**What registers does the timer value live in?**

-TMROH and TMROL form the 16-bit timer value

-Timer 0 High Byte acts as a buffer

**What register does the TMROIF live in?**

-PIR3

**What clock source?**

-set to processors clock of Fosc/4 =16Mhz

-to execute one instruction or one cycle is 6.25E-8 s or 62.5 nanoseconds long

-how many clock cycles in 1us => 1us/62.5ns = 16 clock cycles(instructions) are in 1us

-how many clock cycles in 65,535 us?

-65,535us x (16 clock cycles/1us)= 1048560 clock cycles

-need to slow down the timer’s clock frequency using prescaler, prescaler divides input clock frequency that the timer sees down to a slower value

**What we need?**

Timer prescaler : 1:16

- because 16Mhz/16 = 1Mhz

-takes 1us to execute one instruction or one cycle

-takes us 16us to execute 16 clock cycles

-65,535us x (16 clock cycles/16 us) = 65,535 clock cycles for 16 bits

-This is because we want it between 1us to 65,535 us

|  |  |  |
| --- | --- | --- |
| Register | Binary Init Value | Hex Init Value |
| T0CON0 | 1(0)01 0000 | 0x90 |
| T0CON1 | 0101 0100 | 0x54 |

**Program Design Part 1**

1. **Stop the timer.**

* In register T0CON0 bit 7 clear to 0
* Bitwise/bitmask operation so that T0CON0 bit 7 is always 0

1. **Split TMR0H and TMR0L int**

* Recognize that it the value within these two registers are is the (65,535-#microseconds ). This is because this is where the counter starts due to the offset or input number into the function
* To split into low and high byte, we must mask the bottom 8 bits of the number and then put that value into TMR0H. Then we must mask the top 8 bits of the number and then put that value in TMROL
* Declare and initialize a variable to store the (65,535 - #microseconds).
  + For TMROL bit mask the top 8 bits with & operator, operating it with 0x0011
  + For TMROH bit mask the bottom 8 bits with & operator, operating it with 0x1100. However you will need to shift this value to the right by 8 bits i>>8

1. **Clear TMROIF**

* Bitwise/bitmask operation so that PIR3 bit 7 is always 0
* Bitwise/bitmask operation so that T0CON0 bit 7 is always 1

**Program Design Part 2**

1. **Blink RA0 LED**

* Blink at one second
* Declare a new static variable called u16counter and set to 0
* Increment the new static variable by 1
* Write an if statement to test whether or not u16counter has reached 500
* Within if statement reset u16counter = 0
* Toggle value
  + Note to only toggle the RA0 which means that this is LATA’s least significant bit
  + Now we must bit-mask the using an AND operator with LATA & 0x01. This is so that the other LEDs are off and will remain off.
  + Next use the bitwise XOR operator to toggle the value. This means (LATA & 0X01)^0X01

**Program Design for Interesting LED Pattern**

1. **Must Create an array of u8 to store patterns**

* Recall arrays int[5]={1,0,2,4,7,8}
* Except in our case we must store LED bit patterns and then access each bit pattern through indexing
* Declare and initialize the au8Pattern array with LED bit patterns.
* Index au8Pattern[i] where i can start at 0

1. **Implementing the pattern**

* The delay or timer is built into the system and userapp will be repeatedly called. Therefore, no for loop is actually necessary.
* It is good though to implement a static counter to slow down the timer or cause a delay between the switch from current bit pattern to the next one
* However, we need to declare au8Pattern first and fill it in with values and a static variable that will increment the index i each time
* Then we need to apply our au8Pattern[i] to LATA properly with the OR operator and then reset LATA back to 0x80 again to clear it for the next pattern
* When you reach the end of the array we want to repeat the array again and therefore when index == 6, we reset the index to 0 once again so that it can repeat

Code

void TimeXus(INPUT\_PARAMETER\_)

{

/\* OPTIONAL: range check and handle edge cases \*/

/\* Disable the timer during config \*/

T0CON0 = T0CON0 & 0x7F;

/\* Preload TMR0H and TMR0L based on u16TimeXus \*/

/\* Clear TMR0IF and enable Timer 0 \*/

PIR3 = PIR3 & 0X7F;

T0CON0 = T0CON0 & 0xFF;

} /\* end TimeXus () \*/

void UserAppInitialize(void)

{

/\* Timer control register Initialization \*/

T0CON0 = 0x90;

T0CON1 = 0x44;

} /\* end UserAppInitialize() \*/

/\*!----------------------------------------------------------------------------------------------------------------------

@fn void UserAppRun(void)

@brief Application code that runs once per system loop

Requires:

-

Promises:

-

\*/

void UserAppRun(void)

{

u32 TempLATA;

for (u8 counter =0x00; counter < 64; counter++ )

{

//This is for the delay of 250 ms

u32 u32Counter = 400000;//Calculated Number used to burn 250 ms

while(u32Counter>0)

{

u32Counter= u32Counter-1; //Decrement counter to burn 250ms

}

TempLATA = ((LATA + 0x01 | 0x80) & 0xBF);

LATA = TempLATA;

}

//Blink RED LED RA0

static u16 u16counter = 0;

u16counter= u16counter +1;

if(u16counter >= 500)

{

u16counter = 0;

LATA= (LATA & 0X01)^0X01;

}

//Interesting pattern design

u8 au8Pattern[] ={0x01, 0x02, 0x04, 0x08, 0x10, 0x20};

static int index = 0;

LATA = 0x80;

LATA = LATA & auPattern[index];

index = index +1;

void TimeXus(u16 microsecondsoffset)

{

/\* OPTIONAL: range check and handle edge cases \*/

/\* Disable the timer during config \*/

T0CON0 = T0CON0 & 0x7F; // turns timer off

u16 timervalue = 0xFFFF-microsecondsoffset; //value for which the timer starts counting

/\* Preload TMR0H and TMR0L based on u16TimeXus \*/

TMR0L = (u8)(timervalue & 0x00FF);

TMR0H = (u8)((timervalue >> 8) & 0x00FF);

/\* Clear TMR0IF and enable Timer 0 \*/

PIR3 = PIR3 & 0X7F; // clears the TMR0IF flag

T0CON0 = T0CON0 | 0x80; // starts timer

} /\* end TimeXus () \*/

void UserAppInitialize(void)

{

/\* Timer control register Initialization \*/

T0CON0 = 0x90; //Sets the appropriate settings for timer0

T0CON1 = 0x54;

LATA = 0X80;

} /\* end UserAppInitialize() \*/

/\*!----------------------------------------------------------------------------------------------------------------------

@fn void UserAppRun(void)

@brief Application code that runs once per system loop

Requires:

-

Promises:

-

\*/

void UserAppRun(void)

{

/\*Bit counter\*/

//u32 TempLATA;

//for (u8 counter =0x00; counter < 64; counter++ )

//{

//This is for the delay of 250 ms

// u32 u32Counter = 400000;

//while(u32Counter>0)

//{

//u32Counter= u32Counter-1;

//}

//TempLATA = ((LATA | 0x80) +0x01);

//LATA = TempLATA;

// }

static u16 u16counter = 0;

u16counter= u16counter+1;

/\*Blink RED LED RA0\*/

//if(u16counter >= 500) //causes the operation of toggle to delay

//{

// u16counter = 0;

//LATA= (LATA & 0X01)^0X01;

//}

/\*Interesting pattern design \*/

static int index = 0;

u8 au8Pattern[] ={0x01,0x08, 0x04, 0x02, 0x10, 0x20};

if (u16counter >= 200) //causes the operation of implementing next pattern to delay

{

u16counter =0;

LATA = 0x80; //resets the LED to be ready for next pattern

LATA = LATA | au8Pattern[index]; //access the pattern in array and implements it

index = index +1;

}

if(index == 6) //repeats the pattern of the bit patterns

{

index=0;

}

} /\* end UserAppRun \*/