```
{{

  ┌──────────────────────────────────────────────┐
  ********************************************
  *        BS2 Function Library Object       *
  *  Functional Equivalents of many BS2 Commands  *
  *                Version 1.1.0               *
  *                  3/30/06                   *
  *         Primary Author: Martin Hebel       *
  *       Electronic Systems Technologies      *
  *   Southern Illinois University Carbondale  *
  *            www.siu.edu/~isat/est           *
  *                                            *
  * Questions? Please post on the Propeller forum *
  *       http://forums.parallax.com/forums/   *
  ********************************************
  *     --- Distribute Freely Unmodified ---   *
  ********************************************

    To use include in code:
  ------------------------------------------------
  CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_200_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 DE
      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.

Revision History:
3/14/06 - Initial release
3/30/06 - Modified RCTime function due to math errors with large values
        - Made modifications to documentation notes for varios functions

}}


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

```

```
60    con
61      ' SHIFTIN Constants
62      MSBPRE   = 0
63      LSBPRE   = 1
64      MSBPOST  = 2
65      LSBPOST  = 3
66      ' SHIFTOUT Constants
67      LSBFIRST = 0
68      MSBFIRST = 1
69      ' SEROUT/SERIN Constants
70      NInv     = 1
71      Inv      = 0
72
73      cntMin     = 400       ' Minimum waitcnt value to prevent lock-up
74      DEBUG_BAUD = 9600      ' DEBUG Serial speed - maximum!
75      DEBUG_MODE = 1         ' Non-Inverted
76
77    PUB Start (Debug_rx, Debug_tx)
78    '' Initialize variables and pins for DEBUGIN and DEBUG, typically:
79    ''    BS2.Start(31,30)
80
81      Debug_Pin := Debug_tx                    ' DEBUG Tx Pin
82      DebugIn_Pin := Debug_rx                  ' DEBUG Rx Pin
83      s:= clkfreq                              ' Clock cycles for 1 s
84      ms:= clkfreq / 1_000                     ' Clock cycles for 1 ms
85      us:= clkfreq / 1_000_000                 ' Clock cycles for 1 us
86      Last_Freq := 0                           ' Holds last setting for FREQOUT_SET
87
88    PUB COUNT (Pin, Duration) : Value
89    {{
90      Counts rising-edge pulses on Pin for Duration in mS
91      Maximum count is around 30MHz
92      Example Code:
93        x := BS2.count(5,100)                ' Measure count for 100 mSec
94        BS2.Debug_Dec(x)                     ' DEBUG value
95        BS2.Debug_Char(13)                   ' CR
96    }}
97
98      dira[PIN]~                                              ' Set as input
99           ctra := 0                                          ' Clear any value in ctra
100                                                             ' set up counter, pos edg
    trigger
101          ctra := (%01010 << 26 ) | (%001 << 23) | (0 << 9) | (PIN)
102          frqa := 1                                          ' 1 count/trigger
103          phsa:=0                                            ' Clear phase - holds
    accumulated count
104          pause(duration)                                    ' Allow to count for dura
    in mS
105          Value := phsa                                      ' Reurn total count
106
107
108   PUB DEBUG_CHAR(Char)
109   {{
110     Sends chracter (byte) data at 9600 Baud to DEBUG TX pin.
111       BS2.Debug_Char(13)   ' CR
112       BS2.Debug_Char(65)   ' Letter A
113       BS2.Debug_Char("A")  ' Letter A
114    }}
115
```

```
116     SEROUT_CHAR(Debug_Pin,char,DEBUG_Baud,DEBUG_MODE,8)       ' Send character using SE
117
118   PUB DEBUG_BIN(value, Digits)
119   {{
120     Sends value as binary value without %, for up to 32 Digits
121       BS2.DEBUG_BIN(x,16)
122         Code adapted from "FullDuplexSerial"
123   }}
124     value <<= 32 - digits                                     ' Shift bits for number o
      digits
125     repeat digits                                             ' Repeat for number of di
126       DEBUG_CHAR((value <-= 1) & 1 + "0")                     ' Shift value and test ea
      1 + 0 ASCII
127
128   PUB DEBUG_DEC(Value)
129   {{
130     Sends value as decimal value.
131       BS2.DEBUG_DEC(x)
132   }}
133
134     SEROUT_DEC(Debug_Pin,Value,Debug_Baud,DEBUG_MODE,8)       ' Send using SEROUT_DEC o
135
136   PUB DEBUG_HEX(value, digits)
137   {{
138     Sends value as binary value without $ for number of digits defined
139       BS2.DEBUG_HEX(x,4)
140         Code adapted from "FullDuplexSerial"
141   }}
142     value <<= (8 - digits) << 2                               ' Shiftover for number of
      digits
143     repeat digits                                             ' lookup ASCII for nibble
      and shift
144       Debug_CHAR(lookupz((value <-= 4) & $F : "0".."9", "A".."F"))
145
146   PUB DEBUG_STR(stringPtr)
147   {{
148     Sends a string for DEBUGing
149       BS2.Debug_Str(string("Spin-Up World!",13))
150       BS2.Debug_Str(@myStr)
151
152   }}
153       SEROUT_Str(Debug_Pin,stringPtr,Debug_Baud,DEBUG_MODE,8)' send using serout_str
154
155
156   PUB DEBUG_IBIN(value, digits)
157   {{
158     Sends value as binary value with %, for up to 32 Digits
159       BS2.DEBUG_IBIN(x,16)
160   }}
161       debug_CHAR("%")                                         ' Send leading %
162       DEBUG_BIN(value,digits)                                 ' Send value as binary
163
164
165   PUB DEBUG_IHEX(Value,Digits)
166   {{
167     Sends value as binary value without $ for number of digits defined
168       BS2.Debug_IHEX(x,4)
169   }}
170       DEBUG_CHAR("$")                                         ' Send leading $
```

```
171     DEBUG_HEX(value,digits)                                 ' Send value as Hex
172
173
174   PUB DEBUGIN_CHAR : ByteVal
175   {{
176     Accepts a single serial character (byte) on DEBUGIN_Pin at 9600 Baud
177     Will cause cog-lockup while waiting without a cog-watchdog (see example)
178       x := BS2.DEBUGIN_Char
179       BS2.DEBUG_Char(x)
180   }}
181       ByteVal := SERIN_CHAR(DEBUGIN_PIN,DEBUG_BAUD,DEBUG_MODE,8)' Send character using
      SEROUT_Char
182
183
184   PUB DEBUGIN_DEC : Value
185   {{
186     Accepts a decimal value on DEBUGIN_Pin at 9600 Baud, up through a CR
187     Will cause cog-lockup while waiting without a cog-watchdog (see example)
188     Values may be +/-, no error checking for garbage.
189       x := BS2.DEBUGIN_Dec
190       BS2.DEBUG_Dec(x)
191   }}
192      Value := SERIN_DEC(DEBUGIN_PIN,DEBUG_BAUD,DEBUG_MODE,8)     ' Get using Serin_DEC
193
194   PUB DEBUGIN_STR (stringptr)
195   {{
196     Accepts a character string on DEBUGIN_Pin at 9600 Baud, up through a CR
197     Maximum is 49 character, such as "abc, 123, you and me!"
198     Will cause cog-lockup while waiting without a cog-watchdog (see example)
199     NOTE: There is NO buffer overflow protection!
200
201   VAR
202       Byte myString[50]
203
204       Repeat
205           BS2.DebugIn_Str(@myString)    ' Accept string passing pointer for variable
206           BS2.Debug_Str(@myString)      ' display string at pointer
207           BS2.Debug_Char(13)            ' CR
208           BS2.Debug_Char(myString[5])   ' show 5th character
209
210   }}
211
212      SERIN_Str(DEBUGIN_Pin, stringPtr, DEBUG_Baud, DEBUG_MODE, 8)    ' Get using SERIN_
213
214   PUB FREQOUT(Pin,Duration, Frequency)
215   {{
216     Plays frequency defines on pin for duration in mS, does NOT support dual frequenc
217       BS2.Freqout(5,500,2500)     ' Produces 2500Hz on Pin 5 for 500 mSec
218   }}
219      Update(Pin,Frequency,0)                                  ' Set tone using FREQOUT_
220      Pause(Duration)                                          ' duration pause
221      Update(Pin,0,0)                                          ' stop tone
222
223
224   PUB FREQOUT_SET(Pin, Frequency)
225   {{
226      Plays frequency defined on pin INDEFINATELY does NOT support dual frequencies.
227      Use Frequency of 0 to stop.
228        BS2.FREQOUT_Set(5, 2500)   ' Produces 2500Hz on Pin 5 forever
```

```
229       BS2.FREQOUT_Set(5,0)       ' Turns off frequency
230   }}
231      If Frequency <> Last_Freq                                ' Check to see if freq ch
232         Update(Pin,Frequency,0)                               ' update tone
233         Last_Freq := Frequency                                ' save last
234
235
236   PUB FREQIN (pin, duration) : Frequency
237   {{
238     Measure frequency on pin defined for duration defined.
239     Positive edge triggered
240       x:= BS2.FreqIn(5)
241   }}
242     dira[PIN]~
243          ctra := 0                                            ' Clear ctra settings
244                                                               ' trigger to count rising
      on pin
245          ctra := (%01010 << 26 ) | (%001 << 23) | (0 << 9) | (PIN)
246          frqa := 1000                                         ' count 1000 each trigger
247            phsa:=0                                            ' clear accumulated value
248            pause(duration)                                    ' pause for duration
249            Frequency := phsa / duration                       ' calculate freq based on
      duration
250
251   PUB PAUSE(Duration) | clkCycles
252   {{
253      Causes a pause for the duration in mS
254      Smallest value is 2 at clkfreq = 5Mhz, higher frequencies may use 1
255      Largest value is around 50 seconds at 80Mhz.
256        BS2.Pause(1000)    ' 1 second pause
257   }}
258
259      clkCycles := Duration * ms-2300 #> cntMin                ' duration * clk cycles -
260                                                               ' - inst. time, min cntMi
261      waitcnt( clkCycles + cnt )                               ' wait until clk gets the
262
263
264   PUB PAUSE_uS(Duration) | clkCycles
265   {{
266      Causes a pause for the duration in uS
267      Smallest value is 20 at clkfreq = 80Mhz
268      Largest value is around 50 seconds at 80Mhz.
269        BS2.Pause_uS(1000)    ' 1 mS pause
270   }}
271      clkCycles := Duration * uS #> cntMin                     ' duration * clk cycles -
272                                                               ' - inst. time, min cntMi
273      waitcnt(clkcycles + cnt)                                 ' wait until clk gets the
274
275
276   PUB PULSOUT(Pin,Duration)  | clkcycles
277   {{
278      Produces an opposite pulse on the pin for the duration in 2uS increments
279      Smallest value is 10 at clkfreq = 80Mhz
280      Largest value is around 50 seconds at 80Mhz.
281        BS2.Pulsout(500)    ' 1 mS pulse
282   }}
283     ClkCycles := (Duration * us * 2 - 1250) #> cntMin         ' duration * clk cycles -
284                                                               ' - inst. time, min cntMi
```

```
285     dira[pin]~~                                             ' Set to output

286     !outa[pin]                                              ' set to opposite state
287     waitcnt(clkcycles + cnt)                                ' wait until clk gets the
288     !outa[pin]                                              ' return to orig. state
289   PUB PULSOUT_uS(Pin,Duration) | ClkCycles
290   {{
291      Produces an opposite pulse on the pin for the duration in 1uS increments
292      Smallest value is 10 at clkfreq = 80Mhz
293      Largest value is around 50 seconds at 80Mhz.
294        BS2.Pulsout_uS(500)    ' 0.5 mS pulse
295   }}
296     ClkCycles := (Duration * us-1050) #> cntMin             ' duration * clk cycles
297                                                             ' - inst. time, min cntM
298     dira[pin]~~                                             ' Set to output

299     !outa[pin]                                              ' set to opposite state

300     waitcnt(clkcycles + cnt)                                ' wait until clk gets the

301     !outa[pin]                                              ' return to orig. state


302
303
304   PUB PULSIN (Pin, State) : Duration
305   {{
306     Reads duration of Pulse on pin defined for state, returns duration in 2uS resolut
307     Shortest measureable pulse is around 20uS
308     Note: Absence of pulse can cause cog lockup if watchdog is not used - See distribu
    example
309       x := BS2.Pulsin(5,1)
310       BS2.Debug_Dec(x)
311   }}
312
313      Duration := PULSIN_Clk(Pin, State) / us / 2 + 1        ' Use PulsinClk and calc
    2uS increments
314
315
316   PUB PULSIN_uS (Pin, State) : Duration | ClkStart, clkStop, timeout
317   {{
318     Reads duration of Pulse on pin defined for state, returns duration in 1uS resolut
319     Note: Absence of pulse can cause cog lockup if watchdog is not used - See distribu
    example
320       x := BS2.Pulsin_uS(5,1)
321       BS2.Debug_Dec(x)
322   }}
323
324      Duration := PULSIN_Clk(Pin, State) / us + 1            ' Use PulsinClk and calc
    1uS increments
325
326   PUB PULSIN_Clk(Pin, State) : Duration
327   {{
328     Reads duration of Pulse on pin defined for state, returns duration in 1/clkFreq
    increments - 12.5nS at 80MHz
329     Note: Absence of pulse can cause cog lockup if watchdog is not used - See distribu
    example
330       x := BS2.Pulsin_Clk(5,1)
331       BS2.Debug_Dec(x)
332   }}
```

```
333
334     DIRA[pin]~
335     ctra := 0
336     if state == 1
337       ctra := (%11010 << 26 ) | (%001 << 23) | (0 << 9) | (PIN) ' set up counter, A le
    count
338     else
339       ctra := (%10101 << 26 ) | (%001 << 23) | (0 << 9) | (PIN) ' set up counter, !A
    count
340     frqa := 1
341     waitpne(State << pin, |< Pin, 0)                        ' Wait for opposite state
342     phsa:=0                                                 ' Clear count
343     waitpeq(State << pin, |< Pin, 0)                        ' wait for pulse
344     waitpne(State << pin, |< Pin, 0)                        ' Wait for pulse to end
345     Duration := phsa                                        ' Return duration as cour
346     ctra :=0                                                ' stop counter
347
348   PUB PWM(Pin, Duty, Duration) | htime, ltime, Loop_Dur
349   {{
350      Produces 400hz PWM on pin at 0-255 for duration in mS
351      allowable range 2 - 252 at 80MHz clock freq, is 3-252 at 20MHz, 12-242 at 1Mhz
352       BS2.PWM(5,128,1000)
353   }}
354       htime := us * 10 * Duty #> cntMin                     ' Calculate high time
355       ltime := us * 10 * (255-Duty) #> cntMin               ' calculate low time
356       dira[pin]~~                                           ' set as output
357
358       if Duty < 1                                           ' Duty 0? always low
359          outa[pin]:=0
360          pause(duration)
361       elseif Duty > 254                                     ' Duty 255? High alway h
362          outa[pin]:=1
363          pause(duration)
364       else
365           outa[pin]:=0
366           Repeat Duration * 100/255                        ' Send drive for duratio
367             !outa[pin]
368             waitcnt(htime + cnt)                           ' High time
369             !outa[pin]
370             waitcnt(ltime + cnt)                           ' Low time
371
372
373   PUB PWM_100(Pin, Duty, Duration) | htime, ltime, Loop_Dur
374   {{
375     Produces 1Khz PWM on pin at 0-100% for duration in mS
376       BS2.PWM_100(5,128,1000)
377   }}
378
379       htime := us * 10 * Duty #> cntMin                     ' Calculate high time
380       ltime := us * 10 * (100-Duty) #> cntMin               ' calculate low time
381       dira[pin]~~                                           ' set as output
382
383       if Duty == 1                                          ' Duty 0? always low
384          outa[pin]:=0
385          pause(duration)
386       elseif Duty > 99                                      ' Duty 100? High alway h
387          outa[pin]:=1
388          pause(duration)
389       else
```

```
390         outa[pin]:=0
391         Repeat Duration                                     ' Send drive for duration
392           !outa[pin]
393           waitcnt(htime + cnt)                              ' High time
394           !outa[pin]
395           waitcnt(ltime + cnt)                              ' Low time
396
397
398   PUB RCTIME (Pin,State):Duration | ClkStart, ClkStop
399   {{
400      Reads RCTime on Pin starting at State, returns discharge time, returns in 1uS uni
401        dira[5]~~                 ' Set as output
402        outa[5]:=1                ' Set high
403        BS2.Pause(10)             ' Allow to charge
404        x := RCTime(5,1)          ' Measure RCTime
405        BS2.DEBUG_DEC(x)          ' Display
406   }}
407
408      DIRA[Pin]~
409      ClkStart := cnt                                        ' Save counter for start
410      waitpne(State << pin, |< Pin, 0)                       ' Wait for opposite state
411      clkStop := cnt                                         ' Save stop time
412      Duration := (clkStop - ClkStart)/uS                    ' calculate in 1us resolu
413
414
415   PUB SERIN_CHAR(pin, Baud, Mode, Bits) : ByteVal | x, BR
416   {{
417     Accepts asynchronous character (byte) on defined pin, at Baud, in Mode for #bits
418     Mode: 0 = Inverted - Normally low        Constant: BS2#Inv
419           1 = Non-Inverted - Normally High   Constant: BS2#NInv
420       BS2.SERIN_Char(5,DEBUG_Baud,BS2#NInv,8)
421       BS2.Debug_Char(x)
422       BS2.Debug_DEC(x)
423   }}
424       BR := 1_000_000 / Baud                                 ' Calculate bit rate
425       dira[PIN]~                                             ' Set as input
426       waitpeq(Mode << PIN, |< PIN, 0)                        ' Wait for idle
427       waitpne(Mode << PIN, |< PIN, 0)                        ' WAit for Start bit
428       pause_us(BR*100/90)                                    ' Pause to be centered in
    bit time
429       byteVal := ina[Pin]                                    ' Read LSB
430       Repeat x from 1 to Bits-1                                          ' Number of b
431           pause_us(BR-70)                                    ' Wait until center of ne
432           ByteVal := ByteVal | (ina[Pin] << x)               ' Read next bit, shift ar
433
434
435   PUB SERIN_DEC (Pin,Baud,Mode,Bits) : value | ByteIn, ptr, x, place
436   {{
437     Accepts asynchronous decimal value (-1234) on defined pin, at Baud, in Mode for #b
    character
438     Does not check for garbage (123A!5)
439     Mode: 0 = Inverted - Normally low        Constant: BS2#Inv
440           1 = Non-Inverted - Normally High   Constant: BS2#NInv
441       x := SERIN_Char(5,DEBUG_Baud,1,8)
442       BS2.Debug_Char(x)
443       BS2.Debug_DEC(x)
444   }}
445       place := 1                                             ' Set place to 1's
```

```
446     ptr :=-1                                                ' ptr to -1 to advance to
    loop
447     repeat while DataIn[ptr] <> 13                          ' Keep accepting until CR
448        ptr++                                                ' increment pointer
449        dataIn[ptr] := SERIN_CHAR(Pin, Baud, Mode, Bits)     ' Accept data from SERIN_
450     repeat x from (ptr-1) to 1                              ' Count down from last in
    first in
451        value := value + ((DataIn[x]-"0") * place)           ' Get value by subtractin
    ASCII 0 x place
452        place := place * 10                                  ' next place
453     if dataIn[0] == "-"                                     ' Check if - sign
454        value := value * -1
455     elseif dataIn[0] == "+"                                 ' check if + sign
456        value := value
457     else
458        value := value + (DataIn[0]-48) * place              ' if neither + or -, use
459
460 PUB SERIN_STR (Pin,stringptr,Baud,Mode,Bits) | ptr
461 {{
462   Accepts a character string on defnined Pin at Baud for bits/char, up through a CR
463   Maximum is 49 character, such as "abc, 123, you and me!"
464   Will cause cog-lockup while waiting without a cog-watchdog (see distributed examp
465   NOTE: There is NO buffer overflow protection.
466
467 VAR
468     Byte myString[50]
469
470     Repeat
471         BS2.Serin_Str(5,@myString,9600,1,8)    ' Accept string passing pointer for va
472         BS2.Debug_Str(@myString)               ' display string at pointer
473         BS2.Debug_Char(13)                     ' CR
474         BS2.Debug_Char(myString[5])            ' show 6th character
475 }}
476     dira[pin]~                                              ' Set pin to input
477     bytefill(@dataIn,1,49)                                  ' Fill string memory with
    null)
478     repeat while DataIn[ptr] <> 13                          ' accept character and ur
479        ptr++
480        dataIn[ptr] := SERIN_CHAR(Pin, Baud, Mode, Bits)     ' Store character in stri
481     dataIn[ptr]:=0                                          ' set last character to n
482     byteMove(stringptr,@datain,50)                          ' move into string pointe
    position
483
484
485 PUB SEROUT_CHAR(Pin, char, Baud, Mode, Bits ) | x, BR
486 {{
487   Send asynchronous character (byte) on defined pin, at Baud, in Mode for #bits
488   Mode: 0 = Inverted - Normally low        Constant: BS2#Inv
489         1 = Non-Inverted - Normally High   Constant: BS2#NInv
490     BS2.Serout_Char(5,"A",9600,BS2#NInv,8)
491 }}
492         BR := 1_000_000 / (Baud)                            ' Determine Baud rate
493         char := ((1 << Bits ) + char) << 2                  ' Set up string with star
    stop bit
494         dira[pin]~~                                         ' set as output
495         if MODE == 0                                        ' If mode 0, invert
496                 char:= !char
497         pause_us(BR * 2 )                                   ' Hold for 2 bits
498         Repeat x From 1 to (Bits + 2)                       ' Send each bit based on
```

```
rate
499             char := char >> 1
500             outa[Pin] := char
501             pause_us(BR - 65)
502           return
503   PUB SEROUT_DEC(Pin, Value, Baud, Mode, Bits) | i
504   {{
505     Send asynchronous decimal value (-1234) on defined pin, at Baud, in Mode for #bits
506     Mode: 0 = Inverted - Normally low        Constant: BS2#Inv
507           1 = Non-Inverted - Normally High   Constant: BS2#NInv
508       BS2.SEROUT_dec(5,-1234,9600,1,8)    ' Tx -1234
509       BS2.SEROUT_Char(5,13,9600,1,8)      ' CR to end
510   }}
511   '' Print a decimal number
512
513     if value < 0                                            ' Send - sign if < 0
514       -value
515       SEROUT_CHAR(Pin, "-", Baud, Mode,Bits)
516
517     i := 1_000_000_000
518
519     repeat 10                                               ' test each 10's place
520       if value => i                                         ' send character based or
521   0
521         SEROUT_CHAR(Pin, value / i + "0", Baud, Mode,Bits)    ' Take modulus of i
522         value //= i
523         result~~
524       elseif result or i == 1
525         SEROUT_CHAR(Pin, "0", Baud, Mode,Bits)              ' Divide i for next place
526       i /= 10
527
528
529   PUB SEROUT_STR(Pin, stringptr, Baud, Mode, bits)
530   {{
531     Sends a string for serout
532       BS2.Serout_Str(5,string("Spin-Up World!",13),9600,1,8)
533       BS2.Serout_Str(5,@myStr,9600,1,8)
534         Code adapted from "FullDuplexSerial"
535   }}
536
537       repeat strsize(stringptr)
538         SEROUT_CHAR(Pin,byte[stringptr++],Baud, Mode, bits)  ' Send each character in
539
540   PUB SHIFTIN (Dpin, Cpin, Mode, Bits) : Value | InBit
541   {{
542      Shift data in, master clock, for mode use BS2#MSBPRE, #MSBPOST, #LSBPRE, #LSBPOST
```

```
543     Clock rate is ~16Kbps.  Use at 80MHz only is recommended.
544       X := BS2.SHIFTIN(5,6,BS2#MSBPOST,8)
545  }}
546      dira[Dpin]~                                            ' Set data pin to input
547      outa[Cpin]:=0                                          ' Set clock low
548      dira[Cpin]~~                                           ' Set clock pin to outpu
549
550      If Mode == MSBPRE                                      ' Mode - MSB, before clo
551         Value:=0
552         REPEAT Bits                                         ' for number of bits
553            InBit:= ina[Dpin]                                ' get bit value
554            Value := (Value << 1) + InBit                    ' Add to  value shifted b
     position
555            !outa[Cpin]                                      ' cycle clock
556            !outa[Cpin]
557            waitcnt(1000 + cnt)                              ' time delay
558
559      elseif Mode == MSBPOST                                 ' Mode - MSB, after cloc

560         Value:=0
561         REPEAT Bits                                         ' for number of bits

562            !outa[Cpin]                                      ' cycle clock

563            !outa[Cpin]
564            InBit:= ina[Dpin]                                ' get bit value

565            Value := (Value << 1) + InBit                    ' Add to  value shifted b
     position
566            waitcnt(1000 + cnt)                              ' time delay

567
568      elseif Mode == LSBPOST                                 ' Mode - LSB, after cloc

569         Value:=0

570         REPEAT Bits                                         ' for number of bits

571            !outa[Cpin]                                      ' cycle clock

572            !outa[Cpin]

573            InBit:= ina[Dpin]                                ' get bit value

574            Value := (InBit << (bits-1)) + (Value >> 1)      ' Add to  value shifted b
     position
575            waitcnt(1000 + cnt)                              ' time delay

576
577      elseif Mode == LSBPRE                                  ' Mode - LSB, before clo

578         Value:=0

579         REPEAT Bits                                         ' for number of bits

580            InBit:= ina[Dpin]                                ' get bit value

581            Value := (Value >> 1) + (InBit << (bits-1))      ' Add to  value shifted b
     position
```

```
582           !outa[Cpin]                                           ' cycle clock
583           !outa[Cpin]
584           waitcnt(1000 + cnt)                                   ' time delay
585
586
587   PUB SHIFTIN_SLV (Dpin, Cpin, Mode, Bits) : Value | InBit
588   {{
589     Shift data in, SLAVE clock (other device clocks),
590     For mode use BS2#MSBPRE, #MSBPOST, #LSBPRE, #LSBPOST
591     Clock rate above 16Kbps is not recommended.  Use at 80MHz only is recommended.
592     Can cause cog lockup awaiting clock pulses.
593       X := BS2.SHIFTIN_SLV(5,6,BS2#MSBPOST,8)
594       BS2.DEBUG_DEC(x)
595   }}
596       dira[Dpin]~                                               ' Same as SHIFTIN, but c
597       dira[Cpin]~                                               '  acts as input (slave)
598       outa[Cpin]:=0
599       If Mode == MSBPRE
600          Value:=0
601          REPEAT Bits
602             InBit:= ina[Dpin]
603             Value := (Value << 1) + InBit
604             waitpeq(1<< Cpin,|< Cpin, 0)                        ' wait on clock
605             waitpne(1<< Cpin,|< Cpin, 0)
606       elseif Mode == MSBPOST
607          Value:=0
608          REPEAT Bits
609             waitpeq(1<< Cpin,|< Cpin, 0)
610             waitpne(1<< Cpin,|< Cpin, 0)
611             InBit:= ina[Dpin]
612             Value := (Value << 1) + InBit
613
614       elseif Mode == LSBPOST
615          Value:=0
616          REPEAT Bits
617             waitpeq(1<< Cpin,|< Cpin, 0)
618             waitpne(1<< Cpin,|< Cpin, 0)
619             InBit:= ina[Dpin]
620             Value := (InBit << (bits-1)) + (Value >> 1)
621
622       elseif Mode == LSBPRE
623          Value:=0
624          REPEAT Bits
625             InBit:= ina[Dpin]
626             Value := (Value >> 1) + (InBit << (bits-1))
627             waitpeq(1<< Cpin,|< Cpin, 0)
628             waitpne(1<< Cpin,|< Cpin, 0)
629
630
631
632   PUB SHIFTOUT (Dpin, Cpin, Value, Mode, Bits)| bitNum
633   {{
634      Shift data out, master clock, for mode use ObjName#LSBFIRST, #MSBFIRST
635      Clock rate is ~16Kbps.  Use at 80MHz only is recommended.
636        BS2.SHIFTOUT(5,6,"B",BS2#LSBFIRST,8)
```

```
637   }}
638       outa[Dpin]:=0                                         ' Data pin = 0
639       dira[Dpin]~~                                          ' Set data as output
640       outa[Cpin]:=0
641       dira[Cpin]~~
642
643       If Mode == LSBFIRST                                   ' Send LSB first
644          REPEAT Bits
645             outa[Dpin] := Value                             ' Set output
646             Value := Value >> 1                             ' Shift value right
647             !outa[Cpin]                                     ' cycle clock
648             !outa[Cpin]
649             waitcnt(1000 + cnt)                             ' delay
650
651       elseIf Mode == MSBFIRST                               ' Send MSB first
652          REPEAT Bits
653             outa[Dpin] := Value >> (bits-1)                 ' Set output
654             Value := Value << 1                             ' Shift value right
655             !outa[Cpin]                                     ' cycle clock
656             !outa[Cpin]
657             waitcnt(1000 + cnt)                             ' delay
658       outa[Dpin]~                                           ' Set data to low
659
660   PUB SHIFTOUT_SLV (Dpin, Cpin, Value, Mode,  Bits) | bitNum
661   {{
662     Shift data out, SLAVE clock (other device clocks).
663     For mode use ObjName#LSBFIRST, #MSBFIRST
664       Clock rates above 16Kbps is not recommended.  Use at 80MHz only is recommended.
665   }}
666       outa[Dpin]:=0                                         ' Same as above, but acts
      slave
667       dira[Dpin]~~
668       dira[Cpin]~
669
670       If Mode == LSBFIRST
671          REPEAT Bits
672             outa[Dpin] := Value
673             Value := Value >> 1
674             waitpeq(1 << Cpin,|< Cpin, 0)                   ' wait for clock
675             waitpne(1 << Cpin,|< Cpin, 0)
676
677       elseIf Mode == MSBFIRST
678          REPEAT Bits
679             outa[Dpin] := Value >> (Bits-1)
680             Value := Value << 1
681             waitpeq(1<< Cpin,|< Cpin, 0)
682             waitpne(1<< Cpin,|< Cpin, 0)
683
684       outa[Dpin]:=0
685
686
687
688   PRI update(pin, freq, ch) | temp
689
690   {{updates either the A or B counter modules.
691
692     Parameters:
693       pin  - I/O pin to transmit the square wave
694       freq - The frequency in Hz
```

```
695       ch    - 0 for counter module A, or 1 for counter module B
696     Returns:
697       The value of cnt at the start of the signal
698       Adapted from Code by Andy Lindsay
699   }}
700
701         if freq == 0                                          ' freq = 0 turns off squa
702           waitpeq(0, |< pin, 0)                               ' Wait for low signal
703           ctra := 0                                           ' Set CTRA/B to 0
704           dira[pin]~                                          ' Make pin input
705         temp := pin                                           ' CTRA/B[8..0] := pin
706         temp += (%00100 << 26)                                ' CTRA/B[30..26] := %001(
707         ctra := temp                                          ' Copy temp to CTRA/B
708         frqa := calcFrq(freq)                                 ' Set FRQA/B
709         phsa := 0                                             ' Clear PHSA/B (start cy(
710         dira[pin]~~                                           ' Make pin output
711         result := cnt                                         ' Return the start time
712
713   PRI CalcFrq(freq)
714
715     {Solve FRQA/B = frequency * (2^32) / clkfreq with binary long
716     division (Thanks Chip!- signed Andy).
717
718     Note: My version of this method relied on the FloatMath object.
719     Not surprisingly, Chip's solution takes a fraction program space,
720     memory, and time.  It's the binary long-division approach, which
721     implements with the binary
722     long division approach.}
723
724     repeat 33
725       result <<= 1
726       if freq => clkfreq
727         freq -= clkfreq
728         result++
729       freq <<= 1
730
731
```

```
660   PUB SHIFTOUT_SLV (Dpin, Cpin, Value, Mode,  Bits) | bitNum
661   {{
662     Shift data out, SLAVE clock (other device clocks).
663     For mode use ObjName#LSBFIRST, #MSBFIRST
664       Clock rates above 16Kbps is not recommended.  Use at 80MHz only is recommended.
665   }}
666       outa[Dpin]:=0                                             ' Same as above, but acts
      slave
667       dira[Dpin]~~
668       dira[Cpin]~
669
670       If Mode == LSBFIRST
671          REPEAT Bits
672             outa[Dpin] := Value
673             Value := Value >> 1
674             waitpeq(1 << Cpin,|< Cpin, 0)                      ' wait for clock
675             waitpne(1 << Cpin,|< Cpin, 0)
676
677       elseIf Mode == MSBFIRST
678          REPEAT Bits
679             outa[Dpin] := Value >> (Bits-1)
680             Value := Value << 1
681             waitpeq(1<< Cpin,|< Cpin, 0)
682             waitpne(1<< Cpin,|< Cpin, 0)
683
684       outa[Dpin]:=0
685
686
687
688   PRI update(pin, freq, ch) | temp
689
690   {{updates either the A or B counter modules.
691
692     Parameters:
693       pin  - I/O pin to transmit the square wave
694       freq - The frequency in Hz
695       ch   - 0 for counter module A, or 1 for counter module B
696     Returns:
697       The value of cnt at the start of the signal
698       Adapted from Code by Andy Lindsay
699   }}
700
701         if freq == 0                                            ' freq = 0 turns off squ
702           waitpeq(0, |< pin, 0)                                 ' Wait for low signal
703           ctra := 0                                             ' Set CTRA/B to 0
704           dira[pin]~                                            ' Make pin input
705         temp := pin                                             ' CTRA/B[8..0] := pin
706         temp += (%00100 << 26)                                  ' CTRA/B[30..26] := %001(
707         ctra := temp                                            ' Copy temp to CTRA/B
708         frqa := calcFrq(freq)                                   ' Set FRQA/B
709         phsa := 0                                               ' Clear PHSA/B (start cy(
710         dira[pin]~~                                             ' Make pin output
711         result := cnt                                           ' Return the start time
712
713   PRI CalcFrq(freq)
714
715     {Solve FRQA/B = frequency * (2^32) / clkfreq with binary long
716     division (Thanks Chip!- signed Andy).
717
```

```
718     Note: My version of this method relied on the FloatMath object.
719     Not surprisingly, Chip's solution takes a fraction program space,
720     memory, and time.  It's the binary long-division approach, which
721     implements with the binary
722     long division approach.}
723
724     repeat 33
725       result <<= 1
726       if freq => clkfreq
727         freq -= clkfreq
728         result++
729       freq <<= 1
730
731
```