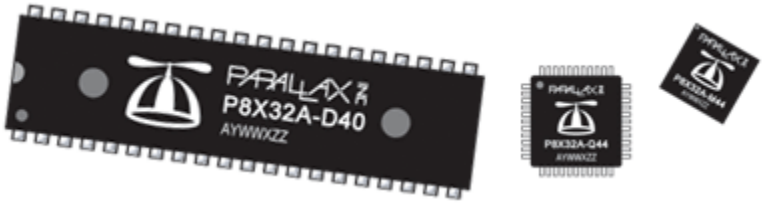
PROPELLER
by Parallax, Inc.

SPIN CODE EXAMPLES



Compiled by
Charlie Johnson
A.K.A. dy8coke

Table of Contents

INTRODUCTION

PLEASE POST EXAMPLES OF SPIN CODE ON THIS THREAD THAT MEET THE FOLLOWING CRITERIA:

1. Can be used to learn Spin code
2. Have the necessary documentation
3. Have been tested

THE DIFFICULTY LEVEL OF THESE EXAMPLES IS SET WITH THE FOLLOWING IN MIND:

(1) The user has no Propeller with which to execute the code
(2) The user has no documentation

```
'*****************************************************************************
'             EXPLANATION OF TERMS IN EXAMPLES USING PROCEDURES
'*****************************************************************************
'  PROCEDURE
'     The term procedure is an umbrella term that can refer to a function procedure
'     (function), a sub-procedure, or an event procedure. (Microsoft's definition)
'
'     (1) FUNCTION PROCEDURE (or FUNCTION)
'         When a function is called it performs some procedure and always
'         returns a value when the procedure ends.
'     (2) SUB-PROCEDURE
'         When a sub-procedure is called, it too performs some procedure, but it
'         does not return a value when the procedure ends.
'
'PUB and PRI are both procedures.  Since they both return values, they are a
'special type of procedure, a procedure called a "function procedure" or just
'a function.
'
'In early examples in this series of examples, PUB and PRI were called
'procedures.  I did this to keep the reader from looking for a return value
'which is necessary with a function.  There was a return value but it
'could not be seen by the reader. When you get the documentation, you
'can dive deeper into this subject. In example 10 and 11, there was
'an explicit return value.
'*****************************************************************************
'*****************************************************************************
'In the future, PUB and PRI will be referred to as METHODS (POSSIBLY).
'*****************************************************************************
'*****************************************************************************
'NOTE: No attempt was made to cover all the variations of the above terms.
'      Only Microsoft's thinking was given. Event procedures were not covered.
'*****************************************************************************
```

EXAMPLE 01 -- ONE LED BLINKS FIVE TIMES

```
'                                       EXAMPLE01
'
'                                   ONE LED BLINKS FIVE TIMES
'*********************************************************************************************
'IMPORTANT:  Please run this code and attempt to understand it before advancing to
'            Example02, Example03, etc.  In the future, examples will be added and each will
'            gradually increase in complexity and/or cover a different concept.
'*********************************************************************************************
'
'DIFFICULTY LEVEL: Very easy
'
'PURPOSE:
'  -- The purpose of this example is to demonstrate an extremely simple program that almost anyone
'     will be able to understand. This is a "Hello World" type of program.
'  -- Future examples will become increasingly more complex.
'  -- This program shows how to turn on and off a single LED at port A16 every second.  This
'     on-off sequence will be repeated five times.
'  -- Also, this program explains one way to use WaitCnt to pause program execution.
'  -- Future examples will build on each other; therefore, less internal documentation will be
'     needed.
'  -- Only one of the eight processors is used in this example.
'  -- IT IS ASSUMED THAT THE PROGRAMMER HAS OR IS READING THE PROPELLER'S EXTERNAL DOCUMENTATION
'     AND HAS SOME PREVIOUS PROGRAMMING EXPERIENCE WITH MICROCONTROLLERS. IT IS FURTHER
'     ASSUMED THAT ONE LED IS CORRECTLY CONNECTED TO PORT A16.  IF YOU HAVE THE DEVELOPMENT
'     KIT, THE CONNECTION WILL NOT BE A PROBLEM.
'  -- Every effort has been made to insure that the code in this and future examples
'     runs correctly.
'Submitted by Dave Scanlan, Feb 27, 2006
'File: Example01_BlinkingOneLED.spin
'*********************************************************************************************
'CORRECT OUTPUT: The LED at port A16 will blink every second, five times.
'*********************************************************************************************
'
'
CON
'Set clock speed to 80 MHz
  _clkmode = xtal1 + pll16x        'Sets the clock speed to 80 MHz using a times 16 multiplier
  _xinfreq = 5_000_000             'Crystal speed: 5 MHz
'
  High = 1                         'A CONstant used to set an output port to about 3 volts
  Low  = 0                         'A CONstant used to set an output port to about 0 volts
  Out = %1                         'A CONstant used to set a port's direction to Out
'
VAR
  Byte Pin                         'Declares Pin to be a global VARiable of a type Byte
'
PUB Start                          'A PUBlick procedure named
Start
  BlinkingLED                      'Calls the PRIvate procedure BlinkingLED
'


PRI BlinkingLED                    'A PRIvate procedure named BlinkingLED
  Pin := 16                        'Assigns 16 to the variable Pin
  DirA[Pin] := Out                 'Makes port A16 an output port
  Repeat 5                         'Repeats the code indented under it 5 times
    'TURN THE LED ON AND OFF
    ' -- The LED, at port A16, is on for 1/2 second and off for 1/2 second.
    ' -- The System Counter increments by one for every System Clock pulse. Thus, if the
    '    System Clock is running at 80MHz, the System Counter will increment 80 million times
    '    in one second; and every counter increment will take 12.5ns (1/80MHz).
    '    Each WaitCnt statement below is set to cause a 1/2 second wait:
    '    40_000_000 counter increments * 12.5ns = 500ms (1/2 second)
'
        OutA[Pin] := High          'LED ON
        WaitCnt(40_000_000 + Cnt) 'ONE-HALF SECOND WAIT
        OutA[Pin] := Low           'LED OFF
```

```
        WaitCnt(40_000_000 + Cnt) 'ONE-HALF SECOND WAIT
'
'INDENTION IS IMPORTANT IN SPIN: There are no ENDIFs, END REPEATs, END SUBs, NEXTs, etc.
```

EXAMPLE 02 -- HOW TO DECLARE A LOCAL VARIABLE AND CALL MORE THAN ONE PROCEDURE

```
'                                       EXAMPLE02
'
'                HOW TO DECLARE A LOCAL VARIABLE AND CALL MORE THAN ONE PROCEDURE.
'*********************************************************************************************
'IMPORTANT: Please run or examine closely previous examples before running this code. If you do
'           this, you are more likely to understand this example.
'*********************************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'
'   LOCAL VARIABLES: This example demonstrates how to declare a local variable. A local variable
'                    is only known within the procedure in which it is declared.  Also, it is
'                    is only active while the code in that procedure is being run.
'   TWO PROCEDURES:  This example demonstrates calling more than one procedure.
'*********************************************************************************************
'
'DIFFICULTY LEVEL: Very easy
'
'PURPOSE:
'  -- The purpose of this example is to demonstrates how to declare a local variable
'     within a procedure, and to demonstrate how to call two procedures. One procedure, when
'     called, will cause the LED at port A16 to blink five times. This will be followed by a call
'     to a second procedure that will cause the LED at port A17 to blink five times.
'
'  -- Note that one sequence of five blinks is followed in time by another sequence of five blinks.
'     This sequential behavior occurs because only one processor is executing the code. In the next
'     example, EXAMPLE03, both sequences of blinks will occur at the SAME TIME because we will use
'     two processors.  One processor will handle one sequence of five blinks and at the same
'     time a second processor will handle a second sequence of five blinks.
'
'ADDITIONAL INFORMATION:
'  -- The comments in the code are typically used to explain new statements or new concepts not
'     explained in previous examples.  In order to eliminate comment clutter, the code comments
'     in previous examples have usually been eliminated.
'  -- IT IS ASSUMED THAT THE PROGRAMMER HAS OR IS READING THE PROPELLER'S EXTERNAL DOCUMENTATION
'     AND HAS SOME PREVIOUS PROGRAMMING EXPERIENCE WITH MICROCONTROLLERS. IT IS FURTHER
'     ASSUMED THAT TWO LEDS ARE CONNECTED TO PORTS A16 AND A17.
'  -- Every effort has been made to insure that the code in all examples runs correctly.
'
'Submitted by Dave Scanlan, March 4, 2006
'File: Example02_LocalVarAndCallingTwoProcedures.spin
'*********************************************************************************************
'CORRECT OUTPUT: The LED at port A16 will blink five times.  This sequence of blinks will be
'                followed by the LED at port A17 blinking five times.
'
'                NOTE: The output from this example will be compared to the output in the next
'                      example where two processors will be running.
'*********************************************************************************************
CON
  _clkmode  = xtal1 + pll16x
  _xinfreq  = 5_000_000
'
  High = 1
  Low  = 0
  Out = %1
'
VAR
  'No global variables needed
'
PUB Start
  BlinkingLED_A16                   'This call is executed first.
  BlinkingLED_A17                   'This call is executed next but only after
                                    '   BlinkingLED_A16 has finished executing.
'
PRI BlinkingLED_A16 | Pin           '| Pin is declared as a local variable
                                    '  Pin is only known within
BlinkingLED_A16
  Pin := 16                         '  Reference to Pin stops when this procedure ends.
  DirA[Pin] := Out
'
  Repeat 5
   OutA[Pin] := High                'LED ON
   WaitCnt(40_000_000 + Cnt)        'ONE-HALF SECOND WAIT
```

```
   OutA[Pin] := Low                 'LED OFF
   WaitCnt(40_000_000 + Cnt)        'ONE-HALF SECOND WAIT
'
'
PRI BlinkingLED_A17 | Pin           '| Pin is declared as a local variable
                                    '  Pin is only known within
BlinkingLED_A17.
  Pin := 17                         '  Reference to Pin stops when this procedure ends.
  DirA[Pin] := Out
'
  Repeat 5
    OutA[Pin] := High               'LED ON
    WaitCnt(40_000_000 + Cnt)       'ONE-HALF SECOND WAIT
    OutA[Pin] := Low                'LED OFF
    WaitCnt(40_000_000 + Cnt)       'ONE-HALF SECOND WAIT
'
'INDENTION IS IMPORTANT IN SPIN.
```

EXAMPLE 03 -- USING TWO COGS (TWO PROCESSORS)

```
'                                          EXAMPLE03
'
'                                 USING TWO COGS (TWO PROCESSORS)
'*********************************************************************************************
'IMPORTANT: This example requires an understanding of examples 01 and 02.
'*********************************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'
'   COG: This is the term used by the designers (Parallax, Inc.) of the Propeller. It refers
'        to a processor in the Propeller chip. This chip has eight Cogs, or processors.
'*********************************************************************************************
'
'DIFFICULTY LEVEL: Easy
'
'PURPOSE:
'  -- The purpose of this example is to demonstrate the use of two Cogs (two processors)
'     running separate procedures in PARALLEL.
'
'ADDITIONAL INFORMATION:
'  -- The two procedures in this example have the same function as in Example 02; that is, the
'     procedures turn on and off an LED every second. However, in this example the procedures
'     run in PARALLEL because each is run by a different processor; consequently, the
'     two LEDs will blink in sync. In Example 02, the two procedures were run sequentially
'     using one processor.
'
'Submitted by Dave Scanlan, March 4, 2006
'File: Example03_UsingTwoCogs.spin
'*********************************************************************************************
'CORRECT OUTPUT: The LEDs at A16 and A17 should blink on and off every second. Both should
'                blink on and off at the same time.
'*********************************************************************************************
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000
'
  High = 1
  Low  = 0
'
VAR
  long  stack0[20]                  'Sets up a stack space for a Cog(processor)
  long  stack1[20]                  'Sets up a stack space for a second Cog(processor)
                                    'long stack0[20]: Allocates 20 longs for the stack.
                                    'long stack1[20]: Allocates 20 longs for the stack.
PUB Start
  cognew(BlinkingLED_A16, @stack0)  'Starts a Cog, calls BlinkingLED_A16, the Cog uses @stack0
  cognew(BlinkingLED_A17, @stack1)  'Starts a Cog, calls BlinkingLED_A17, the Cog uses @stack1
'
PRI BlinkingLED_A16 | Pin
  Pin := 16
  DirA[Pin] := %1
'
  Repeat 5
    OutA[Pin] := High
    WaitCnt(40_000_000 + Cnt)
    OutA[Pin] := Low
    WaitCnt(40_000_000 + Cnt)
'
PRI BlinkingLED_A17 | Pin
  Pin := 17
  DirA[Pin] := %1
'
  Repeat 5
    OutA[Pin] := High
    WaitCnt(40_000_000 + Cnt)
    OutA[Pin] := Low
    WaitCnt(40_000_000 + Cnt)
'
'INDENTION IS IMPORTANT IN SPIN CODE
```

EXAMPLE 04 -- DISPLAYS TEXT ON A VIDEO MONITOR

```
'                                          EXAMPLE04
'
'                                DISPLAYS TEXT ON A VIDEO MONITOR
'*********************************************************************************************
'IMPORTANT: This example requires an understanding of examples 01, 02, and 03.
'*********************************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'
'   TV_Terminal: This is an object that generates the signals necessary to drive a video monitor.
'
'   VideoDisplay: An instance of the object TV_Terminal.
'
'   Repeat Count From 1 To 5: This is a loop that will increment the variable, Count, from 1 to 5.
'                             The code under it is repeated five times.
'*********************************************************************************************
'DIFFICULTY LEVEL: Easy
'
'PURPOSE:
'  -- Demonstrates how to send text to a video monitor, and how to set the foreground and
'     background colors of a video monitor to white-on-blue.
'
'ADDITIONAL INFORMATION:
'  -- The video signal output is a composite signal.
'  -- TV_Terminal is an object call TV_Terminal.spin found in the "Propeller Tool" subdirectory.
'  -- Any name could have been used instead of "VideoDisplay"
'  -- This examples requires an elementary knowledge of OOP.
'  -- This example assumes you have the development kit or a schematic of it.
'     You will need one of these to generate a composite video out.
'
'Submitted by Dave Scanlan, March 3, 2006
'File: Example04_VideoOutput.spin
'*********************************************************************************************
'CORRECT OUTPUT: A text string and an incrementing decimal number will be displayed five times.
'*********************************************************************************************
'
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000
  NewLine = 13
  ClearScreen = 0
 '
OBJ
  VideoDisplay: "TV_Terminal"                   'Creates the object VideoDisplay
'
PUB Start
'
  DisplayTextOnMonitor
'
PRI DisplayTextOnMonitor | Count
  VideoDisplay.start                            'Initializes the VideoDisplay object
  SetScreenWhiteOnDarkBlue                      'Calls a procedure that sets the foreground
                                                '  color to white and the background color
                                                '  to dark blue.
'
  Repeat Count From 1 To 5                      'For every loop, Count increments by one.
    WaitCnt(40_000_000 + Cnt)
    VideoDisplay.str(string("THE COUNT IS: "))  'Sends a text string to the video monitor.
    VideoDisplay.dec(Count)                     'Sends a numeric value to a video monitor.
    VideoDisplay.out(NewLine)                   'Sends a NewLine to a video monitor.
    WaitCnt(40_000_000 + Cnt)
  VideoDisplay.out(ClearScreen)                 'Clear screen.
'
PRI SetScreenWhiteOnDarkBlue                    'Sets the foreground color to white and the
  VideoDisplay.out(3)                           '  background color to dark blue on the monitor.
  VideoDisplay.out(5)
'
'INDENTION IS IMPORTANT IN SPIN:  Spin does not use Ends in its blocks of code.   For
'                                 example, there is no End Repeat after the Repeat loop and
'                                 you must indent after CON, OBJ, PUB, PRI, and DAT.
'
```

EXAMPLE 05 -- USING TWO COGS (PROCESSORS) WITH ARGUMENTS AND PARAMETERS

```
'                                          EXAMPLE 05
'
'                        USING TWO COGS(PROCESSORS) WITH ARGUMENTS AND PARAMETERS
'*********************************************************************************************
'IMPORTANT: This example may require an understanding of examples 01, 02, 03, AND 04
'*********************************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'
'   ARGUMENTS/PARAMETERS:  Arguments pass values to their corresponding Parameters.
'                          Note: Some programmers use the word "parameter" to refer
'                                to both the argument and its corresponding parameter.
'                                This programmer does not.  Having seperate names makes
'                                it easier to refer to them in the documentation.
'*********************************************************************************************
'
'DIFFICULTY LEVEL: Easy
'
'PURPOSE:
'  -- The purpose of this example is to demonstrate the use of Arguments and Parameters when
'     two Cogs(processors) are running in parallel.
'
'ADDITIONAL INFORMATION:
'  -- Two procedures run in parallel as they did in EXAMPLE 03. This time, however,
'     the WaitCnt time is set by passing two different values to the two procedures by using
'     arguments and parameters.
'
'     The Argument (WaitPeriod_Arg) passes 8_000_000 to its corresponding parameter
'     (WaitPeriod_Par) in one procedure, and 80_000_000 in the other procedure.
'     The result of this is that one LED will blink on and off every 0.2 seconds and
'     the other every 2.0 seconds.
'
'Submitted by Dave Scanlan, March 5, 2006
'File: Example05_TwoCogsArgParGlobalVar.spin
'*********************************************************************************************
'CORRECT OUTPUT: The LED at A16 will blink on and off every 0.2 seconds and the LED at A17 will
'                blink on and off every 2.0 seconds.  The blinking will occur in parallel, but
'                the LED at A16 will finish ten times faster. Both LEDs will blink five times.
'*********************************************************************************************
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000
'
  High = 1
  Low  = 0
'
VAR
  Long Stack0[20]                           'Sets up a stack space for a Cog(processor)
  Long Stack1[20]                           'Sets up a stack space for a second Cog(processor)
                                            'long Stack0[20]: Allocates 20 longs for the stack.
                                            'long Stack1[20]: Allocates 20 longs for the stack.
'
  Long WaitPeriod_Arg                       'Arg in WaitPeriod_Arg stands for Argument.
                                            'Sets the rate at which the LED blinks.
'
'
PUB Start
'
  WaitPeriod_Arg := 8_000_000               '0.1 second wait period
'
 'Starts a New Cog(Calls BlinkingLED_A16 (WaitPeriod Argument),This Cog uses @stack0)
  CogNew(BlinkingLED_A16 (WaitPeriod_Arg), @Stack0)

  WaitPeriod_Arg := 80_000_000              '1.0 second wait period

 'Starts a New Cog(Calls BlinkingLED_A17 (WaitPeriod Argument), This Cog uses @stack1)
  CogNew(BlinkingLED_A17 (WaitPeriod_Arg), @Stack1)
'
'
'LED Blinks Fast
PRI BlinkingLED_A16 (WaitPeriod_Par) | Pin  'Par in WaitPeriod_Par stands for Parameter.
  Pin := 16
  DirA[Pin] := %1
```

```
'
  Repeat 5
    OutA[Pin] := High
    WaitCnt(WaitPeriod_Par + Cnt)            '0.1 second wait period
    OutA[Pin] := Low
    WaitCnt(WaitPeriod_Par + Cnt)            '0.1 second wait period
'
'
'LED Blinks Slowly
PRI BlinkingLED_A17 (WaitPeriod_Par)| Pin    'Par in WaitPeriod_Par stands for Parameter.
  Pin := 17
  DirA[Pin] := %1
'
  Repeat 5
    OutA[Pin] := High
    WaitCnt(WaitPeriod_Par + Cnt)            '1.0 second wait period
    OutA[Pin] := Low
    WaitCnt(WaitPeriod_Par + Cnt)            '1.0 second wait period


'INDENTION IS IMPORTANT IN SPIN CODE.
'FOR THE ADVANCED LEARNER:
'  -- Arguments and Parameters are used to help eliminate tightly coupled code. In other words,
'     procedures of code can pass values to each other without using global variables.
'     The use of global variables to pass values between procedures is call Common Coupling,
'     and this we try to avoid. (Use Google Search Words: Common Coupling Structured Design)
'     Because the Argument (WaitPeriod_Arg) was declared as a global variable, two
'     different names were required for an argument and its corresponding parameter. More
'     on this in EXAMPLE 06.)
'  -- In order to keep this example simple, a global variable is used; but in EXAMPLE 06
'     we will show you how to eliminate this global variable by  using local variables
'     instead.
'  -- In Spin, Arguments pass values to their corresponding Parameters "by value".
```

EXAMPLE 06 -- SAME AS EXAMPLE 05, BUT WITH A LOCAL VARIABLE AS THE ARGUMENT

```
'                                       EXAMPLE 06
'
'                   SAME AS EXAMPLE 05, BUT WITH A LOCAL VARIABLE AS THE ARGUMENT
'*********************************************************************************************
'IMPORTANT: This example WILL require an understanding of examples 01, 02, 03, 04, and 05
'*********************************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'   LOCAL VARIABLE AS AN ARGUMENT: In EXAMPLE 05, the argument was a global variable. In this
'                                  example the argument is a local variable.
'
'                                  If a global variable is used as an argument, the exact same
'                                  name cannot be used as the name for the argument's
'                                  corresponding parameter.  This is why two different
'                                  names were used in EXAMPLE 05. (See EXAMPLE 05)
'
'                                  In this example, we do not need to make the names different
'                                  because the argument is declared as a local variable.
'                                  Thus, WaitPeriod is used for the argument and the parameter.
'                                  Having the same name makes tracing through the code with
'                                  nested procedural calls, much easier to follow.
'
'   NOTE:  IN ANY COMPUTER LANGUAGE IT IS ALWAYS BEST TO USE A LOCAL VARIABLE WHEN POSSIBLE TO
'          CONTROL COUPLING BETWEEN PROCEDURES.  THE PROGRAMMER SHOULD KEEP THE PROCEDURES IN
'          ANY PROGRAM LOOSELY COUPLED, NOT TIGHTLY COUPLED.  GLOBAL VARIABLES CAUSE TIGHTLY
'          COUPLED PROCEDURES AND LEAD TO ERRORS THAT ARE DIFFICULT TO TRACE.
'
'*********************************************************************************************
'
'DIFFICULTY LEVEL: INTERMEDIATE
'
'PURPOSE:
'  -- The purpose of this example is to show ONE of the benifits for using a local
'     variable as an argument.
'
'ADDITIONAL INFORMATION:
'  -- EXAMPLE 05 where a global variable was used for the argument
'       ARGUMENT NAME                  PARAMETER NAME
'         WaitPeriod_Arg                 WaitPeriod_Par
'
'  -- EXAMPLE 06 where a local variable was used for the argument
'     ARGUMENT NAME                   PARAMETER NAME
'       WaitPeriod                     WaitPeriod
'
'Submitted by Dave Scanlan, March 5, 2006
'File: Example05_TwoCogsArgParLocalVar.spin
'*********************************************************************************************
'CORRECT OUTPUT: The LED at A16 will blink on and off every 0.2 seconds and the LED at A17 will
'                blink on and off every 2.0 seconds.  The blinking will occur in parallel, but
'                the LED at A16 will finish ten times faster. Both LEDs will blink five times.
'*********************************************************************************************
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000
'
  High = 1
  Low  = 0
'
VAR
  Long Stack0[20]                       'Sets up a stack space for a Cog(processor)
  Long Stack1[20]                       'Sets up a stack space for a second Cog(processor)
                                        'long Stack0[20]: Allocates 20 longs for the stack.
                                        'long Stack1[20]: Allocates 20 longs for the
stack.

PUB Start  | WaitPeriod                 'WaitPeriod sets the rate at which the LED blinks.
'
  WaitPeriod := 8_000_000               '0.1 second wait period

 'Starts a New Cog(Calls BlinkingLED_A16 (WaitPeriod argument), This Cog uses @stack0)
  CogNew(BlinkingLED_A16 (WaitPeriod), @Stack0)

  WaitPeriod := 80_000_000              '1.0 second wait period
```

```
 'Starts a New Cog(Calls BlinkingLED_A17 (WaitPeriod argument), This Cog uses @stack1)
  CogNew(BlinkingLED_A17 (WaitPeriod), @Stack1)
'
'
'LED Blinks Fast
PRI BlinkingLED_A16 (WaitPeriod) | Pin    'Par in WaitPeriod_Par stands for Parameter.
  Pin := 16
  DirA[Pin] := %1
'
  Repeat 5
    OutA[Pin] := High
    WaitCnt(WaitPeriod + Cnt)              '0.1 second wait period
    OutA[Pin] := Low
    WaitCnt(WaitPeriod + Cnt)              '0.1 second wait period
'
'
'LED Blinks Slowly
PRI BlinkingLED_A17 (WaitPeriod)| Pin      'Par in WaitPeriod_Par stands for Parameter.
  Pin := 17
  DirA[Pin] := %1
'
  Repeat 5
    OutA[Pin] := High
    WaitCnt(WaitPeriod + Cnt)               '1.0 second wait period
    OutA[Pin] := Low
    WaitCnt(WaitPeriod + Cnt)               '1.0 second wait period
'INDENTION IS IMPORTANT IN SPIN CODE.
'FOR THE ADVANCED LEARNER:
'  -- Arguments and Parameters are used to help eliminate tightly coupled code.
'     The use of global variables to pass values between procedures is call Common
'     Coupling, and this we try to avoid.
'  -- Global variables have a strong potential to cause serious errors which are very
'     difficult to locate.
'  -- A FULL EXPLANATION OF GLOBAL VARIABLE EFFECTS WOULD REQUIRE A TWO-HOUR LECTURE...OR LONGER.
'     I suggest a Google search using these words: Common Coupling Structured Design
'  -- Arguments pass values to their corresponding parameters "by value" in Spin.
```

EXAMPLE 07 -- CHECKS FOR A HIGH ON PIN A1; HAS VIDEO OUTPUT

```
'                                          EXAMPLE07
'
'                             CHECKS FOR A HIGH ON PIN A1; HAS VIDEO OUTPUT
'*****************************************************************************************
'IMPORTANT: This example requires an understanding of examples 01 and 03.
'*****************************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'
'   If-Else statement: This statement operates the same as If-Else statements in most
'                      other languages.  Be careful because Spin is indention sensitive.
'                      Note the indention in the code.
'
'   DirA[1] := In      Sets the direction of Pin A1 to Input. BE CAREFUL TO NOT EXCEED
'                      3.3 VOLTS.  (A maximum input voltage has not been established at
'                      this time.)
'
'   VideoDisplay.Out(3): This combination of statements will produce white on red.
'   VideoDisplay.Out(7)
'
'   Repeat statement: "Repeat" in this example is an infinite loop. This causes the program
'                     to constantly check for a High on Pin A1. You must indent
'                     properly for this to work as desired.
'*****************************************************************************************
'DIFFICULTY LEVEL: Very Easy
'
'MAIN PURPOSE:
'  -- Demonstrates how to check for a High on an Input Pin.
'
'ADDITIONAL INFORMATION:
'  -- The video signal output is a composite signal.
'  -- TV_Terminal is an object call TV_Terminal.spin found in the "Propeller Tool"
'     subdirectory.
'  -- Any name could have been used instead of "VideoDisplay"
'  -- This example assumes you have the development kit or a schematic of it.
'     You will need one of these to generate a composite video out.
'  -- USE A 10K PULL-DOWN RESISTER TO KEEP PIN A1 FROM FLOATING: Place a 10K resistor
'     between Pin A1 and Vss.
'  -- This example only uses one Cog.
'
'Submitted by Dave Scanlan, March 8, 2006
'File: Example07_ReadingAnInputPort.spin
'*****************************************************************************************
'CORRECT OUTPUT: If Pin A1 is High the following happens:
'                  (1) The message "INPUT AT PIN A1" is repeatedly displayed on the
'                      monitor.
'                  (2) The monitor's foreground and background become white-on-blue.
'                  (3) The LED at A16 is turned on.
'
'                If Pin A1 is Low the following happens:
'                  (1) The message "NO INPUT" is repeatedly displayed on the monitor.
'                  (2) The monitor's foreground and background become white-on-red.
'                  (3) The LED at A16 is turned off.
'
'NOTE: USE A 10k PULL-DOWN RESISTER TO KEEP PIN A1 FROM FLOATING: Place a 10K resistor
'      betweein Pin A1 and Vss.
'*****************************************************************************************
'
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000
  NewLine = 13
  High = 1
  Low = 0
  Out = %1
  In = %0
'
VAR
'  No gobal variables in this program
'
OBJ
  VideoDisplay: "TV_Terminal"              'Creates the object VideoDisplay
'
PUB Start                                  'Any name will do here: Start, Begin, Main, etc.
```

```
  VideoDisplay.Start                          'Initializes the VideoDisplay
object
  CheckForInput
'
PRI CheckForInput
'
  DirA[1]  := In                              'Sets Pin A1 to Input
  DirA[16] := Out                             'Sets Pin A16 to Output
'
  Repeat                                      'An infinite loop. Indention is important
'
    If InA[1] == High                         'Be sure to indent under the If
      SetScreenToWhiteOnDarkBlue
      VideoDisplay.str(String("INPUT AT PIN A1"))
      OutA[16] := High
    Else                                      'Be sure to indent under the Else
      SetScreenToWhiteOnRed
      VideoDisplay.str(String("NO INPUT"))
      OutA[16] := Low
'
    VideoDisplay.out(NewLine)
'
PRI SetScreenToWhiteOnDarkBlue                'This combination of Out's sets the monitor
  VideoDisplay.Out(3)                         ' to display white-on-blue
  VideoDisplay.Out(5)
'
PRI SetScreenToWhiteOnRed                     'This combination of Out's sets the monitor
  VideoDisplay.Out(3)                         ' to display white-on-red.
  VideoDisplay.Out(7)
'
'
'INDENTION IS IMPORTANT IN SPIN.
'
```

EXAMPLE 08 -- DISPLAYS KEYBOARD CHARACTERS ON A VIDEO MONITOR

```
'                                     EXAMPLE 08
'
'                        DISPLAYS KEYBOARD CHARACTERS ON A VIDEO MONITOR
'*********************************************************************************
'IMPORTANT: This example requires an understanding of example 04.
'*********************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'   Keyboard_iso: This is a Spin object that must be used in order to use
'                 GetKey and Present
'   GetKey: GetKey will return the code for any key pressed on the keyboard.
'           Using GetKey, you can display characters on a video monitor.
'   Present: Returns a TRUE if a keyboard is connected.
'*********************************************************************************
'DIFFICULTY LEVEL: Very Easy
'
'MAIN PURPOSE:
'  -- Demonstrates how to interface the keyboard to the Propeller and how to
'     display a key press on a video monitor.
'
'ADDITIONAL INFORMATION:
'  -- A PS/2 keyboard must be properly connected to the Properller.
'  -- This example assumes you have the development kit or a schematic of it.
'  -- This example only uses one Cog (processor).
'
'Submitted by Dave Scanlan, March 12, 2006
'File: Example08_KeyboardInput.spin
'*********************************************************************************
'CORRECT OUTPUT: Keyboard characters will be displayed on a video monitor.
'*********************************************************************************
'
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000
  NewLine = 13
  PS2ToProperllerPins_Setup = 6  'See number (1) at the bottom.
  KeyLocks_Setup = %0_000_010    'CapsLock ON: See number (2) at the bottom.
' KeyLocks_Setup = %0_000_000    'CapsLock OFF: See number (2) at the bottom.
  AutoRepeat_Setup = %01_01000   'See number (3) at the bottom.
'
OBJ
  VideoDisplay: "TV_Terminal"
  KB : "Keyboard_iso"            'Creates an instance of Keyboard_iso called KB
'
PUB Start
'
  VideoDisplay.start
  SetScreenToWhiteOnDarkBlue
' KB.Startx(PS/2 pins setup, CapsLock ON, .5second delay and 15cps key repeat rate)
  KB.Startx(PS2ToProperllerPins_Setup, KeyLocks_Setup, AutoRepeat_Setup)
  WaitCnt(50_000_000 + Cnt)          'Gives KB.Present time to find the keyboard.
'
  If KB.Present                      'Present is true if a keyboard is connected
    VideoDisplay.str(string("KEYBOARD CONNECTED"))
    VideoDisplay.Out(NewLine)
    Repeat
      VideoDisplay.out(KB.GetKey)    'Waits for keypress. Displays character on
                                     ' the video monitor.
  Else
    VideoDisplay.str(string("ERROR: KEYBOARD NOT CONNECTED"))
'
PRI SetScreenToWhiteOnDarkBlue
  VideoDisplay.out(3)
  VideoDisplay.out(5)

'ADDITIONAL INFORMATION:
'*********************************************************************************
'(1) 6 is a setup code that will properly match the pins on a PS/2 male
'    conector with a PS/2 female connector that is connected to the
'    the Propeller.  See Keyboard_iso.spin for more pin setup codes.
'(2) %0_000_010 is a setup code that will set CapsLock ON (bit 1). See
'    Keyboard_iso for more KeyLock setup settings, such as NumLock and
'    ScrollLock.
'(3) %01_01000 is a setup code that will produce a .5sec (bit 5) delay before
```

```
'     auto repeat starts and a 15cps repeat rate (bit 3). See Keyboard_iso
'     for more setup settings.
'*************************************************************************
```

EXAMPLE 09 -- DEBUGGING USING A VIDEO MONITOR

```
'                                       EXAMPLE 09
'
'                               DEBUGGING USING THE VIDEO MONITOR
'*******************************************************************************
'IMPORTANT: This example requires an understanding of example 04.
'*******************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'
'  VideoDisplay.Dec(Value):  Dec(Value) will output any Byte, Word, or Long
'                            as a decimal value.
'
'  VideoDisplay.Hex(Value,#OfDigits): Hex(Value,#OfDigits) will output any
'                                     Byte, Word, or Long in a Hex format
'                                     using a specified number of digits.
'
'  VideoDisplay.Bin(Value,#OfDigits): Bin(Value,#OfDigits) will output any
'                                     Byte, Word, or Long in a Binary format
'                                     using a specified number of digits.
'*******************************************************************************
'DIFFICULTY LEVEL: Very easy
'
'PURPOSE:
'  -- Demonstrates how to display on a video monitor the value of any variable
'     during program execution.  It further demonstrates how to monitor the
'     state (HIGH or LOW) of any input port.
'
'Submitted by Dave Scanlan, March 13, 2006
'File: Example09_DebuggingUsingTheVideoMonitor.spin
'*******************************************************************************
'CORRECT OUTPUT: The values of three variables are displayed on the video
'                monitor in Decimal, Hex, and Binary format.
'                After 10 seconds, Pin A1 is monitored and its state is
'                displayed on the video monitor.
'*******************************************************************************
'
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000
  NewLine = 13
  Out = %1
  In = %0
'
VAR
  Word Value
'
OBJ
  VideoDisplay: "TV_Terminal"
'
PUB Start
  VideoDisplay.start             'Initialize VideoDisplay Object
  SetScreenToWhiteOnDarkBlue
  DirA[1] := In                  'Set port A1 to Input
'
 'DISPLAYS VALUE AS DECIMAL
  Value := 500
  VideoDisplay.Dec(Value)         'Displays a decimal value
  'Output on the video monitor:
  '500
  VideoDisplay.Out(NewLine)
'
 'DISPLAYS VALUE AS HEX
  Value := $FF
  VideoDisplay.Hex(Value,2)       'Displays a hex value using 2 digits
  'Output on the video monitor:
  'FF
  VideoDisplay.Out(NewLine)
'
 'DISPLAYS VALUE AS BINARY
  Value := %0001_0001
  VideoDisplay.Bin(Value,8)       'Displays a binary value using 8 digits
  'Output on the video monitor:
  '00010001
  VideoDisplay.Out(NewLine)
```

```
'
  WaitCnt(1_000_000_000 + Cnt)    'Keeps output on the screen for 10 seconds.
'
 'MONITORS Pin A1
  Repeat                          'Continuously monitors the state of pin A1
    VideoDisplay.Bin(InA[1],1)
    VideoDisplay.Out(NewLine)
    'Output on the video monitor:
    '0 or 1  depending on input state at pin A1
    '0 or 1  depending on input state at pin A1
    '0 or 1  depending on input state at pin A1
    '0 or 1  depending on input state at pin A1
    '0 or 1  depending on input state at pin A1
    '...
'
PRI SetScreenToWhiteOnDarkBlue
  VideoDisplay.out(3)
  VideoDisplay.out(5)
'
'
'ADDITIONAL INFORMATION:
'   -- The video signal output is a composite signal.
'   -- TV_Terminal is an object call TV_Terminal.spin found in the
'      "Propeller Tool" subdirectory.
'   -- Any name could have been used instead of "VideoDisplay"
'   -- This examples requires an elementary knowledge of OOP.
'   -- This example assumes you have the development kit or a schematic of it.
'      You will need one of these to generate a composite video out.
```

EXAMPLE 10 -- HOW TO CODE A FUNCTION THAT FINDS THE LARGEST OF TWO VALUES

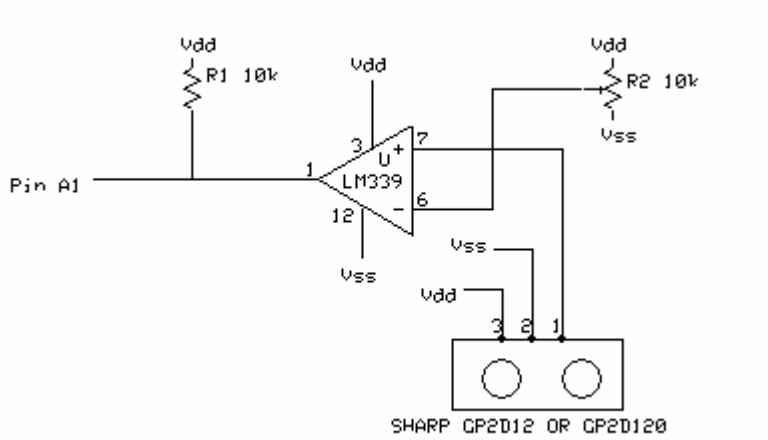
```
'
'
'                                       EXAMPLE 10
'
'             HOW TO CODE A FUNCTION THAT FINDS THE LARGEST OF TWO VALUES
'*********************************************************************************
'IMPORTANT: This example may require an understanding of examples 04, 05, and 06.
'*********************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'
'     FUNCTION: A function is a type of procedure that returns a value. A function
'               is often called a function procedure.
'
'*********************************************************************************
'DIFFICULTY LEVEL: Intermediate
'
'PURPOSE: This code demonstrates how to code a function.  This function will return
'         the largest of two values.
'
'Submitted by Dave Scanlan, March 15, 2006
'File: Example10_FunctionFindLargestVer1.spin
'*********************************************************************************
'CORRECT OUTPUT:  The largest value (10) will be displayed on the video monitor.
'*********************************************************************************
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000
'
VAR
'No global variables used.
'
OBJ
  VideoDisplay: "TV_Terminal"
'
PUB Start | X, Y, LargestValue        'X, Y, and LargestValue are Local variables.
  VideoDisplay.Start
  SetScreenToWhiteOnDarkBlue
'
  X := 5
  Y := 10
'
  LargestValue := FindTheLargest(X,Y) 'Calls the function and assigns the
'                                      result to "LargestValue."
  VideoDisplay.Dec(LargestValue)
'
PRI FindTheLargest (X,Y) | Largest     'Declares the function.
  If X > Y
    Largest := X
  Else
    Largest := Y
'
  Result := Largest                    '"Result" is a Spin reserved word and is the
'                                        location use to return the result of the
'                                        function.
PRI SetScreenToWhiteOnDarkBlue
  VideoDisplay.Out(3)
  VideoDisplay.Out(5)
'
'
'ADDITIONAL INFORMATION ON HOW THIS FUNCTION WORKS:
'  PRI FindTheLargest (X,Y)
'     -- This is the declaration of the function called "FindTheLargest".
'     -- The X and Y are parameters.
'     -- The Spin reserved word "Result" is a location used to return
'        the result of the function.
'
'  LargestValue := FindTheLargest(X,Y)
'    -- This statement calls the function and passes the arguments, X(5)and Y(10),
'       to their corresponding parameters in the function.
'    -- The code in the function is executed and the result is stored in
'       the location called "Result"
'    -- In the final step, the value stored in the location called "Result"
```

```
'       is assigned to "LargestValue" although this final step is not obvious.
'       A way to conceptualize the assignment is: LargestValue := Result.
```

EXAMPLE 11 -- A SECOND WAY TO CODE A FUNCTION: FIND THE LARGEST OF TWO VALUES (Ver. 2)

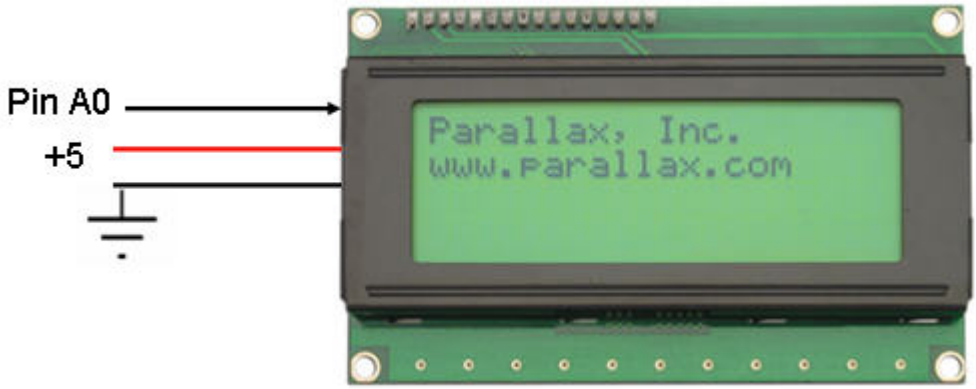
```
'
'                                     EXAMPLE 11
'
'     A SECOND WAY TO CODE A FUNCTION: FIND THE LARGEST OF TWO VALUES (Ver. 2)
'*********************************************************************************
'IMPORTANT: This example may require an understanding of examples 04,05,06 and 10
'*********************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'
'     FUNCTION W/O USING THE "RESULT" KEYWORD": In this example, we will return
'                                              a result without using the
'                                              keyword "Result".  We will make up
'                                              a word we want to use.
'*********************************************************************************
'DIFFICULTY LEVEL: Intermediate
'
'PURPOSE: This code demonstrates how to code a function.  This function will return
'         the largest of two values. The function will not use the keyword "Result."
'
'Submitted by Dave Scanlan, March 16, 2006
'File: Example11_FunctionFindLargestVer2.spin
'*********************************************************************************
'CORRECT OUTPUT:  The largest value (10) will be displayed on the video monitor.
'*********************************************************************************
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000
'
VAR
'No global variables used.
'
OBJ
  VideoDisplay: "TV_Terminal"
'
PUB Start | X, Y, LargestValue        'X, Y, and LargestValue are Local variables.
  VideoDisplay.Start
  SetScreenToWhiteOnDarkBlue
'
  X := 5
  Y := 10
'
  LargestValue := FindTheLargest(X,Y) 'Calls the function and assigns the
'                                      result to "LargestValue."
  VideoDisplay.Dec(LargestValue)
'
PRI FindTheLargest (X,Y) : Largest     'Largest returns the result. Note the ":".
  If X > Y
    Largest := X
  Else
    Largest := Y
'
PRI SetScreenToWhiteOnDarkBlue
  VideoDisplay.Out(3)
  VideoDisplay.Out(5)
'
'
'ADDITIONAL INFORMATION ON HOW THIS FUNCTION WORKS:
'  Rather than use the reserved word "Result" we can choose the word we
'  want. In the function declaration below "Largest" was used to return
'  the function's value.
'  PRI FindTheLargest (X,Y) : Largest
'
'  IMPORTANT: You must use the colon ":" before Largest.
'             We could have used any valid identifier.
'             Largest holds a pointer to the function's return value.
'
'IN ORDER TO KEEP THINGS SIMPLE, I HAVE ELIMINATED SOME OF THE DETAILS.  REFER
'TO THE DOCUMENTATION WHEN YOU GET IT.
```

EXAMPLE 12 -- PROXIMITY DETECTION USING THE SHARP GP2D12




```
'                                       EXAMPLE 12
'
'                           PROXIMITY DETECTION USING THE SHARP GP2D12
'*********************************************************************************************
'IMPORTANT: This example may require an understanding of Example 01
'*********************************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'
'  CONCEPT NAME: Proximity Detection
'                - This program uses the Sharp GP2D12 which will detect objects at a range
'                  of 10 to 80 cm. The device uses IR for object detection.
'                  -  This device is often used in robotics to sense the presence of objects.
'                  -  This device is inexpensive, but it is non-linear.  See resource below.
'                  -  Resource: www.acroname.com/robotics/info/articles/sharp/sharp.html#s1
'                  -  Refer to the schematic above for the physical interface of the
'                     Sharp GP2D12 to the Propeller.
'*********************************************************************************************
'DIFFICULTY LEVEL:  Very Easy
'
'PURPOSE:  The purpose of this program is to show how to connect a proximity detector to
'          Propeller and how to write the code for the interface.
'
'Submitted by Dave Scanlan, April 17, 2006
'File: Example12_IR.spin
'*********************************************************************************************
'CORRECT OUTPUT:  When an object is detected, the LED at A16 will turn on, and it will stay on
'                 until the object is no longer detected.
'*********************************************************************************************
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000
'
  High = 1
  Low  = 0
  Out = %1
  In = %0
'
VAR
  Byte Pin
'
PUB Start
  ProximitySensor
'
PRI ProximitySensor
```

```
  Pin := 16                             'LED's location is A16.
  DirA[Pin] := Out
  Pin := 1                              'The input from LM339 (voltage comparitor).
  DirA[Pin] := In
'
  Repeat                                'Infinite loop
    If InA[1] == High
      OutA[16] := High
    else
      OutA[16] := Low

'*****************************************************************************************************
'ADDITIONAL INFORMATION:
'   -- Set the variable resistor about midway and it will detect an object at
'      a distance of about 14 cm. Experiment with other settings to get the
'      detection distance you want.
'   -- The Sharp GP2D120 will allow for detection closer to the sensor.  The
'      range is 4 to 30 cm. This program and hookup will work fine for this sensor, also.
```



(Serial LCD used in this example)

EXAMPLE 13 -- USING AN LCD TO DISPLAY COG NUMBER, CLOCK FREQUENCY, AND CTN VALUES

```
'                                     EXAMPLE 13
'
'       USING AN LCD TO DISPLAY COG NUMBER, CLOCK FREQUENCY, AND CTN VALUE
'*********************************************************************************
'IMPORTANT: This example will require an understanding of example 01
'*********************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'    GOGID:    Returns the current COG's ID (0-7)
'    CLKFREQ: Returns the current system's clock frequency
'    CNT:      Returns the current 32-bit System Counter Register value.  CNT is the
'              system counter that is incremented once every clock cycle.  In this
'              example, it is incremented 80 million times each second, and updated
'              on the LCD display every second.
'    LCD:      This example was tested on Parallax's serial 4x20
'              (4 lines by 20 characters) display with a backlight. (Stock#: 27979)
'*********************************************************************************
'DIFFICULTY LEVEL: VERY EASY
'*********************************************************************************
'PURPOSE:
'   -- The purpose of this example is to display "some" of the system values on a
'      4x20 LCD module. These values are: GOGID, CLKFREQ, and CNT.
'
'Submitted by Dave Scanlan, April 22, 2006
'File: Example13_DisplaySystemValues
'*********************************************************************************
'CORRECT OUTPUT: The LCD will display a title, the current cog's ID, the clock
'                frequency, and the CNT(System Counter Register value).
'                Note: The CNT value changes 80 million times each second. Its
'                      current value will be updated on the LCD panel after a
'                      one-second delay. See the LCD format below:
'
'                                     4x20 LCD Module
'                                |□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
'                                |       SYSTEM VALUES         |
'                                |   GOG ID:X                  |
'                                |   CLK FREQ:XXXXXXXX         |
'                                |   COUNTER:XXXXXXXXXXX       |
'                                |□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
'
'*********************************************************************************
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000
'
  LCD_Pin   = 0                         'The LCD is connected to pin A0.
  LCD_Baud  = 19_200                    'Baud
  LCD_Lines = 4                         'The number of lines on this LCD
'
```

```
  Off  = 0
  On   = 1
'
OBJ
  LCD: "debug_lcd"                        'Creates the object LCD from "debug_lcd"
'
VAR
'No Global Variables
'
PUB Start
  DisplaySystemValues
'
PRI DisplaySystemValues
'
  If LCD.Start(LCD_Pin, LCD_Baud, LCD_Lines)'Initialize the LCD object
    LCD.Cursor(Off)                       'Set cursor off
    LCD.Backlight(On)                     'Set backlight on
    LCD.Cls                               'Clear the LCD screen
'
    LCD.Str(string("   SYSTEM VALUES"))   'Print string on LCD
    LCD.NewLine                           'Start on the next line
    LCD.Str(string("COG ID:"))            'Print string on LCD
    LCD.Gotoxy(7, 1)                      'Send cursor to col 7 line 1
    LCD.DecF(COGID, 1)                    'Print GOGID as a signed decimal
    LCD.NewLine
    LCD.Str(string("CLK FREQ:"))
    LCD.Gotoxy(9, 2)
    LCD.DecF(CLKFREQ, 8)                  'Print CLKFREQ as a signed decimal
    LCD.NewLine
    LCD.Str(String("COUNTER:"))
'
    Repeat                                'Infinite loop
      LCD.Gotoxy(8, 3)
      LCD.DecF(CNT, 11)                    'Print current value of CNT
      WaitCNT(80_000_000 + CNT)           'Wait one second
'*****************************************************************************
'ADDITIONAL INFORMATION:
'See Jon William's example (Debug_Lcd Test (4x20 LCD))on this thread for more serial
'LCD options.
'Price of this LCD: $39.95
'On an LCD panel, the top line is line 0 and beginning column is column 0.
'
'FORMAT FOR DecF
' DecF(Value, Width)
'  Prints signed decimal value in fixed-width field
'MEMORY USAGE:
'   Program: 308 Longs
'   Variables: 19 Longs
'   Stack/Free: 7,861 Longs
```

(This reading closely corresponds to readings from an HP 5314A.)

EXAMPLE 14 -- PROPELLER FREQUENCY COUNTER: External Frequency Source

```
'                                     EXAMPLE 14
'               PROPELLER FREQUENCY COUNTER: External Frequency Source
'*********************************************************************************
'IMPORTANT: This example may require an understanding of Example: 13
'*********************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'          SPECIAL PURPOSE PROPELLER REGISTERS: CTRA, FRQA, AND PHSA.  Using these
'          registers, an incoming frequency source is sampled for one second. The
'          result is then displayed on a Parallax LCD.
'*********************************************************************************
'DIFFICULTY LEVEL: Intermediate
'*********************************************************************************
'PURPOSE: The purpose of this example is to show how to create a frequency counter
'          using the Propeller's special purpose registers and display a frequency
'          count in Hz on an LCD.
'Submitted by Dave Scanlan, April 25, 2006
'File: Example14_FreqCtrLCD_ExternalSource.spin
'*********************************************************************************
'CORRECT OUTPUT: After sampling an incoming digital wave, the result will be
'                displayed as below:
'                                    4x20 LCD Module
'                                |□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
'                                |          PROPELLER          |
'                                |      FREQUENCY COUNTER      |
'                                |                             |
'                                |      FREQ:XXXXXXXXXXXHz     |
'                                |□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
'*********************************************************************************
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000
  LCD_Pin   = 0                         'The LCD is connected to pin A0.
  LCD_Baud  = 19_200                    'Baud
  LCD_Lines = 4                         'The number of lines on this LCD
  In = %0
  On = 1
  Off = 0
```

```
'
VAR
 Long Frequency
'
OBJ
 LCD: "debug_lcd"                          'Creates the object LCD from "debug_lcd"
'
PUB Start
  Repeat
     MeasureFrequency
     DisplayFrequency
'
PUB MeasureFrequency | Pin
  Pin := 5                                  'Pulses are sampled on this pin.
  DirA[Pin] := In
  CTRA := 0                                 'Clear settings
  CTRA := (%01010 << 26 ) | (%001 << 23) | (0 << 9) | (Pin) 'Trigger to count rising
'                                                           edge on Pin A5
  FRQA := 1000                              'Count 1000 pulses on each trigger
  PHSA := 0                                 'Clear accumulated value
  WaitCNT( 80_000_000 + CNT )               'Wait for 1 second
  Frequency := PHSA / 1000                  'Calculate Freq based on the sampling
'                                            duration, 1000 ms (1 second)
PUB DisplayFrequency
' If LCD.Start(0,       19_200,   4        )
  If LCD.Start(LCD_Pin, LCD_Baud, LCD_Lines) 'Initialize the LCD object
    LCD.Cursor(Off)                         'Set cursor off
    LCD.Backlight(On)                       'Set backlight on
    LCD.Cls                                 'Clear the LCD screen
    LCD.Str(string("      PROPELLER"))      'Display string on LCD
    LCD.Newline
    LCD.Str(string("  FREQUENCY COUNTER"))  'Display string on LCD
    LCD.Gotoxy(2,3)                         'Gotoxy(Col,Row)
    LCD.Str(string("FREQ:"))                'Display string on LCD
    LCD.Gotoxy(7,3)
    LCD.DecF(Frequency, 8)                  'Display Freq. as a decimal.
    LCD.Gotoxy(15,3)
    LCD.Str(string("Hz"))                   'Display string on LCD
'*****************************************************************************
'ADDITIONAL INFORMATION:
'  --  The code for sampling an incoming wave is an adaptation from
'      Dr. Martin Hebel's original code in the object, BS2.Function.spin.
'  --  The results of this counter agree closely with the readings from an HP 5314A
'      frequency counter with deviations of 0 to 6 Hz at 2MHz.
'  --  For this example, the amplitude was 3 volts. Until, the max input voltage,
'      has been "firmly" established for the Propeller, I would proceed with
'      caution.  Remember, this processor is a 3.3VDC device.
'  --  For analog signals, a pre-conditioning Schmitt trigger is suggested.
'      Resource: http://en.wikipedia.org/wiki/Schmitt_trigger
'FORMAT FOR: LCD.DecF(Frequency, 8)
' LCD.DecF(Value, Field_Width)
'  Prints signed decimal value in fixed-width field
```

(This frequency reading is exactly the same reading using an HP 5314A)

EXAMPLE 15 -- TWO-PROCESSOR FREQUENCY COUNTER: Internal Frequency Source

```
'                                     EXAMPLE 15
'
'         TWO-PROCESSOR FREQUENCY COUNTER: Internal Frequency Source

'*********************************************************************************
'IMPORTANT: This example may require an understanding of examples 5, 6, 13, 14.
'*********************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'
'  INTERNAL FREQUENCY GENERATOR: Generates digital pulses at a rate of
'                                of 27,933Hz.  One cog (procssor) is
'                                used for this; that is, this Cog has
'                                no other function but to generate pulses.
'*********************************************************************************
'DIFFICULTY LEVEL: Intermediate
'*********************************************************************************
'PURPOSE: This example shows how to use two Cogs (processors). One processor
'         generates digital pulses at a frequency of 27,933Hz and the second
'         processor measures the pulses coming from the first processor and
'         displays the frequency of these pulses on a Parallax serial LCD.
'
'         Example 14 vs. Example 15
'         -- Example 14 used one processor; and the frequency source was from
'            an EXTERNAL source, such as a function generator.
'         -- Example 15 uses two processors, and the frequency source is from
'            an INTERNAL source; that is, from one of the two Cogs (processors).
'         Pin connections: Besides connecting the serial LCD, you must
'                          connect pin A6(Pulse output) to pin A5(Frequency
'                          counter input). Also, see Example 13 for the serial
'                          LCD connections to the Propeller.
'
'Submitted by Dave Scanlan, April 26, 2006
'File: Example15_FreqCtrLCD_InternalSource.spin
'*********************************************************************************
'CORRECT OUTPUT: After sampling the digital wave coming from one of the Cogs,
'                the result will be displayed as below:
'                                    4x20 LCD Module
'                                  |□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
'                                  |          PROPELLER          |
'                                  |      FREQUENCY COUNTER      |
'                                  |                             |
'                                  |      FREQ:   27933Hz        |
'                                  |□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
```

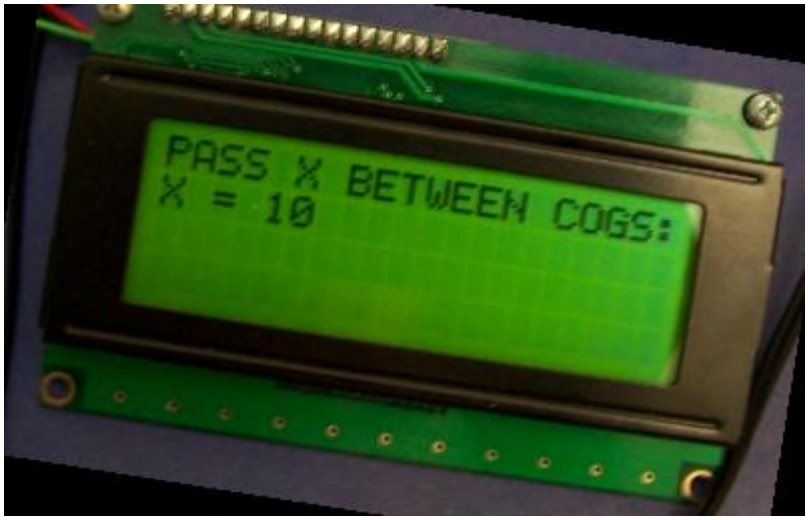
```
'*****************************************************************************
CON
  _clkmode  = xtal1 + pll16x
  _xinfreq  = 5_000_000
  LCD_Pin   = 0                         'The LCD is connected to pin A0.
  LCD_Baud  = 19_200                    'Baud
  LCD_Lines = 4                         'The number of lines on this LCD
  Out = %1
  In = %0
  On = 1
  Off = 0
  High = 1
  Low = 0
'
VAR
 Long Stack1[30]                         'Sets up a stack space for a one Cog
 Long Stack2[40]                         'Sets up a stack space for a second Cog
'
OBJ
 LCD: "debug_lcd"                        'Creates the object LCD. The file
'                                         "bebug_lcd" can be found on the Parallax
'                                          Web site under Propeller object downloads.
'                                          http://www.parallax.com/propeller/object.asp
PUB Start
  CogNew(FrequencyOut_PinA6, @Stack1)
  Cognew(MeasureFrequency_PinA5, @Stack2)
'
PRI MeasureFrequency_PinA5 | Pin, Frequency
' If LCD.Start(0,       19_200,   4        )
  If LCD.Start(LCD_Pin, LCD_Baud, LCD_Lines) 'Initialize the LCD object
    LCD.Cursor(Off)                         'Set cursor off
    LCD.Backlight(On)                       'Set backlight on
    LCD.Cls                                 'Clear the LCD screen
    LCD.Str(string("      PROPELLER"))      'Displays string on LCD
    LCD.Newline
    LCD.Str(string("  FREQUENCY COUNTER"))  'Displays string on LCD
    LCD.Gotoxy(2,3)                         'Gotoxy(Col,Row)
    LCD.Str(string("FREQ:"))                'Displays string on LCD
   Repeat
    Pin := 5
    DirA[Pin] := %0
    CTRA := 0                               'Clear CTRA settings
    CTRA := (%01010 << 26 ) | (%001 << 23) | (0 << 9) | (Pin) 'Trigger to count
'                                                             rising edge on A5
    FRQA := 1000                             'Count 1000 each trigger
    PHSA := 0                                'Clear accumulated value
    WaitCNT( 80_000_000 + CNT )              'Wait for 1 second
    Frequency := PHSA / 1000                 'Calculate Freq based on duration,
'                                             1000 ms (1 second)
    LCD.Gotoxy(7,3)
    LCD.DecF(Frequency, 8)                   'Displays freq in an 8 char field.
    LCD.Gotoxy(15,3)                         'Start on the next line
    LCD.Str(string("Hz"))                    'Displays string on LCD
'Generates a frequency of 27,933Hz. This seems to be the
' "top" speed for Spin, not so for assembly.
'Try other settings: WaitCNT(4_000_000 + CNT)is 10Hz; WaitCNT(400_000+ CNT) is 100Hz.
PRI FrequencyOut_PinA6 | Pin
  Pin := 6                                  'Output pin for 27,933Hz signal.
  DirA[Pin] := Out
  Repeat
    OutA[Pin] := High
    WaitCNT(400 + CNT)
    OutA[Pin] := Low
    WaitCNT(400 + CNT)
'*****************************************************************************
'ADDITIONAL INFORMATION:
'  --  The code for sampling an incoming wave is an adaptation from
'      Dr. Martin Hebel's original code in the object BS2.Function.spin.
'  --  The results of this example agree perfectly with the readings from an HP 5314A
'      frequency counter at a frequency of 27,933Hz.
'
'FORMAT FOR: LCD.DecF(Frequency, 8)
' LCD.DecF(Value, Field_Width)
'  Prints signed decimal value in fixed-width field
'
'PIN CONNECTIONS:
' (Input)PIN  A5•──────┤
' (Output)PIN A6•──────┤ Jumper wire from Pin A5 to Pin A6
```

```
' See Example 13 for LCD-to-Propeller pin connections.
'
'MEMORY USAGE:
'  Program: 324 Longs
'  Variables: 89 Longs
'  Stack/Free: 7,775 Longs
```

EXAMPLE 16 -- ARRAYS: Using an array and displaying array values on a serial LCD

```
'
'                                       EXAMPLE 16
'
'    ARRAYS: Using an array and displaying array values on an LCD
'
'*********************************************************************************
'IMPORTANT: This example may require an understanding of example 6
'*********************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'
'   Array
'     - How to declare an array: ArrayName[NumberOfElements]
'     - Example: Array[10] has 10 elements:  Array[0] thru Array[9]
'
'*********************************************************************************
'DIFFICULTY LEVEL: Easy
'
'PURPOSE: The purpose of this example is to show how to declare an array,
'         place values into an array, access values from an array, and
'         display these value on an LCD.
'
'Submitted by Dave Scanlan,  April 29, 2006
'File: Example16_ArrayLCD.spin
'*********************************************************************************
'CORRECT OUTPUT: The LCD will display the result below:
'
'                                       4x20 LCD Module
'                                |□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
'                                |  ARRAY VALUES:                  |
'                                |  0123456789                     |
'                                |  LARGEST VALUE:9                |
'                                |                                 |
'                                |□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
'
'*********************************************************************************
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000
  LCD_Pin   = 0                         'The LCD is connected to pin A0.
  LCD_Baud  = 19_200                    'Baud
  LCD_Lines = 4                         'The number of lines on this LCD
  On = 1
  Off = 0
'
VAR
  Byte Array[10]                         'All 10 elements are global
  Byte Largest                           'Largest is global
'
OBJ
  LCD: "Debug_LCD"                       'Create the LCD object
'                                         Be sure this program can find Debug_LCD.spin
```

```
PUB Start
  FillArray
  FindLargestValue
  DisplayValuesOnLCD
'
PRI FillArray | I, J                    ' "I" and "J" are local variables
  J := 0
  Repeat  I From 0 To 9                 'Fill array positions from 0 thru 9
    Array[I] := J
    J := J + 1
'
PRI FindLargestValue | I                ' "I" is a local variable
  Largest := Array[0]                   'Start the compares with the first element
  Repeat I From 1 to 9                  'Find the largest value in an array
    If Array[I] > Largest
      Largest := Array[I]
'
PRI DisplayValuesOnLCD | I
' If LCD.Start(0,       19_200,   4        )
  If LCD.Start(LCD_Pin, LCD_Baud, LCD_Lines) 'Initialize LCD
    LCD.Cursor(Off)
    LCD.Backlight(On)
    LCD.Cls
    LCD.Str(string("ARRAY VALUES:"))    'Display string on LCD
    LCD.NewLine
'
    Repeat I From 0 To 9
      LCD.DecF(Array[I],1)              'Display all array values
'                                        (Array[I],1): The "1" is the field width.
    LCD.NewLine
    LCD.Str(string("LARGEST VALUE:"))
    LCD.Gotoxy(14,2)                    'LCD.Gotoxy(Col,Row)
    LCD.DecF(Largest,1)                 'Display the Largest value
'                                        (Largest,1): The "1" is the field width.
'*****************************************************************************************
'ADDITIONAL INFORMATION:
'  --  With some languages, a declaration of Array[10] produces 11 locations:
'      Array[0],[1],[2],[3],[4],[5],[6],[7],[8],9],[10]
'  --  With Spin, a declaration of Array[10] produces exactly 10 locations:
'      A[0],A[1],A[2],A[3],A[4],A[5],A[6],A7],A[8],A[9].
'  --  MEMORY USAGE
'        Program: 308 Longs
'        Variables: 22 Longs
'        Stack/Free: 7,858 Longs
```



EXAMPLE 17 -- ONE WAY TO PASS A VALUE FROM COG TO COG AND INSUFFICIENT STACK SPACE

```
'                                     EXAMPLE 17
'
'                      ONE WAY TO PASS A VALUE FROM ONE COG TO ANOTHER COG
'                                AND INSUFFICIENT STACK SPACE

'*********************************************************************************
'IMPORTANT: This example may require an understanding of examples 6 and 14
'*********************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'
'    PASS A VALUE BETWEEN COGS
'      - In this example, the value of the variable X is passed from one Cog
'        to another Cog.
'    THE IMPORTANCE OF SUFFICIENT STACK SPACE FOR A COG.
'      - Without sufficient stack space for a Cog, the Cog will not function
'        correctly.

'*********************************************************************************
'DIFFICULTY LEVEL: Easy
'*********************************************************************************
'
'PURPOSE: The purpose of this example is to illustrate a procedure for passing
'         a value from one Cog to another Cog.  In this example the value of
'         variable X is passed from Cog0 to Cog1. These two Cogs will
'         communicate by sharing access to a common location in memory.
'         The location is X and it is a global variable.
'         In the code below you will reduce the stack space of a Cog by
'         10 longs and observe the results: The program will not run correctly.
'Submitted by Dave Scanlan,  May 3, 2006
'File: Example17_PassDataBetweenCogs.spin
'*********************************************************************************
'CORRECT OUTPUT: The LCD will display the result below:
'
'                                     4x20 LCD Module
'                                |□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
'                                | PASS X BETWEEN GOGS:       |
'                                | X = 10                     |
'                                |                            |
'                                |                            |
'                                |□□□□□□□□□□□□□□□□□□□□□□□□□□□□|

'*********************************************************************************
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000
  LCD_Pin   = 0
  LCD_Baud  = 19_200
  LCD_Lines = 4
  Off = 0
```

```
  On = 1

VAR
   Long Stack0[20]
   Long Stack1[30]
'  Long Stack1[20]                                If you use only 20 longs the
'                                                 program will not run correctly.
'                                                 CLEAR THE LCD, THEN TRY
'                                                 Stack1[20].
   Long X                                         'Location to be shared between
'                                                  two cogs.
OBJ
  LCD: "Debug_LCD"
PUB Start
  CogNew(SendsValue, @Stack0)
  CogNew(RecievesValueAndDisplaysIt, @Stack1)

PRI SendsValue                                   'Passes a value
  X := 10
PRI RecievesValueAndDisplaysIt                   'Recieves a value and displays
'                                                  it on an LCD panel.
  If LCD.Start(0,       19_200,   4        )
  If LCD.Start(LCD_Pin, LCD_Baud, LCD_Lines)     'Initialize LCD
    LCD.Cursor(Off)
    LCD.Backlight(On)
    LCD.Cls
    LCD.Str(string("PASS X BETWEEN COGS:"))
    LCD.Gotoxy(0,1)                          'LCD.Gotoxy(FirstCol,SecondRow)
    LCD.Str(string("X ="))
    LCD.Gotoxy(4,1)                          'LCD.Gotoxy(FifthCol,SecondRow)
    LCD.DecF(X,2)                            'Displays value: X

'ADDITIONAL INFORMATION:
'   -- Note that the locations on an LCD start with 0; that is, HOME is 0,0.
'   -- Several months ago, when I was beta testing the Propeller, I had trouble
'      getting the Cogs to communicate with each other. After some "wasted"
'      time I realized that I did not have enough stack space for one of my
'      cogs.  I hope this example will allow you to "skip" this error.
'MEMORY USAGE:
'   Program: 295 Longs
'   Variables: 70 Longs
'   Stack/Free: 7,823 Longs
```

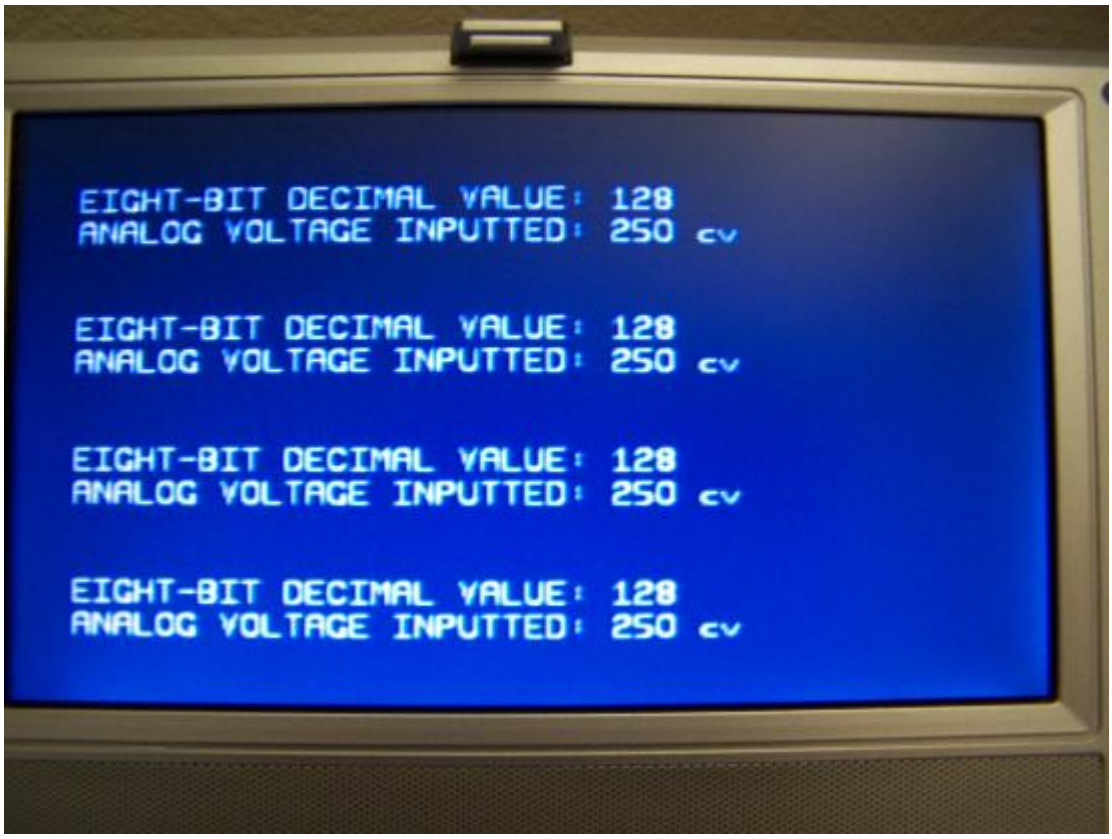
EXAMPLE 18 -- MENU-DRIVEN, SHIFT-RIGHT / SHIFT-LEFT DEMO: USES PS/2 KEYBOARD AND LCD PANEL

```
'                                    EXAMPLE 18
'
'                     MENU-DRIVEN, SHIFT-RIGHT AND SHIFT-LEFT DEMO:
'                           USES PS/2 KEYBOARD AND LCD PANEL
'*********************************************************************************
'IMPORTANT: This example may require an understanding of examples 8 and 14
'*********************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'    LCD MENUS
'      - In this example, an LCD will be used to display two menus.
'    KEYBOARD USED TO SELECT MENU CHOICES
'      - Using the keyboard, the user will be able to select menu choices.
'    SHIFT LEFT AND SHIFT RIGHT
'      - The bits in the variable X will be shifted left or right.
'      - X is only 8 bits to keep things simple.
'      - The shifting process will be displayed on the LCD in real time.
'    REPEAT-UNTIL and CASE
'      - Menu input is often constucted from these two control structures.
'*********************************************************************************
'DIFFICULTY LEVEL: Easy-to-intermediate for a beginner.
'*********************************************************************************
'PURPOSE: The purpose of this example is to illustrate:
'          1.  How to set up menus in Spin on an LCD.
'          2.  How to use the keyboard to make menu choices.
```

```
'          3.  How to shift bits in a variable while watching the process
'              in real time. The user can select how many bits to shift.
'Submitted by Dave Scanlan,  May 6, 2006
'File: Example18_ShiftLeftRight.spin
'*****************************************************************************
'CORRECT OUTPUT: The LCD will display the Menus below:
'
'                                    4x20 LCD Module
                                   |□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
'                                  |      BIT SHIFT DEMO       |
'                                  | R - Shift Right           |
'                                  | L - Shift Left            |
'                                  |>R<                        |
                                   |□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
'
'                                    4x20 LCD Module
'                                  |□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
'                                  |HOW MANY BITS? >7<          |
'                                  | BINARY X: 00000001        |
'                                  | DECIMAL X: 1              |
'                                  |Restart: Esc Key           |
'                                  |□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
'*****************************************************************************
'
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000
'
'KEYBOARD SETUP
  PS2ToProperllerPins_Setup = 6  'See number (1) at the bottom.
  KeyLocks_Setup = %0_000_010    'CapsLock ON: See number (2) at the bottom.
' KeyLocks_Setup = %0_000_000    'CapsLock OFF: See number (2) at the bottom.
  AutoRepeat_Setup = %01_01000   'See number (3) at the bottom.
'
'LCD SETUP
  LCD_Pin   = 0
  LCD_Baud  = 19_200
  LCD_Lines = 4
'
  Off = 0
  On = 1
  R = 82
  L = 76
'
VAR
 Byte X
 Byte NumOfBits                  'Number of bits to shift
 Byte ASCII_Key                  'Holds ASCII value from Keyboard
 Byte Right
 Byte Left
'
OBJ
  KB : "Keyboard_iso"             'Creates Keyboard (KB) object
  LCD: "Debug_LCD"                'Creates LCD object
'
PUB Start
  Initialize_KeyBoardAndLCD
  Display_ShiftRightOrLeftMenu
  Input_RightOrLeftChoice
  Repeat
    Display_NumOfBitsToShiftMenu
    Input_NumOfBitsToShift
    ShiftBitsInVariableX
'
'INITIALIZES KEYBOARD AND LCD
PRI Initialize_KeyBoardAndLCD
  KB.Startx(PS2ToProperllerPins_Setup, KeyLocks_Setup, AutoRepeat_Setup)
  WaitCnt(50_000_000 + Cnt)     'YOU MUST HAVE THIS DELAY TO INITIALIZE KB
' LCD.Start(0,       19_200,   4        )
  LCD.Start(LCD_Pin, LCD_Baud, LCD_Lines)
'
'DISPLAYS MENU FOR LEFT OR RIGHT SHIFT CHOICE
PRI Display_ShiftRightOrLeftMenu
  LCD.Cursor(On)
  LCD.Backlight(On)
  LCD.Cls
  LCD.Str(string("   BIT SHIFT DEMO"))
```

```
  LCD.Gotoxy(1,1)
  LCD.Str(string("R - Shift Right"))
  LCD.Gotoxy(1,2)
  LCD.Str(string("L - Shift Left"))
  LCD.Gotoxy(0,3)
  LCD.Str(string("> <"))
'
'INPUTS MENU DATA: (RIGHT OR LEFT SHIFT CHOICE)
PRI Input_RightOrLeftChoice
  REPEAT
    LCD.Gotoxy(1,3)             'LCD.Gotoxy(Col,Row)
    ASCII_Key := KB.GetKey
    Case ASCII_Key
      "R" : LCD.Str(string("R"))
            Right := True
            Left := False
      "L" : LCD.Str(string("L"))
            Left := True
            Right := False
  UNTIL ASCII_Key == R OR ASCII_Key == L
  WaitCNT(10_000_000 + CNT)
'
'DISPLAYS MENU FOR CHOOSING NUMBER OF SHIFT BITS
PRI Display_NumOfBitsToShiftMenu
  WaitCNT(80_000_000 + CNT)     'Delay while user reads display
  LCD.Cls
  LCD.Str(String("HOW MANY BITS? > <"))
  LCD.Gotoxy(1,1)
  LCD.Str(String("BINARY X:"))
  LCD.Gotoxy(1,2)
  LCD.Str(String("DECIMAL X:"))
  LCD.Gotoxy(0,3)
  LCD.Str(String("Restart: Esc Key"))
'
'INPUT MENU DATA: (NUMBER OF SHIFT BITS)
PRI Input_NumOfBitsToShift
  LCD.Gotoxy(16,0)
  ASCII_Key := KB.GetKey
  If ASCII_Key == $CB           'Looks for Esc Key ($CB)
      Start                     'Restarts program
  Case ASCII_Key
     "0" : LCD.Str(string("0"))
           NumOfBits := 0
     "1" : LCD.Str(string("1"))
           NumOfBits := 1
     "2" : LCD.Str(string("2"))
           NumOfBits := 2
     "3" : LCD.Str(string("3"))
           NumOfBits := 3
     "4" : LCD.Str(string("4"))
           NumOfBits := 4
     "5" : LCD.Str(string("5"))
           NumOfBits := 5
     "6" : LCD.Str(string("6"))
           NumOfBits := 6
     "7" : LCD.Str(string("7"))
           NumOfBits := 7
     "8" : LCD.Str(string("8"))
           NumOfBits := 8
    Other: Input_NumOfBitsToShift
'
'DISPLAYS SHIFTING BITS AND CHANGING DECIMAL VALUE
PRI ShiftBitsInVariableX
  X := %1111_1111
  LCD.Gotoxy(11,1)
  LCD.Bin(X,8)
  LCD.Gotoxy(12,2)
  LCD.DecF(X,3)
  LCD.ClrLn(3)
  WaitCNT(80_000_000 + CNT)     '1 second delay while user reads display
  Repeat NumOfBits
    If Right
      X := X >> 1               'Shift bits in X to the right.
    If Left
      X := X << 1               'Shift bits in X to the left.
    LCD.Gotoxy(11,1)
    LCD.Bin(X,8)
    LCD.Gotoxy(12,2)
```

```
    LCD.DecF(X,3)              'Decimal with a fixed field of size 3.
    WaitCNT(80_000_000 + CNT)  '1 second delay while user reads display
  WaitCNT(80_000_000 + CNT)    '1 second delay while user reads display
'*****************************************************************************
'ADDITIONAL INFORMATION:
'  --  This example will help a beginning programmer visualize what happens
'      when bits are shifted.
'  --  You will need a PS/2 keyboard and a serial LCD (4x20)
'  --  The code was written to make the application easy to understand.
'
'KEYBOARD SETTINGS EXPLAINED (SEE CONSTANT SECTION ABOVE.)
'(1) 6 is a setup code that will properly match the pins on a PS/2 male
'    conector with a PS/2 female connector that is connected to the
'    the Propeller.  See Keyboard_iso.spin for more pin setup codes.
'(2) %0_000_010 is a setup code that will set CapsLock ON (bit 1). See
'    Keyboard_iso for more KeyLock setup settings, such as NumLock and
'    ScrollLock.
'(3) %01_01000 is a setup code that will produce a .5sec (bit 5) delay before
'    auto repeat starts and a 15cps repeat rate (bit 3). See Keyboard_iso
'    for more setup settings.
'
'MEMORY USAGE:
'  Program: 798 Longs
'  Variables: 42 Longs
'  Stack/Free: 7,348 Longs
```

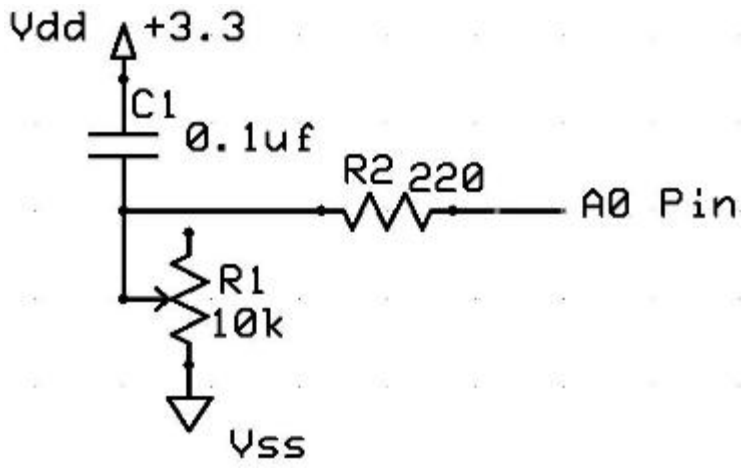


EXAMPLE 19 -- A/D CONVERSION WITH RESULTS DISPLAYED ON A VIDEO MONITOR

```
{
                                        ADC0831
                                  /CS •1├──────┤8• Vcc  +5
                                 Vin+ •2│      │7• CLK
                                  Vin-•3│      │6• DO
                                  GND •4├──────┤5• Vref

    +5V                                   ADC0831
     │      Pin A0 --------→•1├──────┤8• -------- +5V
  10k ≷<------------------→•2│      │7• -------- Pin A2
     │                    │---•3│      │6• ---/\/--- Pin A1 ←--  The series resistor is 10k.
    Vss            Vss     │---•4├──────┤5• -------- Vref +5 V   The purpose of this resistor
                                                                  is to limit current.


'
'
          WARNING: DO NOT USE THE +5 VOLTS AS THE PROPELLER'S Vdd.
                   THE PROPELLER'S Vdd MUST NEVER EXCEED 3.6 VOLTS.

          NOTE: The ADC0831 is rated at 4.5 to 6.3 volts.  I was
                able to get the ADC0831 to function using the
                Propeller's 3.3 volts power supply.  How well
                did it function? Don't know, but quick-and-dirty
                testing looked ok.
'
                RESISTOR SYMBOL IS NOT DISPLAYING CORRECTLY WHEN POSTED.
'
}
'                                EXAMPLE 19
'
'         A/D CONVERSION WITH RESULTS DISPLAYED ON A VIDEO
MONITOR
'*********************************************************************************
'IMPORTANT: This example may require an understanding of example 09
```

```
'*****************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'       ADC0831 A/D CONVERTER
'          1. This is a serial 8-bit converter.
'          2. Zero to 5 volt range on a single 5 volt power supply.
'          3. Conversion time: 32uS.
'          4. Supply voltage 4.5 to 6.3 volts.
'          5. With an overshoot, the digital output is about 5 volts.
'             Since the Propeller is a 3.3 volt device, it is prudent
'             to add a 10k resistor between Pin 6 of the ADC0831
'             and Pin A1 of the Propeller.  This limits the current
'             at Pin A1 of the Propeller.
'          6. References: http://www2.ics.hawaii.edu/~chin/331/lab08c.pdf
'                       : www.parallax.com/dl/sw/bs2Tutorial.ppt
'                         (Slides 223-230)
'
'       BS2.SHIFTIN (DataPin, CLK_Pin, Mode, Bits)
'          1. Shifts data in from a synchronous device.  In this case,
'             the device is the serial A/D converter.
'*****************************************************************************
'DIFFICULTY LEVEL: Easy
'
'PURPOSE: The purpose of this Example is to illustrate how to interface
'         a popular A/D device to the propeller and to illustrate how to
'         display the results on a video monitor.
'         WARNING: DO NOT CONNECT THE 5V SOURCE TO Vdd OF THE PROPELLER.
'                  MAX VOLTAGE TO RUN THE PROPELLER IS 3.6V.
'
'Submitted by Dave Scanlan, May 8, 2006
'File: Example19_A2D_ADC0831_Video.spin
'*****************************************************************************
'CORRECT OUTPUT:   The video monitor will dispaly the following:
'
'                           EIGHT-BIT DECIMAL VALUE: XXX
'                           ANALOG VOLTAGE INPUTTED: XXX cv
'
'
'*****************************************************************************
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000
  OUT = %1
  IN = %0
  HIGH = 1
  LOW = 0
  NewLine = 13
  ClearScreen = 0
'
  ADC_CS = 0           ' /CS Pin on Propeller
  ADC_Data = 1         ' Data Pin on Propeller
  ADC_CLK = 2          ' Clock Pin on Propeller
'
OBJ
   BS2:          "BS2_Functions"
   VideoDisplay: "TV_Terminal"
'
VAR
   Byte ADC_Result                              'A/D result (8 bits)
'
PUB START                                       'Initialize VideoDisplay
    VideoDisplay.start
    SetScreenWhiteOnDarkBlue
    VideoDisplay.out(ClearScreen)               'Clear screen.
  Repeat
    A2D_Conversion
    DisplayOutputOfADC
'
PRI A2D_Conversion
    DIRA[ADC_CS] := OUT
    DIRA[ADC_Data] := IN
    DIRA[ADC_CLK] := OUT
    OUTA[ADC_CS] := LOW     ' Select ADC chip
    ADC_Result := BS2.SHIFTIN(ADC_Data, ADC_CLK,BS2#MSBPOST,9)
    OUTA[ADC_CS] := HIGH    ' Deselect ADC chip when done
'
PRI DisplayOutputOfADC
    VideoDisplay.str(string("EIGHT-BIT DECIMAL VALUE: "))'Send text to monitor.
    VideoDisplay.dec(ADC_Result)  'Sends a decimal value to a video monitor.
```

```
    VideoDisplay.out(NewLine)
    VideoDisplay.str(string("ANALOG VOLTAGE INPUTTED: "))'Send text to monitor.
    VideoDisplay.dec(ADC_Result * 500/255)  ' Hundredth of a volt
    VideoDisplay.str(string(" cv"))         ' cv (centivolt)
    VideoDisplay.out(NewLine)               ' Send a NewLine to monitor.
    WaitCnt(80_000_000 + Cnt)               ' One second delay
    VideoDisplay.out(NewLine)
    VideoDisplay.out(NewLine)
'
'
PRI SetScreenWhiteOnDarkBlue                ' Sets the foreground color to
  VideoDisplay.out(3)                       '  white and the  background color
  VideoDisplay.out(5)                       '  to dark blue on the monitor.
'*****************************************************************************
'ADDITIONAL INFORMATION
'  WARNING: DO NOT CONNECT THE 5V POWER SOURCE TO Vdd OF THE PROPELLER.
'           MAX VOLTAGE TO RUN THE PROPELLER IS 3.6V.
'  I DID NOT DO EXTENSIVE TESTING ON THIS EXAMPLE; THEREFORE, I CANNOT
'  RECOMMEND IT FOR COMMERICAL APPLICATIONS. I BELIEVE IN MURPHY'S LAW.
'  I did not check for failed initialization of VideoDisplay object.
'  MEMORY USAGE:
'    Program: 1,735 Longs
'    Variables: 3,519 Longs
'    Stack/Free: 2,935 Longs
```



EXAMPLE 20 -- MEASURES THE CHARGE TIME OF A CAPACITOR, USES TWO COGS, AND VIDEO OUTPUT

```
'
'                                EXAMPLE 20
'
'             THIS EXAMPLE MEASURES THE CHARGE TIME OF A CAPACITOR,
'                    USES TWO COGS, AND HAS VIDEO OUTPUT
'*********************************************************************************
'IMPORTANT: This example may require an understanding of examples 6 & 9
'*********************************************************************************
'     WHAT'S NEW:
'       RCTIME METHOD
'       - This method measures in clock pulses (12.5 ns) the time it
'         takes to charge a capacitor (0.1uf) in an R/C circuit.
'         The resistor in the circuit is a 10k pot.
'
'*********************************************************************************
'DIFFICULTY LEVEL: Easy
'*********************************************************************************
'PURPOSE:
'   -- The purpose of this example is to illustrate how to program
'      an RCTIME method and how to use it. The output from the RCTIME
'      method is converted to microseconds before being displayed
'      on the video monitor.
'Submitted by Dave Scanlan, May 13, 2006
'File: Example20_RCTimeTwoCogs __.spin
'*********************************************************************************
'CORRECT OUTPUT:  The approximate time in microseconds for the 0.1uf
'                 capacitor to charge is displayed as below:
'                        VIDEO DISPLAY UNIT
'                  _____________________________________
'                 | CAP CHARGE TIME: ~XXX us            |
'                 |                                     |
'                 |                                     |
'                 |_____________________________________|
'
'*********************************************************************************
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000
  NewLine = 13
'
VAR
  Long  Count
  Long  stack0[60]  'Sets up a stack space for a Cog(processor)
  Long  stack1[50]  'Sets up a stack space for a second Cog(processor)
'
OBJ
  VideoDisplay: "TV_Terminal"
 '
PUB Start
  VideoDisplay.Start
  SetScreenWhiteOnDarkBlue
  CogNew(RCTIME, @stack0)
  CogNew(DisplayCount, @stack1)
'
PRI RCTIME | ClkStart, ClkStop, Pin, State, Out, In, High
```

```
  High := 1
  Out  := %1
  In   := %0
  Pin  := 0
  State := 1
'
  Repeat
    DirA[Pin] := Out
    OutA[Pin] := High      ' Discharge the 0.1uf capacitor
    WaitCnt(80_000 + Cnt)  ' 1ms pause
    DirA[Pin] := In        ' Reads high or low.
    ClkStart := Cnt                    ' Counter value at Start
    Waitpne(State << Pin, |< Pin, 0)   ' Wait for low on Pin 0
    ClkStop := Cnt                     ' Counter value at Stop
    Count := (ClkStop - ClkStart)      ' 12.5ns units
'
PRI DisplayCount
  Repeat
    VideoDisplay.Str(String("CAP CHARGE TIME: ~"))
    VideoDisplay.Dec(Count * 125/10_000) 'Time in microseconds
    VideoDisplay.Str(String("us"))
    VideoDisplay.Out(NewLine)
'
PRI SetScreenWhiteOnDarkBlue
  VideoDisplay.out(3)
  VideoDisplay.out(5)
'
'ADDITIONAL INFORMATION
'    The code used in the RCTIME method is a modification
'    of Martin Hebel's code in BS2_Functins.spin.
'
'    According to my 7904 Tektronics scope, cap charge times
'    displayed on the video monitor in this example are
'    in close, but not in perfect, agreement with the scope's.
'        IT WOULD BE GREAT IF SOMEONE WITH ACCURATE
'        EQUIPMENT WOULD CHECK THESE CHARGE TIMES.
'
'    Memory Usage:
'    - Program:    1,719 Longs
'    - Variables:  3,630 Longs
'    - Stack/Free: 2,839 Longs
```

EXAMPLE 21 -- W/R TO AN EXTERNAL EEPROM(24LC32A) USING I2C AND DISPLAY RESULT ON AN LCD

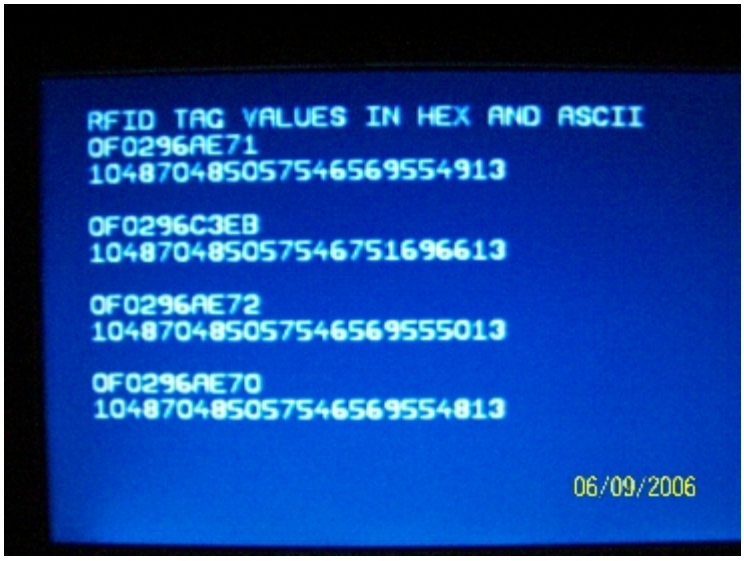
```
'
'
'
'      SERIAL EEPROM                                          SERIAL LCD (4x20)
'       ________                                             ____________________
'Vss ←□•|1   8|•→Vdd (3.3 VDC)        (Pin A2)RX--→ |                    |
'     □•|      |•→Vss (Write Enabled)         +5←---  |     LCD SCREEN     |
'     □•|      |•SCL--------•--R1--→ Pin A1  Gnd---→ |                    |
'     □•|4___5|•SDA---•----|--R2--→ Pin A2           |____________________|
'                    |    |
'       EEPROM       R3   R4
'       24LC34A      ↓    ↓                          R1: 160
'                   Vdd  Vdd  (3.3VDC)               R2: 160
'                                                    R3: 4.7K
'                                                    R4: 4.7K
'
'                              EXAMPLE 21
'
'                    W/R TO AN EEPROM(24LC32A) USING I2C
'                      AND DISPLAY RESULT ON AN LCD
'*************************************************************************
'IMPORTANT: This example may require an understanding of other examples
'*************************************************************************
'WHAT'S NEW IN THIS EXAMPLE:
'
'   I2C: The name I2C is shorthand for a standard Inter-IC
'        (integrated circuit) bus.
'         1.  Basically it is a two-wire bus using a bi-directional
'             data line (SDA: Seral DAta) and a clock line
'             (SCL: Serial CLock).
'         2.  Resources: http://www.embedded.com/story/OEG20010718S0073
'                        ww1.microchip.com/downloads/en/DeviceDoc/21713F.pdf
'         3.  This is a commonly used serial interface for IC devices.
'
'   24LC32A: This is an 8-pin, serial, 4096 byte (32Kbits) EEPROM. It uses
'            an I2C protocol.
'            CHARACTERISTICS:
'              1. 1,000,000 Erase/Write cycles
'              2. Max clock: 400 KHz
'              3. 2.5 to 5.5 volts
'
'   Control byte for 24LC32A: %1010_000_0
'           FUNCTION OF BITS:
'             1.  1010 is the device ID
'             2.  000 These are the  bus-address bits for this device.
'                   -- Eight devices (0-7) are possible on this bus.
'                   -- This example uses address 000.
'                   -- 000 was selected by grounding pins 2,3,and 4
```

```
'            3.  0 is the R/W bit. 0 = Write; 1 = Read
'                 -- This "tells" the EEPROM what the operation is
'                    going to be: Write or Read.
'
'
'*************************************************************************
'DIFFICULTY LEVEL:  Intermediate
'*************************************************************************   '
'PURPOSE: This example illustrates:
'         1. How to hookup a serial EEPROM (24LC32) to the Propeller
'         2. How to hookup a Parallax, serial LCD (20x4) to the Propeller
'         3. How to Write to and Read from a 24LC32 using I2C protocol
'         4. EVERY EFFORT WAS MADE TO SIMPLIFY THE I2C ALGORITHM.
'
'Submitted by Dave Scanlan, May 30, 2006
'File: Example21_EEPROM_I2C_LCD.spin
'*************************************************************************
'CORRECT OUTPUT:
' 1. Write:xxx (Shows the values as they are being written to the EEPROM.)
' 2. Read:xxx (Shows the values as they are being read from the EEPROM.)
' 3. W/R Failures:xxx (Shows the number of W/R failures.)
' 4. W/R Passes:xxxxx (Shows the number of successful W/R operations.)
' 5. ALL LOCATIONS OK (Displayed at the end of W/R's if there are
'                      no W/R failures.)
' 6. All values are displayed in real-time on the LCD.
'
'
'                                  SERIAL LCD (4x20)
'                                 |□□□□□□□□□□□□□□□□□□□□□□□□□□□□|
'                (Pin A2) RX--→  |Write:xxx  Read:xxx      |
'                        +5←---  |W/R Failures:xxx         |
'                    Ground---→  |W/R Passes:xxxxx         |
'                                |ALL LOCATIONS OK         |
                                 □□□□□□□□□□□□□□□□□□□□□□□□□□□□'
'*************************************************************************
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000
'
   SDA  = 0                    'EEPROM Data pin
   SCL  = 1                    'EEPROM Clock pin
   IN   = %0                   'Pin direction
   OUT  = %1                   'Pin direction
   LOW  = 0                    'Pin state
   HIGH = 1                    'Pin state
   Ack  = 0                    'Valid data transfer
   Nak  = 1                    'Invalid data transfer
   LCD_Pin   = 2               'The LCD is connected to pin A2.
   LCD_Baud  = 19_200          'Baud
   LCD_Lines = 4               'The number of lines on this LCD
'
VAR
  Byte EEPROM_ID               'Device ID: 1010 MSBits
  Word EEPROM_Address          'Holds EEPROM addresses used in
'                               Write/Read operations.
  Byte Data                    'Holds Data to be stored in the EEPROM
  Byte AddrOrData              'Working byte for TX and RX routines
  Byte Ack_I2C                 'Ack bit from device
  Byte Index                   'Used to index LOOKUP
  Byte ValueStored             'Holds the value to be written to EEPROM
  Byte ValueRead               'Holds the value read from the EEPROM.
  Word WriteReadFailures       'Holds the number of W/R failures
  Word WR_Passes               'Holds the number of successful W/R passes.
  Byte HiByte                  'Holds the MSB of the EEPROM address
  Byte LoByte                  'Holds the LSB of the EEPROM address
'
OBJ
  LCD : "Debug_Lcd"            'Code must be able to access Debug_Lcd
  BS2 : "BS2_Functions"        'Code must be able to access BS2_Functions
'
'
'
'START
'*************************************************************************
PUB Start
 Initialization                          'Calls Initialization Method
  Repeat EEPROM_Address FROM 0 to 4095   'Used to generate 4096 EEPROM
'                                         addresses for Writing to and
```

```
'                                          Reading from the EEPROM.
'
    Repeat Index FROM 1 TO 4              'Tests each EEPROM location
'                                          using four values.
      ValueStored := LOOKUP(Index: $FF, $AA, $55, $00) 'Four test values
      Data := ValueStored                 'Data is used in Write_Byte
      Write_Byte                          'Calls Write_Byte Method.
      BS2.PAUSE(10)                       'PROGRAM "MUST" PAUSE
      Read_Byte                           'Calls Read_Byte Method
      ValueRead := Data                   'Data value comes from Read_Byte
      DisplayResultsOnLCD                 'Calls DisplayResultsOnLCD
  If WriteReadFailures == 0               'If all locations test OK,
    LCD.GotoXY(0,3)                       ' ALL LOCATIONS OK is
    LCD.STR(STRING("ALL LOCATIONS OK "))  ' displayed.
'
'INITIALIZATON
'*****************************************************************
PRI Initialization
  EEPROM_ID := %1010_000_0           'Device ID: 1010 (MSBits)
  LCD.Start(LCD_Pin, LCD_Baud, LCD_Lines) 'Initialize LCD object
  BS2.Start(31,30)                    'Initialize BS2 object
  LCD.Cls                             'Clear Screen on LCD
  LCD.Backlight(1)                    'Turn on LCD backlight
  LCD.GotoXY(0,0)                     'Go to Home position on LCD
  LCD.STR(STRING("Write:"))           'For value written to EEPROM
  LCD.GotoXY(10,0)
  LCD.STR(STRING("Read:"))            'For value read from EEPROM
  LCD.GotoXY(0,1)
  LCD.STR(STRING("W/R Failures:"))    'W/R failures
  LCD.GotoXY(0,2)
  LCD.STR(STRING("W/R Passes:"))      'W/R successes
'
'DISPLAY Results on LCD
'*****************************************************************
PRI DisplayResultsOnLCD
  LCD.GotoXY(6,0)                    'Puts cursor at Col 5, Line 0
'                                     (Cols and Lines start at 0,0)
  LCD.DEC(ValueStored)               'Current value Written
  LCD.GotoXY(15,0)
  LCD.DEC(ValueRead)                 'Current value Read
  LCD.GotoXY(13,1)
  If ValueStored <> ValueRead        'Tracks W/R failures.
    WriteReadFailures := WriteReadFailures + 1
  LCD.DEC(WriteReadFailures)
  LCD.GotoXY(11,2)
  If ValueStored == ValueRead        'Tracks W/R successes
    WR_Passes := WR_Passes + 1
    LCD.DEC(WR_Passes)
'
'WRITE_Byte
'*****************************************************************
' -- See this URL for a full understanding of the Write algorithm:
'    ww1.microchip.com/downloads/en/DeviceDoc/21713F.pdf
'*****************************************************************
PRI Write_Byte
  Start_I2C                       'Send Start bit to EEPROM
  AddrOrData := EEPROM_ID & %1111_111_0  'Set "0" WRITE bit
  TX_Byte                         'Send Write bit to EEPROM
  IF (Ack_I2C == Nak)             'Wait until not busy
      Write_Byte
  HiByte :=  EEPROM_Address >> 8 'Get High address byte
  AddrOrData := HiByte
  TX_Byte                         'Send High address byte to EEPROM
  LoByte :=  EEPROM_Address       'Get Low address byte
  AddrOrData := LoByte
  TX_Byte                         'Send Low address byte to EEPROM
  AddrOrData :=  Data
  TX_Byte                         'Send Data byte to EEPROM
  Stop_I2C                        'Send Stop bit to EEPROM
'
'READ_Byte
'*****************************************************************
' -- See this URL for a full understanding of the Read algorithm:
'    ww1.microchip.com/downloads/en/DeviceDoc/21713F.pdf
'*****************************************************************
PRI Read_Byte
  Start_I2C                       'Send Start bit to EEPROM
  AddrOrData := EEPROM_ID & %1111_111_0  'Set "0" WRITE bit
```

```
  TX_Byte                          'Send Write bit to EEPROM
  IF (Ack_I2C == Nak)              'Wait until EEPROM not busy
     Write_Byte
  HiByte :=  EEPROM_Address >> 8 'Get High address byte
  AddrOrData := HiByte             'HiByte for code clarity
  TX_Byte                          'Send High address byte to EEPROM
  LoByte :=  EEPROM_Address        'Get Low address byte
  AddrOrData := LoByte             'LowByte for code clarity
  TX_Byte                          'Send LoByte address byte to EEPROM
  Start_I2C                        'End Write and Prepare to Read EEPROM
  AddrOrData := EEPROM_ID | %0000_000_1   'Set READ bit "1".
  TX_Byte                          'Send READ bit to EEPROM
  RX_Byte                          'Get Data byte from EEPROM
  Stop_I2C                         'Send Stop bit to EEPROM
  Data := AddrOrData               'Put data from EEPROM in Data
'
'RX_Byte
'*****************************************************************
'-- See PBASIC documentation for a basic understanding of SHIFTIN and
'   SHIFTOUT.
'*****************************************************************
PRI RX_Byte
  Ack_I2C := Nak                                    'Nak = High
  AddrOrData := BS2.SHIFTIN(SDA,SCL,BS2#MSBPRE,8) 'Get data from EEPROM
  BS2.SHIFTOUT (SDA, SCL,Ack_I2C, BS2#LSBFIRST,1) 'Send ACK or NAK
'
'TX_Byte
'*****************************************************************
'-- See PBASIC documentation for a basic understanding of SHIFTIN and
'   SHIFTOUT
'*****************************************************************
PRI TX_Byte
  BS2.SHIFTOUT (SDA, SCL,AddrOrData, BS2#MSBFIRST,8)
  Ack_I2C := BS2.SHIFTIN(SDA,SCL,BS2#MSBPRE,1)   '1=NO NAK; 0=ACK
'
'START
'*****************************************************************
' -- See this URL for a full understanding of the Start algorithm:
'    ww1.microchip.com/downloads/en/DeviceDoc/21713F.pdf
' -- This code should be easy to follow, because it does not
'    depend on pullup resistor conditions; and it follows the
'    start algorithm perfectly.
' -- There are more statements than some would deem necessary, but
'    these statements produce scope pulses that perfectly match the
'    algorithm. Some statements could be eliminated if processing
'    time is a major consideration. (See comments below.)
'*****************************************************************
PRI Start_I2C
  DIRA[SCL] := OUT
  OUTA[SCL] := LOW   'Could be eliminated to save processing time
  DIRA[SDA] := OUT
  OUTA[SDA] := HIGH
  OUTA[SCL] := HIGH
  OUTA[SDA] := LOW    'Sets SDA to Low to trigger START in I2C.
'                      START is triggered by transitioning SDA to a low
'                      while SCL is high.
'
  OUTA[SCL] := LOW  'Get ready for address and data
'                   Could be eliminated to save processing time
'
'PULSES GENERATED BY THE START CODE:
'    __    ___
'      |  |    SDA Pulse
'      |__|
'
'     __
'    |  |
'  ___|  |___  SCL Pulse
'
'
'
'STOP
'*****************************************************************
' -- See this URL for a full understanding of the Stop algorithm:
'    ww1.microchip.com/downloads/en/DeviceDoc/21713F.pdf
' -- This code should be easy to follow, because it does not
'    depend on pullup resistor conditions; and it follows the
'    stop algorithm perfectly.
' -- There are more statements than some would deem necessary, but
```

```
'    these statements produce scope pulses that perfectly match the
'    algorithm. Some statements could be eliminated if processing
'    time is a major consideration.  (See comments below.)
'*********************************************************************
PRI Stop_I2C
  DIRA[SCL] := OUT
  DIRA[SDA] := OUT
  OUTA[SCL] := LOW    'Could be eliminated to save processing time
  OUTA[SDA] := LOW
  OUTA[SCL] := HIGH
  OUTA[SDA] := HIGH   'Sets SDA High to trigger a STOP in I2C.
'                      STOP is triggered by transitioning SDA to a high
'                      while SCL is high.
  OUTA[SCL] := LOW    'Get ready for address and data
'                      Could be eliminated to save processing time
'
'PULSES GENERATED BY THE STOP CODE:
'    __    ___
'      |  |    SDA Pulse
'      |__|
'
'        __
'       |  |
'    ___|  |___  SCL Pulse
'
'
'
'ADDITIONAL INFORMATION:
'  -- This program is an adaptation of a PBASIC program written by Jon
'     Williams.
'  -- This code "should" work with other serial, 8-pin EEPROMs such as
'     the 24LC64, 24LC128, 24LC256, and 24LC512. These device have
'     NOT been tested with this code.  Minor modifications may be
'     necessary.
' --  The LCD (4x20) is from Parallax: Stock#: 27979
' --  The EEPROM (24LC32A) is from Parallax: Stock#: 604-00020
' --  Over 300,000 W/R operations were used to test this code.
'     No failures were detected.  Only one EEPROM was used.
' --  The code for Start_I2C and Stop_I2C was carefully checked
'     with a 500 MHz Tektronix scope to be sure that the timing is correct.
' --  In my opinion, this program has too many global variables; but in order
'     to make the code easy to understand, I did not use arguments and
'     parameters to communicate between methods.
' --  CAUTION: DO NOT ALLOW THE +5 VOLTS, USED WITH THE LCD, TO POWER
' --  THE PROPELLER.
' --  This code is intended for educational purposes only.
' --  This code is intended to be used with a single EEPROM with the
'     device address set to 000.
' --  I have used "W/R" instead of the standard "R/W" because this is the
'     order of operations in this example.
' --  R1 and R2 are not necessary unless you have more than one device
'     accessing the I2C bus.  These two resistors will compensate for a
'     possible bus short when more than one device is accessing the bus.
'
```



EXAMPLE 22 -- RFID READER WITH VIDEO OUTPUT

```
''
''            Parallax RFID Reader
''
''     ____________________________________
''    |                                    | R1: 1K Ohms
''    |                                    | (Limits current to P1)
''    |          _______________           |
''    |         |               |          |
''    |         |               |          |
''    |         |               |    GND   |• ------> Ground
''    |         |               |    SOUT  |• --R1--> P1
''    |         | _____________ |    ENABLE|• ------> P0
''    |                                    |    Vcc   |• ------>+5V DC
''    |                                    |
''    |____________________________________|
''
''                              EXAMPLE 22
''
''                    RFID READER WITH VIDEO OUTPUT
''*****************************************************************
''IMPORTANT: Do not connect RFID +5 volt supply to the Propeller.
''*****************************************************************
''WHAT'S NEW IN THIS EXAMPLE:
''     RFID READER
''         -- This device reads Radio Frequency ID tags.
''     SERIN_CHAR
''         -- SERIN_CHAR is a method used to input serial data from
''            RFID tags.
''*****************************************************************
''DIFFICULTY LEVEL: Easy
''*****************************************************************
''PURPOSE:
''    1. Illustrates how to connect an RFID reader to the Propeller.
''    2. Illustrates how to read and display RFID tag data using
''       the Parallax RFID reader.
''Submitted by Charlie Johnson, June 10, 2006, [(C) C&C, Inc.]
''Modified, elaborated and posted by Dave Scanlan
''File: Example22_RFID_ReaderWithVideoOutput.spin
''*****************************************************************
```

```
''CORRECT OUTPUT:
''
''               Video Display
''    ____________________________________
''   | RFID TAG VALUES IN HEX AND ASCII    |
''   | XXXXXXXXXX                         |
''   | XXXXXXXXXXXXXXXXXXXXXXX            |
''   |                                    |
''   | XXXXXXXXXX                         |
''   | XXXXXXXXXXXXXXXXXXXXXXX            |
''   |                                    |
''   |                                    |
''   |                                    |
''   |                                    |
''   |                                    |
''   |____________________________________|
''
''
''*****************************************************************
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000
'
  RX_Pin = 1
  Enable_Pin = 0
  Out = %1
  Low = 0
  High = 1
  CR = 13
'
VAR
  Byte Index
  Byte  Rx_Data[12]            'Holds RFID tag's 12-byte record
'
OBJ
  VideoDisplay: "TV_Terminal"  'Declare VideoDisplay
  BS2: "BS2_Functions"         'Declare BS2
'
PUB Start
  DirA[Enable_Pin] := Out
  VideoDisplay.Start(12)       'Instantiates VideoDisplay
  SetScreenWhiteOnDarkBlue
  BS2.Start(31, 30)            'Instantiates BS2
  VideoDisplay.Str(String("RFID TAG VALUES IN HEX AND ASCII"))
  VideoDisplay.Out(Cr)
'
  Repeat
    OutA[Enable_Pin] := Low    'Enables RFID
    Index := 0
    Repeat 12                  'Gets 12 bytes from RFID reader
      Rx_Data[index++] := BS2.SERIN_CHAR(RX_Pin, 2400, BS2#NInv, 8)
    VideoDisplay.Str(@RX_data) 'Displays 10 RFID hex digits.
    Index := 0
    Repeat 12
      VideoDisplay.dec(RX_Data[Index++]) 'Displays full RFID in ASCII
    VideoDisplay.Out(Cr)
    VideoDisplay.Out(Cr)
    OutA[Enable_Pin] := High    'Disables RFID
    BS2.PAUSE(1_000)            'Pause for 1 second
'
PRI SetScreenWhiteOnDarkBlue
    VideoDisplay.Out(3)
    VideoDisplay.Out(5)
''
'' ADDITIONAL INFORMATION
'' -- Be careful to not connect the RFID +5 volt supply to the Propeller
'' -- Thanks for the contributions of Marty Hebel to BS2_Functions.
'' -- Dave Scanlan does not support implanting RFID chips into humans.
'' -- This example is intended for educational use only.
```

Tracy Allen

Light Meter -- Make a light meter from an LED

Coded by Tracy Allen
www.emesystems.com

The attached two programs use an green LED to make a light meter. The LED is hooked up on the prototyping area of the Propeller demo board, with the anode (long wire) to pin a1 and the cathode (short wire) through a 330 ohm resistor to pin a0. A 220 pf capacitor is in parallel with the LED. the diagram shows what the circuit should look like. The Propeller IDE comes with its own font (the Parallax font) that includes schematic symbols that you can include in the documentation of a program. You can't see the font on your screen until you have the font installed, but the diagram is a GIF created from a screen capture (printScreen key), cropped in Paint

The LED is hooked up between two pins so that it can be both forward biased to make it light up, and then reverse biased so that it can be used as a photodiode to measure ambient light levels. In the second mode, the 220 pf capacitor is charged up to 5 volts reverse bias across the LED, and the program measures how long it takes for the capacitor to discharge down to 1.65 volts (like RCTIME on the Stamp). The Propeller runs on 3.3 volts, and its input switching threshold is 1/2 of that, close to 1.65 volts.

The first program illustrates how to make Propeller pin an input, and the rate of flashing of the LED is proportional to the ambient light level. The program also shows a couple of different ways to sample the input: 1) using a program REPEAT loop, 2) using the WAITPEQ command (wait for pin equal...), and 3) using the WAITPNE command (wait for pin not equal ...).

The second program builds on that to turn the 8 leds on the demo board into a display like a VU meter, to show the ratiometric light level. The Propeller language has some pretty amazing math operators built in. The following is the condensed version of the code. Please see the listing for the documentation dcomponent. The Propeller IDE gives you several different views of your program documentation easier.

```
CON
   ledn = 0       ' LED cathode (negative, n-type semiconductor) attached to this pin
   ledp = 1       ' LED anode (positive, p-type semiconductor) attached to this pin

'This program leaves the clock at the default ~12 mHz RC clock source.

VAR
  long ticker, ticker0
  word ratio

PUB light_meter              ' make a light meter from an LED
   blipLED
   readLED
   ticker0 := ticker         ' initial value, defines our maximum light level.
   repeat                    ' repeat the following indented statements forever.
      blipLED
      readLED
      logTicker               ' Display the bar graph


PRI blipLED                   ' briefly flash the LED
  outa[ledp] := 1             ' see lightMeter1 for comments
  dira[ledp] := 1
  outa[ledn] := 0
  dira[ledn] := 1
  waitcnt(240000 + cnt)

PRI readLED                  ' read the time it takes for photocurrent to charge capacitor
```

```
  outa[ledp] := 0            ' see lightMeter1 for comments
  dira[ledp] := 1
  outa[ledn] := 1
  dira[ledn] := 1
  waitcnt(2400 + cnt)
  ticker := 0                ' going to measure time ticks using this variable
  dira[ledn] := 0
  repeat until ina[ledn] == 0
    ++ticker                 ' every time around the loop, ticker increases by 1
                             ' so it measures the time until ina[ledn] goes low
                             ' ticker is a small number in bright light.


PRI logTicker                ' to display the result time as logarithmic (VU) bar graph
  dira := dira | $00ff0000   ' the yellow leds on demo booard are on a16-a23
  ratio := ticker0 * 8 / ticker   ' ratio = 8 when current light level = initial light level
                             ' this ratio can range from 0 in dim light
                             ' (when ticker is a large number, > ticker0 * 8)
                             ' to a large number in bright light (ticker =1)
  outa := (|< >| ratio -1) << 16      ' makes the output VU bar graph on a16--a23
```

Beau Schwabe

LED bitmap -- Fun example uses the LED's on the demo board to display a bitmap

Coded by Beau Schwabe (Parallax)

Follow his directions carefully. Great demo.

```
''*****************************************************************************
''** Fun example uses the LED's on the demo board to display a bitmap           **
''** pattern stored in a data table                                             **
''*****************************************************************************
'' Coded by Beau Schwabe (Parallax).                                           **
''*****************************************************************************
''
''                    Directions:
''                    Load the program. LED's will appear to ALL be on.
''                    Rapidly move the demo board back-n-forth.
''                    (unplug USB2SER connector first-grin
''
''
'' Other options to this program would be to use an accelerometer
'' to determine which direction/speed you are moving, and display
'' an actual message forward or backward accordingly.
  _xinfreq      = 5_000_000           'Set crystal frequency to 5 MHz
  _clkmode      = xtal1 + pll16x      'wind it up to 80 MHz via a X16 PLL

CON
  Rate = 50_000                       'Set motion rate.

VAR

PUB Message | scan
  dira[16..23] := %11111111           'Make I/O's 16 to 23 OUTPUTs

  repeat                              'Enter Endless Loop
    repeat scan from 0 to 26          'Create an offset index from 0 to 26
     outa[16..23] := Propeller[scan] 'Lookup byte value in data table at offset 'scan'
     waitcnt(Rate + cnt)              'pause so we can see something

DAT
Propeller     byte %00000000
              byte %00010000
              byte %00111000
              byte %01111100
              byte %11111110
              byte %01111100
              byte %00111000
              byte %00010000
              byte %00000000

              byte %00000000
              byte %11111111
              byte %10000001
              byte %10000001
              byte %10000001
              byte %11111111
              byte %00000000

              byte %00000000
              byte %10000000
              byte %01000000
              byte %00110000
              byte %00011000
              byte %00000111
              byte %00011000
              byte %00110000
              byte %01000000
              byte %10000000
            byte %00000000
```

RF Transmitter Eliminator

Coded by Beau Schwabe (Parallax)

Here is something fun to play with.... My daughter has a small 49MHz RF remote control Jeep.... I scoped the Transmitter (100K resistor from ANT and battery GND ; scope across resistor)
to reverse engineer the signal. (your mileage may vary). A small variation of the idea and you can do the same with IR remotes.

The setup is as simple as connecting an 8 inch piece of solid #22 hookup wire into whatever pin you assign for transmitting from the demo board.

49MHz_Demo

```
CON

  _CLKMODE = XTAL1 + PLL16X
  _XINFREQ = 5_000_000

     TXPin = 0

VAR

OBJ
  CTR : "CTR"

PUB CTR_Demo

  CTR.SynthFreq("A",TXPin, 49_700_000)                    'SynthFreq({Counter"A" or Counter"B"},Pin, Fre
q)

  repeat
    SyncHeader
    Forward

PUB Ping( uS , Pin)| Dly, SyncPoint
    Dly := ((clkfreq / 1_000_000)* uS)
    SyncPoint := cnt
    dira[Pin] := 1                                       'Modulate pin by making it an INPUT or an OUTPUT
    waitcnt(SyncPoint += Dly)
    dira[Pin] := 0                                       'Modulate pin by making it an INPUT or an OUTPUT

PUB Delay( uS )| Dly, SyncPoint
    Dly := ((clkfreq / 1_000_000)* uS)
    SyncPoint := cnt
    waitcnt(SyncPoint += Dly)

PUB SyncHeader
    repeat 4
      Ping(1500,TXPin)
      Delay(500)

PUB Signal(Mode)
    repeat Mode
      Ping(500,TXPin)
      Delay(500)

PUB Reverse
    Signal(16)                  '15 to 17

PUB Rev_Right
    Signal(28)                  '27 to 29

PUB Rev_Left
    Signal(34)                  '33 to 35

PUB Forward
    Signal(40)                  '39 to 41
```

```
PUB Fwd_Left
    Signal(46)                  '45 to 47

PUB Fwd_Right
    Signal(52)                  '51 to 53

PUB Right
    Signal(58)                  '57 to 59

PUB Left
    Signal(64)                  '63 to 65
```

CTR

```
PUB SynthFreq(CTR_AB, Pin, Freq) | s, d, ctr, frq

  Freq := Freq #> 0 <# 128_000_000      'limit frequency range

  if Freq < 500_000                     'if 0 to 499_999 Hz,
    ctr := constant(%00100 << 26)       '..set NCO mode
    s := 1                              '..shift = 1
  else                                  'if 500_000 to 128_000_000 Hz,
    ctr := constant(%00010 << 26)       '..set PLL mode
    d := >|((Freq - 1) / 1_000_000)     'determine PLLDIV
    s := 4 - d                          'determine shift
    ctr |= d << 23                      'set PLLDIV

  frq := fraction(Freq, CLKFREQ, s)     'Compute FRQA/FRQB value
  ctr |= Pin                            'set PINA to complete CTRA/CTRB value

  if CTR_AB == "A"
     CTRA := ctr                         'set CTRA
     FRQA := frq                         'set FRQA
     DIRA[Pin]~~                         'make pin output

  if CTR_AB == "B"
     CTRB := ctr                         'set CTRB
     FRQB := frq                         'set FRQB
     DIRB[Pin]~~                         'make pin output

PRI fraction(a, b, shift) : f

  if shift > 0                          'if shift, pre-shift a or b left
    a <<= shift                         'to maintain significant bits while
  if shift < 0                          'insuring proper result
    b <<= -shift

  repeat 32                             'perform long division of a/b
    f <<= 1
    if a => b
      a -= b
      f++
    a <<= 1
```

Jon Williams

RCTime

Coded by Jon Williams (Parallax)

This is something I've been playing with for the last couple days -- original program by Beau with very small updates by me (geez, I hope he doesn't mind...). This is very clever on Beau's part; the rctime() method is public an can be called manually if desired. If a cog is available, you can use the start() method and then rctime gets its own cog and just runs all the time, so any time you access your target variable it has a fresh value in it and you didn't have to worry about dreaded interrupts.

```
'' ********************************
'' *  RCTime                      *
'' *    (c) 2006 Parallax, Inc.   *
'' ********************************
''
'' Used to read an RC circuit.  When start() method is used, the rctime()
'' method is installed in its own cog and runs continuously until the
'' stop() method is called.
''
'' If no cog is available the rctime() method may be called as needed.


VAR

  long  cogon, cog

  long  rcstack[16]
  long  rctemp
  long  mode


OBJ

  time : "Timing"


PUB start(pin, state, rcvalueaddress) : okay

'' Start background RCTIME driver - starts a cog
'' -- returns false if no cog available

  stop
  okay := cogon := (cog := cognew(rctime(pin, state, rcvalueaddress), @rcstack)) > 0
  mode := 1

PUB stop

'' Stop rctime - frees a cog

  if cogon~
    cogstop(cog)


PUB rctime(pin, state, rcvalueaddress)

'' Read RC circuit and move result to rcvalueaddress
'' -- if mode 1, will run continuously in own cog
'' -- if mode 0, will terminate and may be called again manually

  repeat
    outa[pin] := state                                  ' set pin to state
    dira[pin] := 1                                      ' make pin an ouput
    time.pause1ms(1)                                    ' allow cap to (dis)charge
    dira[pin] := 0                                      ' make pin an input
    rctemp := cnt                                       ' get current counter
    waitpeq(1-state,|< pin, 0)                          ' wait for state change on pin
    rctemp := || (cnt - rctemp)                         ' calculate duration
    rctemp := rctemp - 1600                             ' adjust for instructions
    rctemp := rctemp >> 4                               ' scale value (divide by 16)
```

```
    long [rcvalueaddress] := rctemp                         ' move result to target

    if mode == 0                                            ' quit if called manually
      quit
```

Here's a bit of a demo I'm playing with that uses RCTime:

```
CON

  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000

  CR       = 13
  FF       = 12
  LF       = 10


OBJ

  debug : "SimpleDebug"
  rc    : "RCTime"
  delay : "Timing"


PUB start | potVal, idx

  if rc.start(9, 1, @potVal)                                ' start bkg RCTIME
    if debug.start(57600)                                   ' start terminal
      repeat
        debug.putc(FF)                                      ' clear screen
        debug.str(@Title1)
        debug.dec(potVal)                                   ' print pot value
        debug.str(@CrLf)
        debug.str(@Title2)
        idx := potVal / 403                                 ' calculate index
        debug.dec(idx)                                      ' print it
        delay.pause1ms(250)
    else
      rc.stop                                               ' stop rc if no terminal cog


DAT

  Title1  byte  "The POT setting is: ", 0
  Title2  byte  "LED index is: ", 0
  CrLf    byte  CR, LF, 0
```

Debug_Lcd

Coded by Jon Williams (Parallax).

I've been using our LCD (as well as the SEETRON BPI-216) as an output device for a couple weeks now -- perhaps I can save you some time.

```
'' Debug_Lcd Test (4x20 LCD)
'' -- Jon Williams, Parallax
'' -- 28 MAR 2006
''
''
'' Uses the Parallax 4x20 serial LCD to display a counter in decimal, hex, and
'' binary formats.

CON

  _clkmode = xtal1 + pll16x                             ' use crystal x 16
  _xinfreq = 5_000_000

  LCD_PIN   = 0                                         ' for Parallax 4x20 serial LCD on A0
  LCD_BAUD  = 19_200
  LCD_LINES = 4


OBJ

  lcd : "debug_lcd"


PUB main | idx

  if lcd.start(LCD_PIN, LCD_BAUD, LCD_LINES)            ' start lcd
    lcd.cursor(0)                                       ' cursor off
    lcd.backLight(true)                                 ' backlight on (if available)
    lcd.custom(0, @Bullet)                              ' create custom character 0
    lcd.cls                                             ' clear the lcd
    lcd.str(string("LCD DEBUG", 13))
    lcd.putc(0)                                         ' display custom bullet character
    lcd.str(string(" Dec", 13))
    lcd.putc(0)
    lcd.str(string(" Hex", 13))
    lcd.putc(0)
    lcd.str(string(" Bin"))

    repeat
      repeat idx from 0 to 255                          ' count up
        updateLcd(idx)
        waitcnt(clkfreq / 10 + cnt)                     ' pad with 1/10 sec

      repeat idx from -255 to 0                         ' count down
        updatelcd(idx)
        waitcnt(clkfreq / 10 + cnt)


PRI updateLcd(value)

  lcd.gotoxy(12, 1)
  lcd.decf(value, 8)                                    ' print right-justified decimal value
  lcd.gotoxy(11, 2)
  lcd.ihex(value, 8)                                    ' print indicated (with $) hex
  lcd.gotoxy(7, 3)
  lcd.ibin(value, 12)                                   ' print indicated (with %) binary


DAT

  Bullet      byte      $00, $04, $0E, $1F, $0E, $04, $00, $00
```