MORE SHIFTY BITS

HEX → BCD CONVERTER REHASH

DOUBLE-DABBLE ALGORITHM

NAME COMES FROM OLD METHOD TO "READ" BINARY NUMBERS:

1101 = 13 (D in hex)

TAKE 1

DOUBLE: 2 DABBLE? +1 = 3

DOUBLE: 6 DABBLE? +0 = 6

DOUBLE: 12 DABBLE? +1 = 13

FOR BINARY TO BCD CONVERSION, ALGORITHM LOOKS LIKE

1. SHIFT BINARY NUMBER INTO BCD REGISTER (1)
2. EXAMINE EACH 4 bit group
3. IF GROUP IS > 4, ADD 3

LOOP UNTIL ALL BITS ARE SHIFTED

4 bit EXAMPLE

START	0000	0000	1101 = D hex = 13 decimal
SHIFT	0000	0001	1010
	0<4	1<4	
SHIFT	0000	0011	0100
	0<4	3<4	
SHIFT	0000	0110	1000
	0<4	6>4	
ADD 3 TO ONES	0000	1001	1000
SHIFT	0001	0011	0000 DONE
	(1)	(3)	

The algorithm then iterates n times. On each iteration, the entire scratch space is left-shifted one bit. However, before the left-shift is done, any BCD digit which is greater than 4 is incremented by 3. The increment ensures that a value of 5, incremented and left-shifted, becomes 16, thus correctly "carrying" into the next BCD digit.

The double-dabble algorithm, performed on the value 243[illegible], looks like this:

100s	Tens	Ones	Original	
0010	0100	0011	11110011	
0000	0000	0000	11110011	Initialization
0000	0000	0001	11100110	Shift
0000	0000	0011	11001100	Shift
0000	0000	0111	10011000	Shift
0000	0000	1010	10011000	Add 3 to ONES, since it was 7
0000	0001	0101	00110000	Shift
0000	0001	1000	00110000	Add 3 to ONES, since it was 5
0000	0011	0000	01100000	Shift
0000	0110	0000	11000000	Shift
0000	1001	0000	11000000	Add 3 to TENS, since it was 6
0001	0010	0001	10000000	Shift
0010	0100	0011	00000000	Shift

FIRST 3 SHIFTS NEED NO TEST

2 4 3

BCD

NOTICE - FOR 8 BIT NUMBER, 100s NEVER LARGER THAN 2

SO WE NEVER NEED TO TEST THIS DIGIT FOR >4 CONDITION (FF = 255)

MAKING IT WORK ON A PIC

PROBLEM: HOW DO YOU SHIFT ONE VARIABLE INTO ANOTHER?

THIS WON'T WORK:

```
RLF   binary
MOVF  binary, W
MOVWF BCDdigits
```

E.G.

10010110 ← 01001011

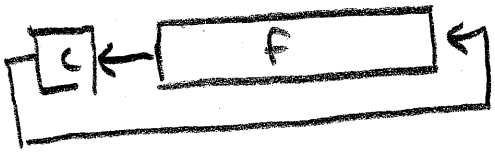
W = 10010110

BCDdigits = 10010110

WANT BCDdigits = 00000001

BUT LOOK! RLF ROTATES THROUGH CARRY!

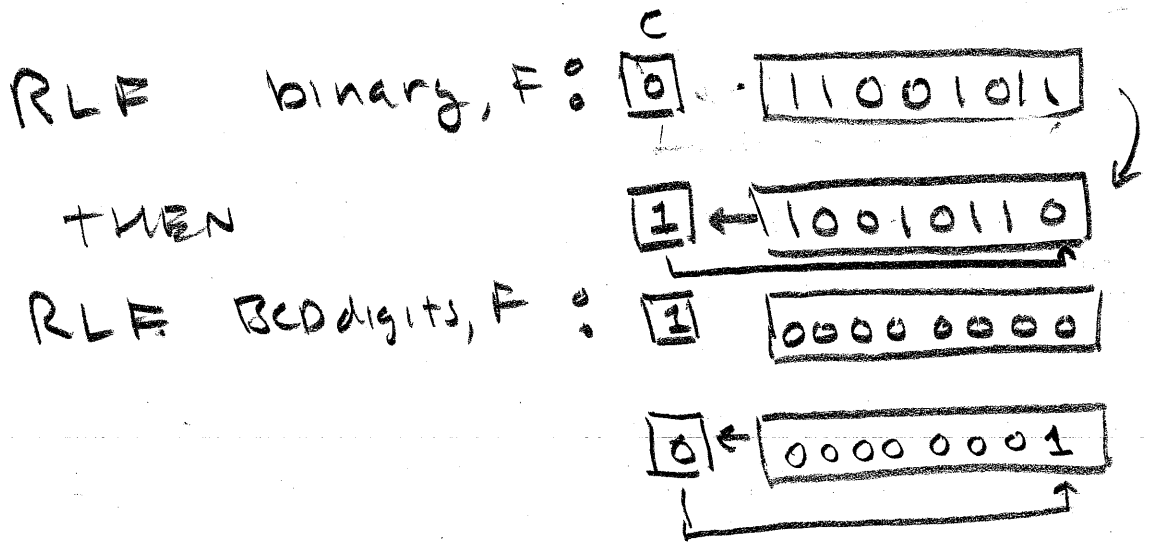
RLF f,d DOES



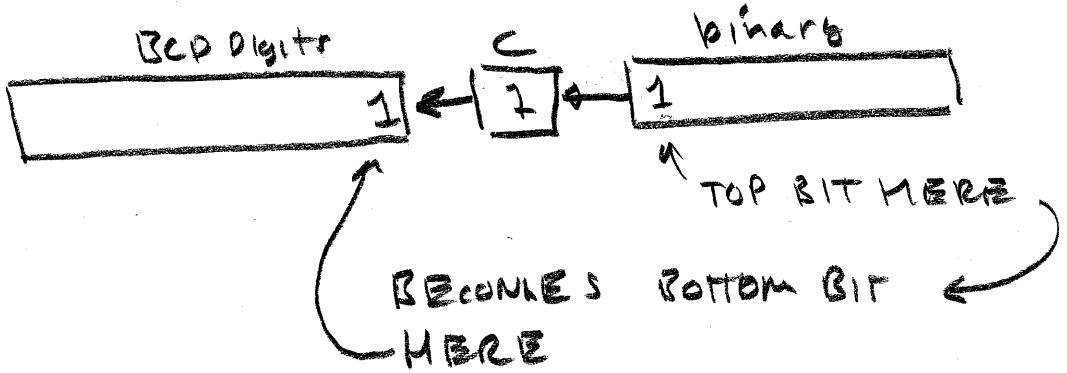
TOP BIT GOES INTO C AND C GOES INTO BOTTOM BIT

REMEMBER: CARRY "LIVES" IN STATUS

WE USE CARRY BIT AS A "PIPE"



SO TWO SUCH RLF STATEMENTS DOES THE FOLLOWING:



PROBLEM: HOW DO YOU TEST TO SEE IF A GROUP OF 4 bits HAS A NUMBER GREATER THAN 4?

(THERE ARE NO ">" "<" OPERATORS)

WE CAN ONLY TEST BITS.

NOTICE

```
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0  ← 4
0 1 0 1  ← 5   } MOST IMPORTANT (3 BITS IN)
0 1 1 0
0 1 1 1  ← 7   }
1 0 0 0  ← 8   } TOP BIT SET
1 0 0 1        }
1 0 1 0        }
```

IF WE ADD 3 TO 5, 6, 7

WE GET 8, 9, 10

TOP BIT SET

ALGORITHM:

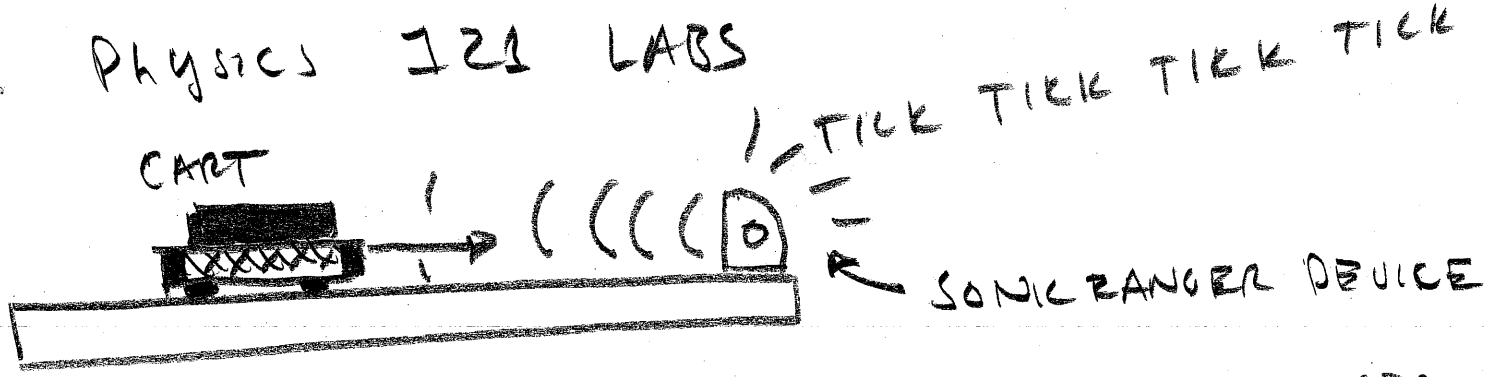
- ADD 3
- TEST TOP BIT
- SET? KEEP ADD 3
- NOT SET? DONT KEEP ADD 3

```
; bin contains the binary value to convert.
; Conversion process destroys contents
; Result is in bcdH, bcdL on return.
; Call _bin2bcd to perform conversion.
;
_bin2bcd    movlw   d'5'
            movwf   counter     ; Setting counter to 5 because first
            clrf    bcdL        ;   three shifts are done without a test.
            clrf    bcdH

            rlf     bin,F       ; These RLFs move the left (high) bits into
            rlf     bcdL,F      ;   bcdL through the carry bit (in STATUS)
            rlf     bin,F
            rlf     bcdL,F
            rlf     bin,F
            rlf     bcdL,F

_repeat     movfw   bcdL        ; Get bcdL into W
            addlw   0x33        ; Add 3 to both low and high nibble
            movwf   temp        ; Copy into temp for test
            movfw   bcdL        ; Get bcdL into W again
            btfsc   temp,3      ; Do the test on the high bit of the low nibble.
            addlw   0x03        ; If the low nibble > 4, then the high bit will
                                ;   be set when 3 was added, so we should add 3.
            btfsc   temp,7      ; Now check the high bit of the high nibble
            addlw   0x30        ;   and add 3 if the high nibble is > 4
            movwf   bcdL        ; Copy the result back into bcdL
            rlf     bin,F       ; Shift bin again.
            rlf     bcdL,F
            rlf     bcdH,F      ; Now include the shift into the high byte
                                ; We don't need to check bcdH because with
                                ;   a maximum of 255, this byte will never be
                                ;   bigger than 2
            decfsz  counter,F   ; Decrement the counter
            goto    _repeat     ; repeat until the last bit of bin is shifted.
            return
Finish
            END     ; End of program
```

TOTAL OF 8 SHIFTS

LAB PROJECT: SONIC RANGER WITH READOUT

Recall Physics 121 LABS



- SONIC RANGER SENDS OUT ULTRASONIC PULSES.
- ECHO OF PULSES IS PICKED UP
- TIME OF ECHO PROPORTIONAL TO DISTANCE

$$\frac{v_s \, \Delta t}{2} = d$$

v_s = SOUND SPEED (~340 m/s)

Δt = ECHO TIME

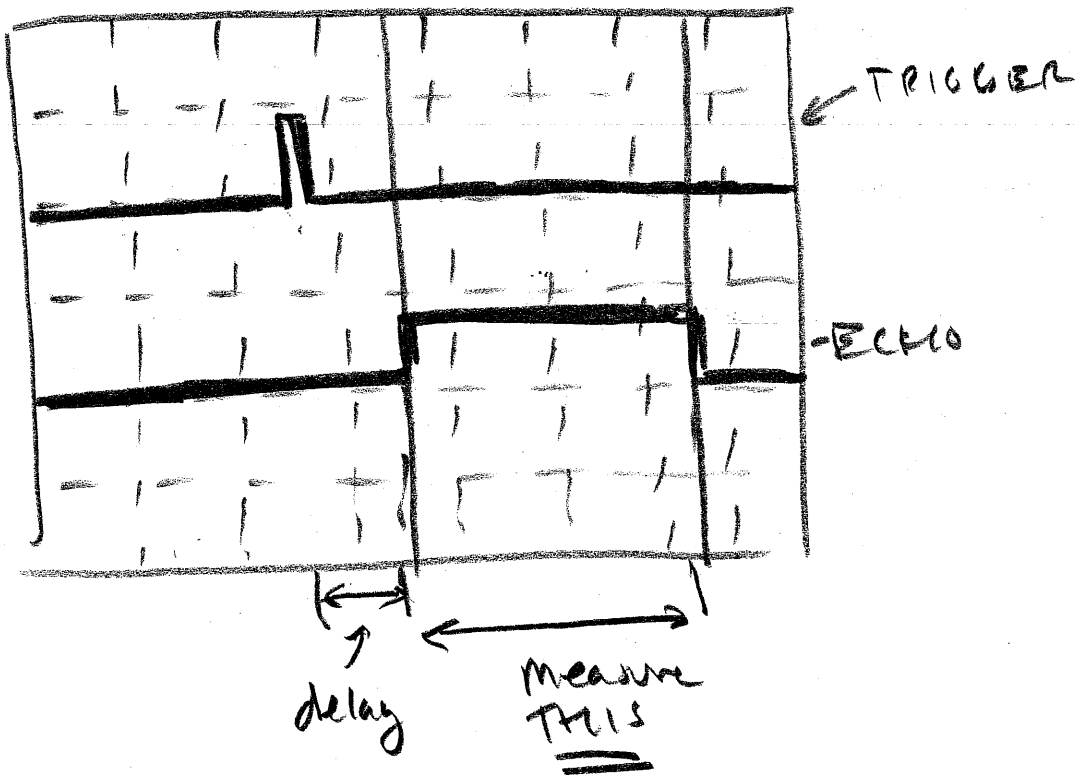
d = distance

↑ SOUND TRAVELS TO and FROM OBJECT

TO USE SONIC RANGER, NEED

- A TRIGGER PULSE, $\geq 10\,\mu s$
- A WAY TO MEASURE TIME OF ECHO PULSE

SIGNALS



HC-SR04 Ultrasonic Range Finder

Manual

Features

- Distance measurement range: 2cm - 400cm
- Accuracy: 0.3cm
- Detect angle: 15 degree
- Single +5V DC operation
- Current comsuption: 15mA

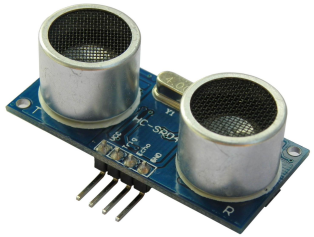


Fig. 1

How It Works

HC-SR04 consists of ultrasonic transmitter, receiver, and control circuits. When trigged it sends out a series of 40KHz ultrasonic pulses and receives echo from an object. The distance between the unit and the object is calculated by measuring the traveling time of sound and output it as the width of a TTL pulse.

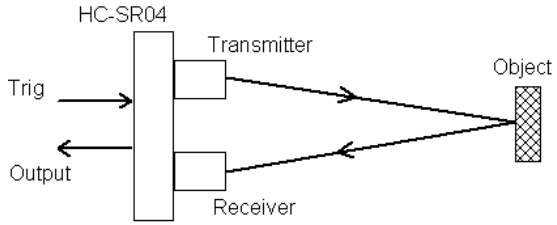


Fig. 2

How To Use It

To measure distance you need to generate a trig signal and drive it to the Trig Input pin. The trig signal leve must meet TTL level requirements (i.e. High level > 2.4V, low level < 0.8V) and its width must be greater than 10us. At the same time you need to monitor the Output pin by measuring the pulse width of output signal. The detected distance can be calculated by the formula below.

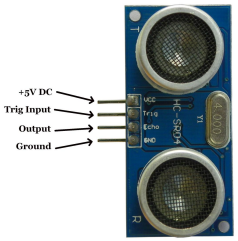


Fig.3

$$\text{Distance} = \frac{\text{Pulse Width} * \text{Sound Speed}}{2}$$

where the pulse width is in unit of second and sound speed is in unit of meter/second. Normally sound speed is 340m/s under room temperature.

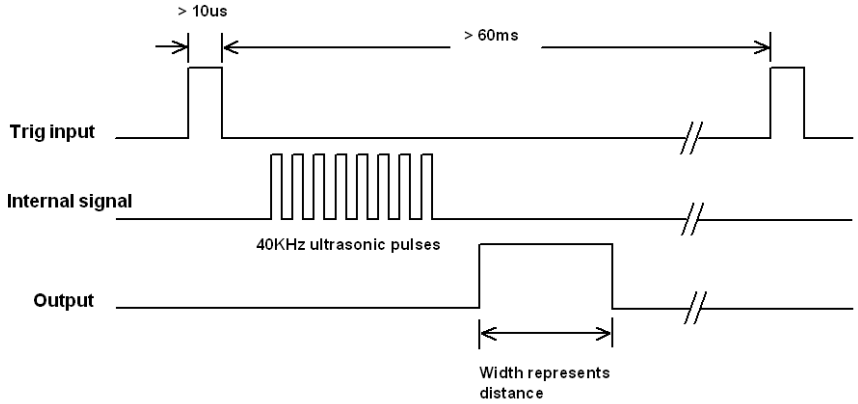


Fig. 4

Notes:
1. The width of trig signal must be greater than 10us
2. The repeat interval of trig signal should be greater than 60ms to avoid interference between connective measurements.

Specifications

Parameters	Specification
Operating Voltage	+5V DC
Operating Current	15mA
Operating Frequency	40KHz
Maximum Distance	400cm
Minimum Distance	2cm
Detect Angle	15 degree
Resolution	0.3cm
Input Trig Signal	>10us TTL pulse
Output Signal	TTL pulse with width representing distance
Weight	
Dimension	45 x 20 x 15 mm

BASIC PLAN:

- SET RA0 AS OUTPUT, RA1 AS INPUT, START TIMER
- TURN RA0 ON, THEN OFF → MAKES TRIGGER
- WAIT FOR ECHO PULSE TO GO HI ON RA1
- WHEN RA1 GOES HIGH, ZERO TIMER
- WAIT
- WHEN RA1 GOES LOW, READ TIMER
- SAVE RESULT IN REGISTER
- CONVERT TO BCD DIGITS
- DISPLAY DIGITS USING PORTB AND HP DISPLAYS
- DO IT AGAIN

TIMER ? VARIOUS OPTIONS

TIMER0

- 8 bit
- Can use internal or external clock
- 3 bit prescaler 1:1 → 1:256

USE THIS

LIKE WIDE PRESCALER

TIMER1

- 16 bit
- Can use Internal or External clock
- 2 bit prescaler 1:1 → 1:8

TIMER2

- 8 bit
- Internal clock only
- Prescaler 1:1, 1:4, 1:16 only
- Has comparator – sets interrupt flag

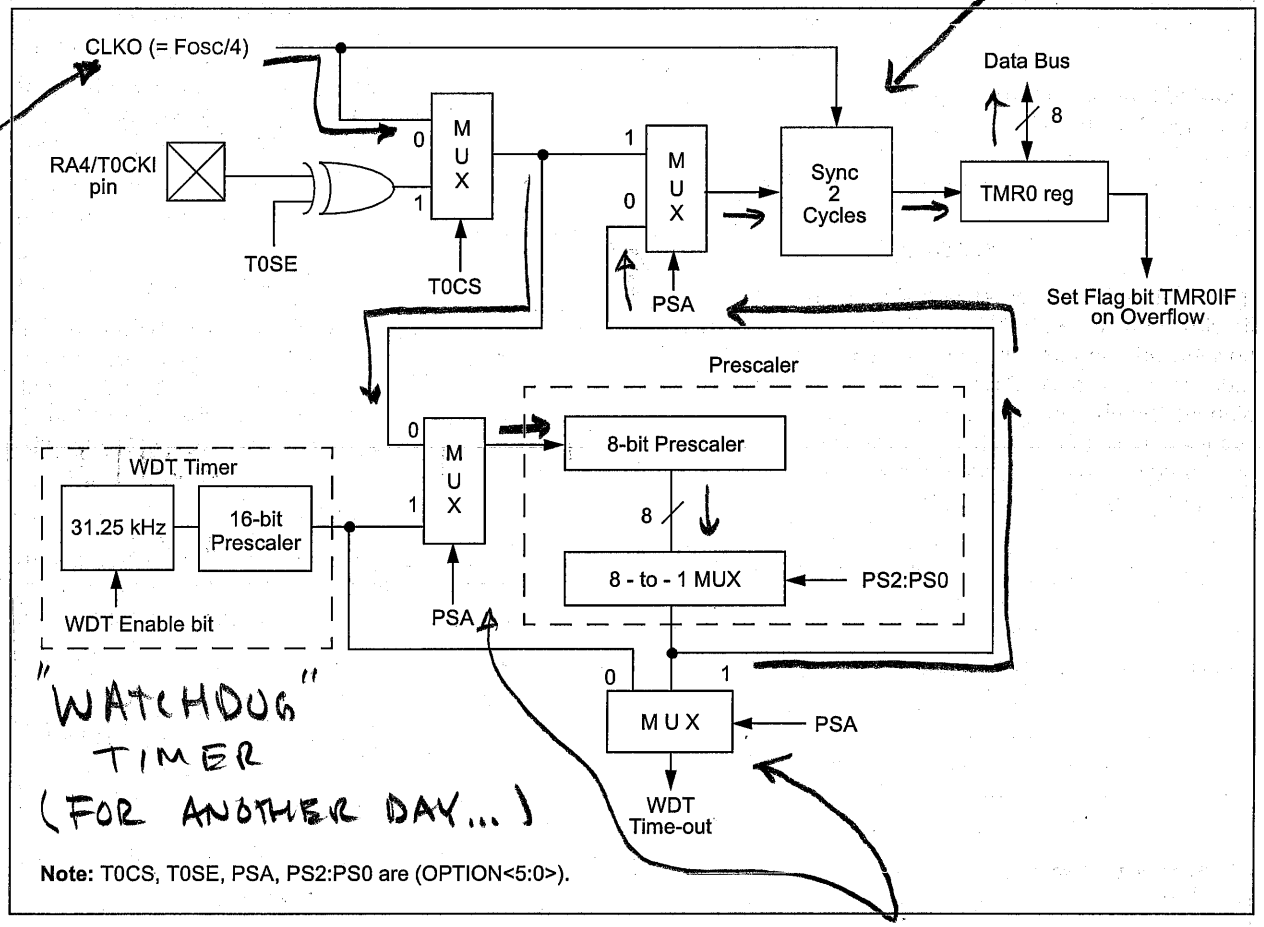
LOOK AT TIMER 0

SYNC DOESN'T MATTER FOR INTERNAL CLOCK.

NOTE CLOCK RUNS AT INSTRUCTION SPEED

Fosc/4

FIGURE 6-1: BLOCK DIAGRAM OF THE TIMER0/WDT PRESCALER



"WATCHDOG" TIMER (FOR ANOTHER DAY...)

PRESCALER CAN BE USED WITH WDT OR TIMER 0, BUT NOT BOTH

TIMER 0 CONTROL REGISTER IS CALLED OPTION_REG

REGISTER 6-1: OPTION_REG REGISTER (ADDRESS 81h, 181h) ← BANK 1

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

~~bit 7 RBPU: PORTB Pull-up Enable bit~~
~~bit 6 INTEDG: Interrupt Edge Select bit~~) NOT USED BY TIMER0

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO) ← INTERNAL CLOCK

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module ← TIMER0 USES PRESCALER

bit 2-0 **PS<2:0>**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

BANK 0

TABLE 6-1: REGISTERS ASSOCIATED WITH TIMER0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other RESETS
01h,101h	TMR0	Timer0 Module Register								xxxx xxxx	uuuu uuuu
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
81h,181h	OPTION	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

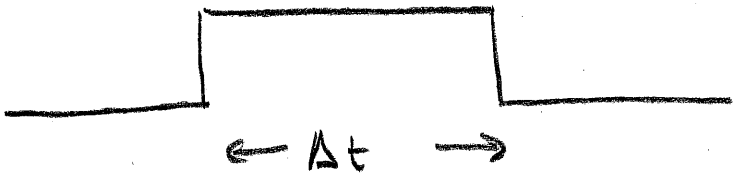
← FOR ANOTHER DAY

BANK 1 →

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by Timer0.

CHOOSING TIMER PARAMETERS

ECHO PULSE PROPORTIONAL TO DISTANCE



- DIGITAL DISPLAY = 2 DIGITS : 00 - 99
- FOR LAB BENCH DEVICE, HAVING 1 digit ≈ 1 cm SEEMS REASONABLE

WHAT IS Δt FOR 1 cm?

$$d = \frac{v_s \Delta t}{2} \rightarrow \boxed{\Delta t = \frac{2d}{v_s} = \frac{(2)(0.01\ m)}{(340\ m/s)} = 0.0588\ ms}$$

THIS CORRESPONDS TO $\frac{1}{\Delta t} = 17{,}000\ Hz$

FOR A TIMER CLOCK FREQUENCY

WANT TIMER0 CLOCK TO BE CLOSE TO 17,000 Hz

Note, WITH 1 MHz INSTRUCTION CYCLE:

TIMER0 CLOCK	PRESCALER
1 MHz	1
500 kHz	2
250 kHz	4
125 kHz	8
62.5 kHz	16
31.25 kHz	32
15.625 kHz	64 ← CLOSEST
7.8125 kHz	128

Note 15.625 kHz is $\frac{17 - 15.625}{17} \times 100 = 8.1\%$ LOW

OK FOR FIRST ATTEMPT

```
list  F=inhx8m, P=16F88, R=hex, N=0 ; File format, chip, and default radix

#include p16f88.inc ;   PIC 16f88 specific register definitions

 __config _CONFIG1, _FOSC_INTOSCCLK & _WDT_OFF & _LVP_OFF & _PWRTE_OFF &
_BODEN_ON & _LVP_OFF & _CPD_OFF & _WRT_PROTECT_OFF & _CCP1_RB0 & _CP_OFF
 __config _CONFIG2 , _IESO_OFF & _FCMEN_OFF

; Definitions ------------------------------------------------------

; You may want to add other variables here
    CBLOCK 0x20
        TimerCounts ; Saving timer counts
        bin         ; used in bin2bcd
        bcdH        ; used in bin2bcd
        bcdL        ; used in bin2bcd
        counter     ; used in bin2bcd
        temp        ; used in bin2bcd
    ENDC

; RAM preserved ----------------------------------------------------

; Constants --------------------------------------------------------

; Program Memory ---------------------------------------------------

            org     0
            goto    Init

; Interrupt Service Routine ----------------------------------------
            org     4               ; ISR beginning

; ------------------------------------------------------------------
; Microcontroller initialization
Init        org 8

; Set Internal oscillator as you choose
SetOSC

; Set up I/O on PORTA<0> Output, PORTA<1> input and PORTB<7:0> output
SetIO

; Set up Timer0, using OPTION_REG
SetTimer
```

```
; Main part of loop
MainLoop

; Make 10 microsecond pulse on PORTA<0>
Pulse

; Wait until PORTA<1> Goes HI, then clear TMR0
PulseWait_and_Clear

; Wait until PORTA<1> Goes LOW, then read TMR0 into W
EndEchoWait_and_Read

; Save TMR0 and pass to BCD converter and then display

            movwf TimerCounts    ; Save for debug comparison
            movwf bin            ; Save TMR0 to bin for Convertion

            call _bin2bcd

            call UpdateDisplay

            call Delay           ; Wait so we don't pulse too fast

            goto MainLoop


; Time Waster Routine
Delay

; Display output routine. Checks to see if there is a digit above "99"
;   If so, it outputs an overflow "OF" indication.
UpdateDisplay
        movf bcdH, w         ;Set Status 0 Flag
        btfss STATUS, Z
        goto Overflow        ;Send OF to register if we've overflowed
        movf bcdL, w
        movwf PORTB
        return

Overflow
        movlw H'0F'
        movwf PORTB
        return

; Include the binary to BCD converter here
#include bin2bcd.inc

Finish
        end                  ; end of program
```