```
''**************************
''* 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
StartPin+2 but MUST NOT cross port boundry (31).
''             NumEnc   = Number of encoders (1..16) to monitor.
''             NumDelta = Number of encoders (0..16) needing delta value support (can be less
than NumEnc).
''             PosAddr  = Address of a buffer of longs where each encoder's position (and deta
position, if any) is to be stored.
''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] :=
long[Pos][EncID])



'***********************************
'* 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 =
:Next to :Sample.
'  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 proper port
                        mov     IPosAddr, #IntPos               'Clear all internal encoder
position values
                        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
pins
                        shr     St2, Pin
:Sample                 mov     IPosAddr, #IntPos               'Reset encoder position buffer
addresses
                        movd    :IPos+0, IPosAddr
                        movd    :IPos+1, IPosAddr
                        mov     MPosAddr, PAR
                        mov     St1, St2                        'Calc 2-bit signed offsets (
St1 = B1:A1)
                        mov     T1,  St2                        '                           T1
 = B1:A1
                        shl     T1, #1                          '                           T1
 = A1:x
        :PinSrc         mov     St2, inb                        '  Sample encoders          (
St2 = B2:A2 left shifted by first encoder offset)
                        shr     St2, Pin                        '  Adj for first encoder    (
St2 = B2:A2)
                        xor     St1, St2                        '           St1  =
B1^B2:A1^A2
                        xor     T1, St2                         '           T1   =
A1^B2:x
                        and     T1, BMask                       '           T1   =
A1^B2:0
                        or      T1, AMask                       '           T1   =
A1^B2:1
                        mov     T2, St1                         '           T2   =
B1^B2:A1^A2
                        and     T2, AMask                       '           T2   =
    0:A1^A2
                        and     St1, BMask                      '           St1  =
B1^B2:0
                        shr     St1, #1                         '           St1  =
    0:B1^B2
                        xor     T2, St1                         '           T2   =
    0:A1^A2^B1^B2
```

```
                        mov     St1, T2                         '           St1  =
    0:A1^B2^B1^A2
                        shl     St1, #1                         '           St1  =        A1^B2^
B1^A2:0
                        or      St1, T2                         '           St1  =        A1^B2^
B1^A2:A1^B2^B1^A2
                        and     St1, T1                         '           St1  =  A1^B2^B1^A2&
A1^B2:A1^B2^B1^A2
                        mov     Idx, TotEnc                     'For all encoders...
:UpdatePos              ror     St1, #2                         'Rotate current bit pair into
31:30
                        mov     Diff, St1                       'Convert 2-bit signed to 32-
bit signed Diff
                        sar     Diff, #30
        :IPos           add     0, Diff                         'Add to encoder position value
                        wrlong  0, MPosAddr                     'Write new position to main
memory
                        add     IPosAddr, #1                    'Increment encoder position
addresses
                        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
encoder
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
position counter (Internal Memory)
MPosAddr                res     1                               'Address of current encoder
position counter (Main Memory)
IntPos                  res     16                              'Internal encoder position
counter buffer

''
''
''*************************
''* FUNCTIONAL DESCRIPTION *
''*************************
''
''Reads 1 to 16 two-bit gray-code rotary encoders and provides 32-bit absolute position values
for each and optionally provides delta position support
''(value since last read) for up to 16 encoders.  See "Required Cycles and Maximum RPM" below
for speed boundary calculations.
''
```

```
''Connect each encoder to two contiguous I/O pins (multiple encoders must be connected to a
contiguous block of pins).  If delta position support is
''required, those encoders must be at the start of the group, followed by any encoders not
requiring delta position support.
''
''To use this object:
''  1) Create a position buffer (array of longs).  The position buffer MUST contain NumEnc +
NumDelta longs.  The first NumEnc longs of the position buffer
''     will always contain read-only, absolute positions for the respective encoders.  The
remaining NumDelta longs of the position buffer will be "last
''     absolute read" storage for providing delta position support (if used) and should be
ignored (use ReadDelta() method instead).
''  2) Call Start() passing in the starting pin number, number of encoders, number needing
delta support and the address of the position buffer.  Start() will
''     configure and start an encoder reader in a separate cog; which runs continuously until
Stop is called.
''  3) Read position buffer (first NumEnc values) to obtain an absolute 32-bit position value
for each encoder.  Each long (32-bit position counter) within
''     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 sized the position buffer and configured Start() appropriately
''     for this feature.
''
''Example Code:
''
''OBJ
''  Encoder : RotaryEncoder
''
''VAR
''  long Pos[3]                              'Create buffer for two encoders (plus room for
delta position support of 1st encoder)
''
''PUB Init
''  Encoder.Start(8, 2, 1, @Pos)             'Start continuous two-encoder reader (encoders
connected to pins 8 - 11)
''
''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 read)
''
''_____________________________________
''REQUIRED CYCLES AND MAXIMUM RPM:
''
''Encoder Reading Cog requires 144 + 50*(TotEnc-1) cycles per sample.  That is: 144 for 1
encoder, 194 for 2 encoders, 894 for 16 encoders.
''
''Conservative Maximum RPM of Highest Resolution Encoder = XINFreq * PLLMultiplier /
EncReaderCogCycles / 2 / MaxEncPulsesPerRevolution * 60
''
''Example 1: Using a 4 MHz crystal, 8x internal multiplier, 16 encoders where the highest
resolution encoders is 1024 pulses per revolution:
''           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 (
encoder pins A and B) and their possible transitions (assuming
'we're sampling fast enough).  A1 is the previous value of pin A, A2 is the current value of
pin A, etc.  '->' means 'transition to'.  The four double-step
'transition possibilities are not shown here because we won't ever see them if we're sampling
fast enough and, secondly, it is impossible to tell direction
'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 values.  Because of the encoder's gray code output, when
'there is an actual transition, A1^B2 (msb of column 2) yields the direction (0 = clockwise, 1
= counter-clockwise).  When A1^B2 is paired with B1^A2, the
'resulting 2-bit value gives more transition detail (00 or 11 if no transition, 01 if
clockwise, 10 if counter-clockwise).
'
'Columns 3 and 4 show the results of further XORs and one AND operation.  The result is a
convenient set of 2-bit signed values: 0 if no transition, +1 if
'clockwise, and -1 and if counter-clockwise.
'
'This object's Update routine performs the sampling (column 1) and logical operations (colum 3)
 of up to 16 2-bit pairs in one operation, then adds the
'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        |
```