



## Experiment #16: Counting Events

This experiment demonstrates an events-based program delay.

### Building The Circuit



```

=====
'
'
' File..... Ex16 - Counter.BS2
' Purpose... Counts external events
' Author.... Parallax
' E-mail.... stamptech@parallaxinc.com
' Started...
' Updated... 01 MAY 2002
'
'   {$STAMP BS2}
'
=====
'
' -----
' Program Description
' -----
'
' Counts extenal events by wait for a low-to-high transition on the event
' input pin.
'
' -----
' Revision History
' -----
'
' -----
' I/O Definitions
' -----
EventIn          VAR          In15          ' event input pin
    
```

## Experiment #16: Counting Events

---

```
' -----  
' Constants  
' -----  
  
IsLow          CON      0  
IsHigh         CON      1  
Target         CON      1000      ' target count  
  
' -----  
' Variables  
' -----  
  
eCount         VAR      Word      ' event count  
  
' -----  
' Initialization  
' -----  
  
Init:  
  PAUSE 250      ' let DEBUG window open  
  DEBUG CLS, "Started... ", CR  
  eCount = 0     ' clear counter  
  
' -----  
' Program Code  
' -----  
  
Main:  
  GOSUB Wait_For_Count      ' wait for 1000 pulses  
  DEBUG "Count complete."  
  
  END  
  
' -----  
' Subroutines  
' -----  
  
Wait_For_Count:  
  IF (EventIn = IsLow) THEN Wait_For_Count      ' wait for input to go high  
  eCount = eCount + 1      ' increment event count  
  DEBUG Home, 10, "Count = ", DEC eCount, CR
```

## Experiment #16: Counting Events

```
IF (eCount = Target) THEN Wait_Done      ' check against target
Wait_Low:
  IF (EventIn = IsHigh) THEN Wait_Low    ' wait for input to go low
  GOTO Wait_For_Count
Wait_Done:
  RETURN
```

### Behind The Scenes

The purpose of the `Wait_For_Count` subroutine is to cause the program to wait for a specified number of events. In an industrial setting, for example, a packaging system we might need to run a conveyor belt until 100 boxes pass.

When the program is passed to `Wait_For_Count`, the input pin is monitored for a low-to-high transition. When the line goes high, the counter is incremented and the program waits for the line to go low. When this happens, the code loops back for the next high input. When the target count is reached, the subroutine returns to the main program. The time spent in the subroutine is determined by the rate of incoming events.

Note that the subroutine expects a clean input. A noisy input could cause spurious counts, leading to early termination of the subroutine. One method of dealing with a noisy input – when the time between expected events is known – is to add a `PAUSE` statement after the start of an event. The idea is to `PAUSE` when the event starts and end the `PAUSE` after the event with a bit of lead-time before the next event is expected. The code that follows works when the events are about a half-second in length and the time between events is two seconds:

```
Wait_For_Count:
  IF (P_in = IsLow) THEN Wait_For_Count  ' wait for high pulse
  pCount = pCount + 1                    ' increment count
  DEBUG Home, 10, "Count = ", DEC eCount, CR
  IF (pCount = Target) THEN Wait_Done    ' check against target
  PAUSE 1500                             ' clean-up noisy input
Wait_Low:
  IF (P_in = IsHigh) THEN Wait_Low      ' wait for pulse to go low
  GOTO Wait_For_Count
Wait_Done:
  RETURN
```