calculate low time
    dira[pin]~~                                             ' set as output

    if Duty < 1                                             ' Duty 0? always low
       outa[pin]:=0
       pause(duration)
    elseif Duty > 254                                       ' Duty 255? High alway high
       outa[pin]:=1
       pause(duration)
    else
        outa[pin]:=0
        Repeat Duration * 100/255                           ' Send drive for duration time
          !outa[pin]
          waitcnt(htime + cnt)                              ' High time
          !outa[pin]
          waitcnt(ltime + cnt)                              ' Low time


PUB PWM_100(Pin, Duty, Duration) | htime, ltime, Loop_Dur
{{
  Produces 1Khz PWM on pin at 0-100% for duration in mS
    PWM_100(5,128,1000)
}}

    htime := us * 10 * Duty #> cntMin                       ' Calculate high time
    ltime := us * 10 * (100-Duty) #> cntMin                 ' calculate low time
    dira[pin]~~                                             ' set as output

    if Duty == 1                                            ' Duty 0? always low
       outa[pin]:=0
```

```
      pause(duration)
   elseif Duty > 99                                          ' Duty 100? High alway high
      outa[pin]:=1
      pause(duration)
   else
       outa[pin]:=0
       Repeat Duration                                       ' Send drive for duration time
         !outa[pin]
         waitcnt(htime + cnt)                                ' High time
         !outa[pin]
         waitcnt(ltime + cnt)                                ' Low time

PUB RCTIME (Pin,State):Duration | ClkStart, ClkStop
{{
   Reads RCTime on Pin starting at State, returns discharge time scaled to BS2 values
     dira[5]~~                   ' Set as output
     outa[5]:=1                  ' Set high
     BS2.Pause(10)               ' Allow to charge
     x := RCTime(5,1)            ' Measure RCTime
     BS2.DEBUG_DEC(x)            ' Display
}}

   DIRA[Pin]~
   ClkStart := cnt                                           ' Save counter
   waitpne(State << pin, |< Pin, 0)                          ' Wait for opposite state to end
   clkStop := cnt                                            ' Save stop time
   Duration := (clkStop - ClkStart) * 1000                   ' calculate in 2us resolution, scale for BS2
   Duration := Duration / (clkfreq / 1000) * 100/130 #> 0    ' Minimum of 0


PUB SERIN_CHAR(pin, Baud, Mode, Bits) : ByteVal | x, BR
{{
  Accepts asynchronous character (byte) on defined pin, at Baud, in Mode for #bits
  Mode: 0 = Inverted - Normally low        Constant: BS2#Inv
        1 = Non-Inverted - Normally High   Constant: BS2#NInv
    SERIN_Char(5,DEBUG_Baud,BS2#NInv,8)
    Debug_Char(x)
    Debug_DEC(x)
}}
    BR := 1_000_000 / Baud                                   ' Calculate bit rate
    dira[PIN]~                                               ' Set as input
    waitpeq(Mode << PIN, |< PIN, 0)                          ' Wait for idle
    waitpne(Mode << PIN, |< PIN, 0)                          ' WAit for Start bit
    pause_us(BR*100/90)                                      ' Pause to be centered in 1st bit time
    byteVal := ina[Pin]                                      ' Read LSB
    Repeat x from 1 to Bits-1                                            ' Number of bits - 1
        pause_us(BR-70)                                      ' Wait until center of next bit
        ByteVal := ByteVal | (ina[Pin] << x)                 ' Read next bit, shift and store


PUB SERIN_DEC (Pin,Baud,Mode,Bits) : value | ByteIn, ptr, x, place
{{
  Accepts asynchronous decimal value (-1234) on defined pin, at Baud, in Mode for #bits/character
  Does not check for garbage (123A!5)
  Mode: 0 = Inverted - Normally low        Constant: BS2#Inv
        1 = Non-Inverted - Normally High   Constant: BS2#NInv
    x := SERIN_Char(5,DEBUG_Baud,1,8)
    Debug_Char(x)
    Debug_DEC(x)
}}
    place := 1                                               ' Set place to 1's
    ptr :=-1                                                 ' ptr to -1 to advance to 0 in loop
    repeat while DataIn[ptr] <> 13                           ' Keep accepting until CR
       ptr++                                                 ' increment pointer
       dataIn[ptr] := SERIN_CHAR(Pin, Baud, Mode, Bits)      ' Accept data from SERIN_Char
    repeat x from (ptr-1) to 1                               ' Count down from last in to first in
        value := value + ((DataIn[x]-"0") * place)           ' Get value by subtracting ASCII 0 x place
       place := place * 10                                   ' next place
    if dataIn[0] == "-"                                      ' Check if - sign
       value := value * -1
    elseif dataIn[0] == "+"                                  ' check if + sign
       value := value
    else
       value := value + (DataIn[0]-48) * place               ' if neither + or -, use value

PUB SERIN_STR (Pin,stringptr,Baud,Mode,Bits) | ptr
```

```
{{
  Accepts a character string on defnined Pin at Baud for bits/char, up through a CR (13)
  Maximum is 49 character, such as "abc, 123, you and me!"
  Will cause cog-lockup while waiting without a cog-watchdog (see distributed example)
  NOTE: There is NO buffer overflow protection.

VAR
    Byte myString[50]

    Repeat
        BS2.Serin_Str(5,@myString,9600,1,8)    ' Accept string passing pointer for variable
        BS2.Debug_Str(@myString)               ' display string at pointer
        BS2.Debug_Char(13)                     ' CR
        BS2.Debug_Char(myString[5])            ' show 6th character
}}
    dira[pin]~                                               ' Set pin to input
    bytefill(@dataIn,1,49)                                   ' Fill string memory with 0's (null)
    repeat while DataIn[ptr] <> 13                           ' accept character and until CR
       ptr++
       dataIn[ptr] := SERIN_CHAR(Pin, Baud, Mode, Bits)      ' Store character in string
    dataIn[ptr]:=0                                           ' set last character to null
    byteMove(stringptr,@datain,50)                           ' move into string pointer position


PUB SEROUT_CHAR(Pin, char, Baud, Mode, Bits ) | x, BR
{{
  Send asynchronous character (byte) on defined pin, at Baud, in Mode for #bits
  Mode: 0 = Inverted - Normally low         Constant: BS2#Inv
        1 = Non-Inverted - Normally High    Constant: BS2#NInv
    Serout_Char(5,"A",9600,BS2#NInv,8)
}}
        BR := 1_000_000 / (Baud)                             ' Determine Baud rate
        char := ((1 << Bits ) + char) << 2                   ' Set up string with start & stop bit
        dira[pin]~~                                          ' set as output
        if MODE == 0                                         ' If mode 0, invert
                char:= !char
        pause_us(BR * 2 )                                    ' Hold for 2 bits
        Repeat x From 1 to (Bits + 2)                        ' Send each bit based on baud rate
          char := char >> 1
          outa[Pin] := char
          pause_us(BR - 65)
        return
PUB SEROUT_DEC(Pin, Value, Baud, Mode, Bits) | i
{{
  Send asynchronous decimal value (-1234) on defined pin, at Baud, in Mode for #bits/Char
  Mode: 0 = Inverted - Normally low         Constant: BS2#Inv
        1 = Non-Inverted - Normally High    Constant: BS2#NInv
    BS2.SEROUT_dec(5,-1234,9600,1,8)    ' Tx -1234
    BS2.SEROUT_Char(5,13,9600,1,8)      ' CR to end
}}
'' Print a decimal number

  if value < 0                                               ' Send - sign if < 0
    -value
    SEROUT_CHAR(Pin, "-", Baud, Mode,Bits)

  i := 1_000_000_000

  repeat 10                                                  ' test each 10's place
    if value => i                                            ' send character based on ASCII 0
      SEROUT_CHAR(Pin, value / i + "0", Baud, Mode,Bits)     ' Take modulus of i
      value //= i
      result~~
    elseif result or i == 1
      SEROUT_CHAR(Pin, "0", Baud, Mode,Bits)                 ' Divide i for next place
    i /= 10


PUB SEROUT_STR(Pin, stringptr, Baud, Mode, bits)
{{
  Sends a string for serout
    BS2.Serout_Str(5,string("Spin-Up World!",13),9600,1,8)
    BS2.Serout_Str(5,@myStr,9600,1,8)
      Code adapted from "FullDuplexSerial"
}}

    repeat strsize(stringptr)
```

```
    SEROUT_CHAR(Pin,byte[stringptr++],Baud, Mode, bits)  ' Send each character in string

PUB SHIFTIN (Dpin, Cpin, Mode, Bits) : Value | InBit
{{
   Shift data in, master clock, for mode use BS2#MSBPRE, #MSBPOST, #LSBPRE, #LSBPOST
   Clock rate is ~16Kbps.  Use at 80MHz only is recommended.
     X := BS2.SHIFTIN(5,6,BS2#MSBPOST,8)
}}
    dira[Dpin]~                                           ' Set data pin to input
    outa[Cpin]:=0                                         ' Set clock low
    dira[Cpin]~~                                          ' Set clock pin to output

    If Mode == MSBPRE                                     ' Mode - MSB, before clock
       Value:=0
       REPEAT Bits                                        ' for number of bits
          InBit:= ina[Dpin]                               ' get bit value
          Value := (Value << 1) + InBit                   ' Add to  value shifted by position
          !outa[Cpin]                                     ' cycle clock
          !outa[Cpin]
          waitcnt(1000 + cnt)                             ' time delay

    elseif Mode == MSBPOST                                ' Mode - MSB, after clock
       Value:=0
       REPEAT Bits                                        ' for number of bits
          !outa[Cpin]                                     ' cycle clock
          !outa[Cpin]
          InBit:= ina[Dpin]                               ' get bit value
          Value := (Value << 1) + InBit                   ' Add to  value shifted by position

          waitcnt(1000 + cnt)                             ' time delay

    elseif Mode == LSBPOST                                ' Mode - LSB, after clock
       Value:=0
       REPEAT Bits                                        ' for number of bits
          !outa[Cpin]                                     ' cycle clock
          !outa[Cpin]
          InBit:= ina[Dpin]                               ' get bit value
          Value := (InBit << (bits-1)) + (Value >> 1)     ' Add to  value shifted by position
          waitcnt(1000 + cnt)                             ' time delay

    elseif Mode == LSBPRE                                 ' Mode - LSB, before clock
       Value:=0
       REPEAT Bits                                        ' for number of bits
          InBit:= ina[Dpin]                               ' get bit value
          Value := (Value >> 1) + (InBit << (bits-1))     ' Add to  value shifted by position
          !outa[Cpin]                                     ' cycle clock
          !outa[Cpin]
          waitcnt(1000 + cnt)                             ' time delay


PUB SHIFTIN_SLV (Dpin, Cpin, Mode, Bits) : Value | InBit
{{
  Shift data in, SLAVE clock (other device clocks),
  For mode use BS2#MSBPRE, #MSBPOST, #LSBPRE, #LSBPOST
  Clock rate above 16Kbps is not recommended.  Use at 80MHz only is recommended.
  Can cause cog lockup awaiting clock pulses.
    X := BS2.SHIFTIN_SLV(5,6,BS2#MSBPOST,8)
    BS2.DEBUG_DEC(x)
}}
    dira[Dpin]~                                           ' Same as SHIFTIN, but clock
    dira[Cpin]~                                           '  acts as input (slave)
    outa[Cpin]:=0
    If Mode == MSBPRE
       Value:=0
       REPEAT Bits
          InBit:= ina[Dpin]
          Value := (Value << 1) + InBit
          waitpeq(1<< Cpin,|< Cpin, 0)                     ' wait on clock
          waitpne(1<< Cpin,|< Cpin, 0)
    elseif Mode == MSBPOST
       Value:=0
       REPEAT Bits
          waitpeq(1<< Cpin,|< Cpin, 0)
          waitpne(1<< Cpin,|< Cpin, 0)
          InBit:= ina[Dpin]
          Value := (Value << 1) + InBit
```

```
    elseif Mode == LSBPOST
       Value:=0
       REPEAT Bits
          waitpeq(1<< Cpin,|< Cpin, 0)
          waitpne(1<< Cpin,|< Cpin, 0)
          InBit:= ina[Dpin]
          Value := (InBit << (bits-1)) + (Value >> 1)

    elseif Mode == LSBPRE
       Value:=0
       REPEAT Bits
          InBit:= ina[Dpin]
          Value := (Value >> 1) + (InBit << (bits-1))
          waitpeq(1<< Cpin,|< Cpin, 0)
          waitpne(1<< Cpin,|< Cpin, 0)



PUB SHIFTOUT (Dpin, Cpin, Value, Mode, Bits)| bitNum
{{
   Shift data out, master clock, for mode use ObjName#LSBFIRST, #MSBFIRST
   Clock rate is ~16Kbps.  Use at 80MHz only is recommended.
     BS2.SHIFTOUT(5,6,"B",BS2#LSBFIRST,8)
}}
    outa[Dpin]:=0                                             ' Data pin = 0
    dira[Dpin]~~                                              ' Set data as output
    outa[Cpin]:=0
    dira[Cpin]~~

    If Mode == LSBFIRST                                       ' Send LSB first
       REPEAT Bits
          outa[Dpin] := Value                                 ' Set output
          Value := Value >> 1                                 ' Shift value right
          !outa[Cpin]                                         ' cycle clock
          !outa[Cpin]
          waitcnt(1000 + cnt)                                 ' delay

    elseIf Mode == MSBFIRST                                   ' Send MSB first
       REPEAT Bits
          outa[Dpin] := Value >> (bits-1)                     ' Set output
          Value := Value << 1                                 ' Shift value right
          !outa[Cpin]                                         ' cycle clock
          !outa[Cpin]
          waitcnt(1000 + cnt)                                 ' delay
    outa[Dpin]~                                               ' Set data to low
PUB SHIFTOUT_SLV (Dpin, Cpin, Value, Mode,  Bits) | bitNum
{{
  Shift data out, SLAVE clock (other device clocks).
  For mode use ObjName#LSBFIRST, #MSBFIRST
    Clock rates above 16Kbps is not recommended.  Use at 80MHz only is recommended.
}}
    outa[Dpin]:=0                                             ' Same as above, but acts as slave
    dira[Dpin]~~
    dira[Cpin]~

    If Mode == LSBFIRST
       REPEAT Bits
          outa[Dpin] := Value
          Value := Value >> 1
          waitpeq(1 << Cpin,|< Cpin, 0)                       ' wait for clock
          waitpne(1 << Cpin,|< Cpin, 0)

    elseIf Mode == MSBFIRST
       REPEAT Bits
          outa[Dpin] := Value >> (Bits-1)
          Value := Value << 1
          waitpeq(1<< Cpin,|< Cpin, 0)
          waitpne(1<< Cpin,|< Cpin, 0)

    outa[Dpin]:=0



PRI update(pin, freq, ch) | temp

{{updates either the A or B counter modules.
```

```
  Parameters:
    pin  - I/O pin to transmit the square wave
    freq - The frequency in Hz
    ch   - 0 for counter module A, or 1 for counter module B
  Returns:
    The value of cnt at the start of the signal
    Adapted from Code by Andy Lindsay
}}

      if freq == 0                                             ' freq = 0 turns off square wave
        waitpeq(0, |< pin, 0)                                  ' Wait for low signal
        ctra := 0                                              ' Set CTRA/B to 0
        dira[pin]~                                             ' Make pin input
      temp := pin                                              ' CTRA/B[8..0] := pin
      temp += (%00100 << 26)                                   ' CTRA/B[30..26] := %00100
      ctra := temp                                             ' Copy temp to CTRA/B
      frqa := calcFrq(freq)                                    ' Set FRQA/B
      phsa := 0                                                ' Clear PHSA/B (start cycle low)
      dira[pin]~~                                              ' Make pin output
      result := cnt                                            ' Return the start time

PRI CalcFrq(freq)

  {Solve FRQA/B = frequency * (2^32) / clkfreq with binary long
  division (Thanks Chip!- signed Andy).

  Note: My version of this method relied on the FloatMath object.
  Not surprisingly, Chip's solution takes a fraction program space,
  memory, and time.  It's the binary long-division approach, which
  implements with the binary
  long division approach.}

  repeat 33
    result <<= 1
    if freq => clkfreq
      freq -= clkfreq
      result++
    freq <<= 1
```

```
repeat 33
  result <<= 1
  if freq => clkfreq
    freq -= clkfreq
    result++
  freq <<= 1
```