```
''**************************
''* Rotary Encoder v0.5     *
''* (C) 2005 Parallax, Inc. *
''**************************

VAR
  byte          Cog             'Cog (ID+1) that is running Update
  byte          TotDelta        'Number of encoders needing deta value support.
  long          Pos             'Address of position buffer


PUB Start(StartPin, NumEnc, NumDelta, PosAddr): Pass
''Record configuration, clear all encoder positions and launch a continuous encoder-reading cog.
''PARAMETERS: StartPin = (0..63) 1st pin of encoder 1.  2nd pin of encoder 1 is StartPin+1.
''                        Additional pins for other encoders are contiguous starting with StartPi
''            NumEnc   = Number of encoders (1..16) to monitor.
''            NumDelta = Number of encoders (0..16) needing delta value support (can be less tha
''            PosAddr  = Address of a buffer of longs where each encoder's position (and deta po
''RETURNS:    True if successful, False otherwise.

  Pin := StartPin
  TotEnc := NumEnc
  TotDelta := NumDelta
  Pos := PosAddr
  Stop
  longfill(Pos, 0, TotEnc+TotDelta)
  Pass := (Cog := cognew(@Update, Pos) + 1) > 0


PUB Stop
''Stop the encoder-reading cog, if there is one.

  if Cog > 0
    cogstop(Cog-1)


PUB ReadDelta(EncID): DeltaPos
''Read delta position (relative position value since last time read) of EncID.

  DeltaPos := 0 + -(EncID < TotDelta) * -long[Pos][TotEnc+EncID] + (long[Pos][TotEnc+EncID] := l



'************************************
'* Encoder Reading Assembly Routine *
'************************************

DAT
'Read all encoders and update encoder positions in main memory.
'See "Theory of Operation," below, for operational explanation.
'Cycle Calculation Equation:
'  Terms:     SU = :Sample to :Update.  UTI = :UpdatePos through :IPos.  MMW = Main Memory Write
'             AMMN = After MMW to :Next.  NU = :Next to :UpdatePos.  SH = Resync to Hub.  NS = :
'  Equation:  SU + UTI + MMW + (AMMN + NU + UTI + SH + MMW) * (TotEnc-1) + AMMN + NS
'             = 92 + 16  +  8  + ( 16  + 4  + 16  + 6  +  8 ) * (TotEnc-1) +  16  + 12
'             = 144 + 50*(TotEnc-1)

                        org     0
```

```
Update                  test    Pin, #$20               wc      'Test for upper or lower port
                        muxc    :PinSrc, #%1                    'Adjust :PinSrc instruction for
                        mov     IPosAddr, #IntPos               'Clear all internal encoder posi
                        movd    :IClear, IPosAddr               '  set starting internal pointer
                        mov     Idx, TotEnc                     '  for all encoders...
        :IClear         mov     0, #0                           '  clear internal memory
                        add     IPosAddr, #1                    '  increment pointer
                        movd    :IClear, IPosAddr
                        djnz    Idx, #:IClear                   '  loop for each encoder

                        mov     St2, ina                        'Take first sample of encoder pi
                        shr     St2, Pin
:Sample                 mov     IPosAddr, #IntPos               'Reset encoder position buffer a
                        movd    :IPos+0, IPosAddr
                        movd    :IPos+1, IPosAddr
                        mov     MPosAddr, PAR
                        mov     St1, St2                        'Calc 2-bit signed offsets (St1
                        mov     T1,  St2                        '                           T1
                        shl     T1, #1                          '                           T1
        :PinSrc         mov     St2, inb                        '  Sample encoders         (St2
                        shr     St2, Pin                        '  Adj for first encoder   (St2
                        xor     St1, St2                        '           St1  =              B
                        xor     T1, St2                         '           T1   =              A
                        and     T1, BMask                       '           T1   =              A
                        or      T1, AMask                       '           T1   =              A
                        mov     T2, St1                         '           T2   =              B
                        and     T2, AMask                       '           T2   =
                        and     St1, BMask                      '           St1  =              B
                        shr     St1, #1                         '           St1  =
                        xor     T2, St1                         '           T2   =
                        mov     St1, T2                         '           St1  =
                        shl     St1, #1                         '           St1  =        A1^B2^B
                        or      St1, T2                         '           St1  =        A1^B2^B
                        and     St1, T1                         '           St1  =  A1^B2^B1^A2&A
                        mov     Idx, TotEnc                     'For all encoders...
:UpdatePos              ror     St1, #2                         'Rotate current bit pair into 31
                        mov     Diff, St1                       'Convert 2-bit signed to 32-bit
                        sar     Diff, #30
        :IPos           add     0, Diff                         'Add to encoder position value
                        wrlong  0, MPosAddr                     'Write new position to main memo
                        add     IPosAddr, #1                    'Increment encoder position addr
                        movd    :IPos+0, IPosAddr
                        movd    :IPos+1, IPosAddr
                        add     MPosAddr, #4
:Next                   djnz    Idx, #:UpdatePos                'Loop for each encoder
                        jmp     #:Sample                        'Loop forever

'Define Encoder Reading Cog's constants/variables

AMask                   long    $55555555                       'A bit mask
BMask                   long    $AAAAAAAA                       'B bit mask
MSB                     long    $80000000                       'MSB mask for current bit pair

Pin                     long    0                               'First pin connected to first en
TotEnc                  long    0                               'Total number of encoders

Idx                     res     1                               'Encoder index
St1                     res     1                               'Previous state
St2                     res     1                               'Current state
```

```
T1                      res     1                               'Temp 1
T2                      res     1                               'Temp 2
Diff                    res     1                               'Difference, ie: -1, 0 or +1
IPosAddr                res     1                               'Address of current encoder posi
MPosAddr                res     1                               'Address of current encoder posi
IntPos                  res     16                              'Internal encoder position count

''
''
''*************************
''* FUNCTIONAL DESCRIPTION *
''*************************
''
''Reads 1 to 16 two-bit gray-code rotary encoders and provides 32-bit absolute position values f
''(value since last read) for up to 16 encoders.  See "Required Cycles and Maximum RPM" below fo
''
''Connect each encoder to two contiguous I/O pins (multiple encoders must be connected to a cont
''required, those encoders must be at the start of the group, followed by any encoders not requi
''
''To use this object:
''  1) Create a position buffer (array of longs).  The position buffer MUST contain NumEnc + Num
''     will always contain read-only, absolute positions for the respective encoders.  The remai
''     absolute read" storage for providing delta position support (if used) and should be ignor
''  2) Call Start() passing in the starting pin number, number of encoders, number needing delta
''     configure and start an encoder reader in a separate cog; which runs continuously until St
''  3) Read position buffer (first NumEnc values) to obtain an absolute 32-bit position value fo
''     the position buffer is updated automatically by the encoder reader cog.
''  4) For any encoders requiring delta position support, call ReadDelta(); you must have first
''     for this feature.
''
''Example Code:
''
''OBJ
''  Encoder : RotaryEncoder
''
''VAR
''  long Pos[3]                           'Create buffer for two encoders (plus room for delta
''
''PUB Init
''  Encoder.Start(8, 2, 1, @Pos)          'Start continuous two-encoder reader (encoders connec
''
''PUB Main
''  repeat
''    <read Pos[0] or Pos[1] here>        'Read each encoder's absolute position
''    <variable> := Encoder.ReadDelta(0)  'Read 1st encoder's delta position (value since last
''
''
''_____________________________________
''REQUIRED CYCLES AND MAXIMUM RPM:
''
''Encoder Reading Cog requires 144 + 50*(TotEnc-1) cycles per sample.  That is: 144 for 1 encode
''
''Conservative Maximum RPM of Highest Resolution Encoder = XINFreq * PLLMultiplier / EncReaderCo
''
''Example 1: Using a 4 MHz crystal, 8x internal multiplier, 16 encoders where the highest resolu
''           Max RPM = 4,000,000 * 8 / 894 / 2 / 1024 * 60 = 1,048 RPM
''
''Example 2: Using same example above, but with only 2 encoders of 128 pulses per revolution:
''           Max RPM = 4,000,000 * 8 / 194 / 2 / 128 * 60 = 38,659 RPM
```

```
'
'_____________________
'THEORY OF OPERATION:
'Column 1 of the following truth table illustrates 2-bit, gray code rotary encoder output (encod
'we're sampling fast enough).  A1 is the previous value of pin A, A2 is the current value of pin
'transition possibilities are not shown here because we won't ever see them if we're sampling fa
'if a transition is missed anyway.
'
'Column 2 shows each of the 2-bit results of cross XOR'ing the bits in the previous and current
'there is an actual transition, A1^B2 (msb of column 2) yields the direction (0 = clockwise, 1 =
'resulting 2-bit value gives more transition detail (00 or 11 if no transition, 01 if clockwise,
'
'Columns 3 and 4 show the results of further XORs and one AND operation.  The result is a conven
'clockwise, and -1 and if counter-clockwise.
'
'This object's Update routine performs the sampling (column 1) and logical operations (colum 3)
'resulting offset (-1, 0 or +1) to each position counter, iteratively.
'
'      1      |      2      |          3           |       4        |     5
'-------------|-------------|----------------------|----------------|-----------
'             |             | A1^B2^B1^A2&(A1^B2):  |   2-bit sign   |
'B1A1 -> B2A2 | A1^B2:B1^A2 |     A1^B2^B1^A2      | extended value | Diagnosis
'-------------|-------------|----------------------|----------------|-----------
' 00  ->  00  |     00      |          00          |      +0        |    No
' 01  ->  01  |     11      |          00          |      +0        | Movement
' 11  ->  11  |     00      |          00          |      +0        |
' 10  ->  10  |     11      |          00          |      +0        |
'-------------|-------------|----------------------|----------------|-----------
' 00  ->  01  |     01      |          01          |      +1        | Clockwise
' 01  ->  11  |     01      |          01          |      +1        |
' 11  ->  10  |     01      |          01          |      +1        |
' 10  ->  00  |     01      |          01          |      +1        |
'-------------|-------------|----------------------|----------------|-----------
' 00  ->  10  |     10      |          11          |      -1        | Counter-
' 10  ->  11  |     10      |          11          |      -1        | Clockwise
' 11  ->  01  |     10      |          11          |      -1        |
' 01  ->  00  |     10      |          11          |      -1        |
```