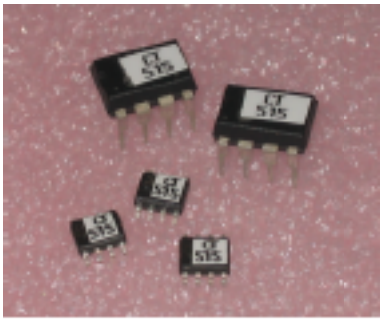
CT515 counter debouncer

EME Systems
www.emesystems.com



The CT515 is a counter/debouncer integrated circuit that adds 5 counter input channels to the OWL2pe, BASIC Stamp or other microcontroller system. The 5 channels can be used to monitor rain gages, anemometers, flow meters, traffic counters, or any such device that closes a switch or generates a pulse to signal events at a low rate (less than 100 Hertz). The CT515 can monitor and count on all 5 of its inputs simultaneously and continuously. At intervals the microcontroller will read out the number of events that have occurred on each channel during the preceding interval. Each counter is a word value, up to 65535, which is then reset to zero after the counter is read out.

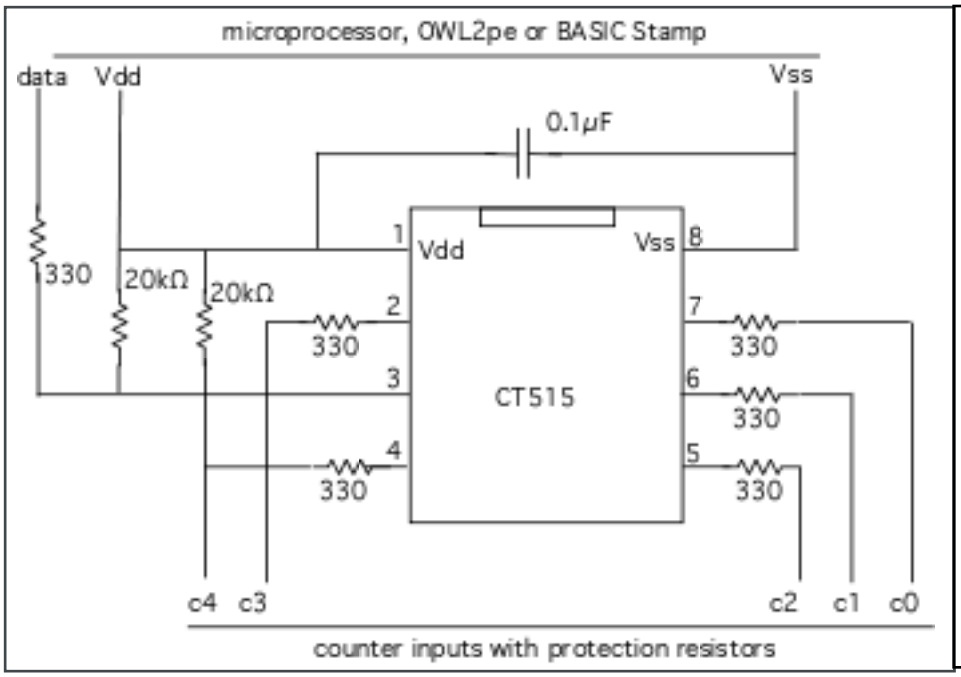


Figure 1: Typical circuit connection includes protection resistors on 5 inputs, pullup resistors on the data and c4 lines, and a power supply bypass capacitor. Internal pullup resistors are enabled on lines c0 to c3. The CT515 can operate on power from 3.0 to 5.3 volts, at 10 µA sleep, 1.2 mA max. 8 pin dip or narrow soic. The communication intereface is one single line of 5 volt level RS232.

The CT515 also acts as a debouncer. Mechanical switches tend to bounce when they close and also when they open, generating multiple high/low transitions at the input. The bouncing can last for several milliseconds. The CT515 reads the inputs repeatedly at 1 millisecond intervals and requires 5 readings in succession to have the same value before it is registered as a stable level to be counted. The CT515 by default debounces both the falling and rising edges, and advances the count on the falling edge of the debounced input. Due to the debouncing, the maximum frequency that can be detected by the CT515 is 100 Hertz, corresponding to a square wave of 5 milliseconds high and 5 milliseconds low. That is adequate for a typical anemometer, which produces a maximum frequency of around 60 Hertz.

The CT515 provides internal pullup resistors for each switch (except for counter number c4, which require an external pullup). The internal pullups are approximately 20 kohms each, so a closed switch input will draw 250 microamps. It is possible to use a transistor switch, or external logic that provides a 5 volt square wave signal instead of a switch to ground. It is advisable to include a small series resistor at each pin, shown as 330Ω in the diagram, in order to limit possible fault currents. Do not use a resistor greater than 1000Ω.

The CT515 operates at low power, sleeping much of the time at 10 microamps. (Not including pullup current) It wakes up instantly (<1 ms) when an event occurs and stays awake until it again detects a stable condition on all inputs. Even in full active operation, it only draws 1.2 milliamps.

Figure 2:
The photo shows the CT515 in the 8 pin soic package mounted on a circuit board. The components around it on the board are the bypass capacitor, the pullup resistor for the data line, and the protection resistors in an array package. The inputs lead to Phoenix terminals on the other side of the circuit board.

The CT515 is connected to the OWL2pe/BASIC Stamp via a single pin. OWL2pe top boards make this connection via pin P9, and the example programs below refer to that pin. (www.owlogic.com) But it can use any pin capable of TTL level RS232. The data pin to the microcontroller has a 20kΩ pullup resistor, which is required. The 330Ω isolation resistor is also recommended. Data is transferred at 9600 baud True (It rests at high level, 5 volts, and goes low zero volts for the start bit.)

When a program needs to read the counters, it brings the data pin low for at least 10 milliseconds, and then releases it to the high level. The CT515, after a 3ms delay to allow the Stamp to prepare, sends back an ascii string (ttl level rs232). The string consists of ascii $80 followed by a byte that contains the immediate state of the 5 inputs, and then 5 binary counter word values, each sent least significant byte first. The bytes are paced, one millisecond apart. After the CT515 sends out this string, it resets all of the counters to zero.

The initial $80 character can also be viewed as a low going pulse, 833 microseonds in duration. That is the proper value for the character $80 sent at exactly 9600 baud (one start bit + 7 low data bits = 8 * 1/9600 = 0.000833). That pulse duration can be used as a check on the CT515 baud rate, especially at extremes of temperature. The third program below illustrates how this is done on a BASIC Stamp using the PULSIN command.

		CWV		CWW		CWX		CWY		CWZ	
$80	$xx	lsb	msb	lsb	msb	lsb	msb	lsb	msb	lsb	msb

It is up to the OWL2pe or Stamp to totalize the values it receives from the CT515. For rain gages and traffic counters, it will totalize the count through time, while for other devices such as anemometers, it will divide by the time interval, to track quantities such as instantaneous, average or maximum rate or velocity.

Example OWL2pe/BASIC Stamp programs.

Example 1: The data can be captured into 5 word variables as follows. This program can also be used to demo and test the operation of the counters.

```
' acquire counts from 5 channels
' and display on debug screen
    cwv VAR Word          ' 5 words to hold the count data
    cww VAR Word
    cwx  VAR Word
    cwy   VAR Word
    cwz  VAR Word
    cwv0 VAR wv.BYTE0     ' byte alias for first byte, for SERIN
    cwd VAR Word          ' word to hold duration of header pulse
    xx  VAR Byte          ' byte to hold current state of CT515 pins
    idx  VAR byte
    goCountPin    PIN  9
     sbaud CON $54  ' 9600 baud for BS2, BS2e and BS2pe

main:
     DO
       GOSUB go_count
       DEBUG CR, DEC cwd*2, TAB, BIN6 xx
       FOR idx=0 TO 4 : DEBUG TAB, DEC cwx(idx) : NEXT
     LOOP

go_count:
     LOW goCountPin       ' get attention of CT515
     PAUSE 10
     INPUT goCountPin     ' pullup resistor pulls the pin high
     PULSIN goCountPin,0,cwd  ' header pulse duration
     SERIN goCountPin,$54,100,noData,[xx,STR cwv0\10]
noData:
     RETURN
```

The SERIN command in this syntax, using the STR modifier, will put the 5 counter results in the 5 words, cwv to cwz, so the program can subsequently display them and do calculations and take other actions. The PULSIN command measures the duration of the initial 1-0-1 pulse returned by the CT515. This is also the ascii character, $80, but it is more informative for testing purposes to measure and display the duration of the pulse. It should be close to 834 microseconds. The value returned by PULSIN is

multiplied times 2, because PULSIN returns its measurements in units of 2 microseconds (…that is on the Stamp BS2, BS2e and BS2pe, different on the other Stamps). The SERIN command first acquires the value xx, which shows the current state of all the pins on the CT515, at a point in time. DEBUG displays the value as binary, so you can see the individual key states as open=1 and closed=0. The values will be normally all high, but as switches are pressed, zeros will appear in the corresponding bit position. The counters should increment on every 0->1 transition.

Example 2: This version uses a 10 byte buffer at the top of the Stamp scratchpad memory, available in the BS2pe, BS2p, BS2px, BS2sx and BS2e. The data from the CT515 is read into this buffer by the following subroutine.

```
' acquire counts from 5 channels
' and display on debug screen
go_count:
     LOW goCountPin
     PAUSE 10
     INPUT goCountPin   ' pullup resistor pulls the pin high
     SERIN goCountPin,$54,100,noData,[WAIT ($80),xx,SPSTR 10]
     RETURN
```

Observe that this routine uses a WAIT command to detect the initial pulse, instead of the PULSIN command. SPSTR 10 puts the ten following bytes into the scratchpad memory starting at scratchpad address 0. The main loop of a data logging program can retrieve the individual counts from the scratchpad RAM .

```
     GET 0, Word variable0
     GET 2, Word variable1
     ' and so on
```

The main program loop will call the subroutine periodically. This can be done in tight synchronization with the real time clock, and this is the method we usually use with the OWL2pe data logger. After calling the subroutine at the top of the timing loop, all of the counter results are held in the scratchpad buffer, and as the main program loop deals with each sensor in turn, it retrieves the values it needs from the buffer. (see example 4)

example 3: This is the same subroutine as in example 1, but this uses the PULSIN value to calculate the baud rate. Most serial cards can sync correctly to even 5% deviation (100 to 108 microsecond bit period) . This routine should only be necessary at extremes of temperature, as the chip has a stable oscillator that is calibrated at the time of manufacture.

```
     cwd VAR word  ' duration variable
go_count:
     LOW goCountPin
     PAUSE 10
     INPUT goCountPin   ' pullup resistor pulls the pin high
     PULSIN goCountPin, 0, cwd
     cwd=cwd*2 MIN 512 MAX 1024  ' µseconds (BS2, 2e, 2pe) limits
     SERIN goCountPin,cwd/8 - 20,100,noData,[xx,SPSTR 10]
     RETURN
```

cwd	cwd/8	cwd/8-20	Stamp baud
800	100	80	10000 baud
834	104	84	9615 baud
864	108	90	9090 baud

Derivation:

Baud rate
= 1/bit period in seconds
= 1000000/(bit period in microseconds)
= 1000000/(cwd/8) as measured by PULSIN, converted to microseconds

Baudmode for Stamp 2, 2e, 2pe, from Stamp manual
= 1000000/baud - 20
= 1000000/(1000000/(cwd/8) - 20
= cwd/8 - 20 where cwd is bit period in microseconds

example 4: OWL2pe program to read the counter data into a scratchpad buffer at regular time intervals. The time is read from a DS1307 RTC as it is found on the OWL2pe data logger. The heartbeat output from the DS1307 is used to synchronize the counter reading to an exact interval of time, using the POLLWAIT command. In this way, the counts can be used for calculation of things like windspeed (count/time).

```
‘ {$STAMP BS2pe}
’ {$PBASIC 2.5}

interval CON 15  ‘ seconds between wakeup scans.

DSpwr PIN 10
DSsda PIN 8
DSsck PIN 9

ct VAR WORD
ww VAR WORD
wx VAR WORD
wy VAR WORD
wz VAR WORD
wj VAR WORD

second VAR ww.BYTE0
minute VAR ww.BYTE1
hour VAR wx.BYTE0
dow    VAR wx.BYTE1
day    VAR wy.BYTE0
month VAR wy.BYTE1
year VAR wz.BYTE0
heart VAR wz.BYTE1

AUXIO
HIGH DSpwr
  I2COUT DSsda,$D0,7,[$10]   ‘ activate heartbeat
  GOSUB realTime
  IF second=$80 THEN I2COUT DSsda,$D0,0,[0] ‘ start RTC
LOW DSpwr
MAINIO

DO
  LOW 9       ‘ attention to CT515
  PAUSE 10
  INPUT 9
  SERIN 9,$54,500,noCT515,[WAIT($80),wx,SPSTR 10]
  noCT515:
  GOSUB realTime    ‘ read and display clock time
  GOSUB showTime
  FOR wj=0 TO 4      ‘ show 5 counter values
    GET wj*2, WORD wx
    DEBUG TAB, DEC wx
  NEXT
  GOSUB waiting        ‘ sync to interval seconds
LOOP
```

```
waiting:
  AUXIO
  IF wx >4 THEN SLEEP wx-4
  IF wx=interval THEN NAP 6
  POLLMODE 2
  DO
    GOSUB secondsRemaining
    IF wx = interval THEN POLLMODE 0 : MAINIO :  RETURN
    POLLIN 15,1
    POLLWAIT 3
    POLLIN 15,0
    POLLWAIT 1
  LOOP

secondsRemaining:
' returns wx=seconds to go in sampling interval, wj=second of hour 0--3559
‘ interval should divide 3600 seconds evenly.  Should be auxio on entry & exit
  HIGH DSpwr
    I2CIN DSsda,$D1,0,[STR second\2]   ' read clock, secs & minutes
  LOW DSpwr
   wj=minute.NIB1*10+minute.NIB0*6+second.NIB1*10+second.NIB0 ‘0 to 3559
  wx = interval - (wj//interval) ' seconds remaining in interval
  RETURN

realTime:  ‘ returns RTC
  AUXIO
  HIGH DSpwr
    I2CIN DSsda,$D1,0,[STR second\8]   ' read clock, date & time & heart
  LOW DSpwr
  MAINIO
  RETURN

showRTC:
  DEBUG “20",HEX2 year,”/“,HEX2 month,”/“,HEX2 day,32
  DEBUG HEX2 hour,”:“,HEX2 minute,”:“,HEX2 second
  RETURN
```