Universidad de Puerto Rico
Recinto de Mayaguez
Departamento de Ingenieria Electrica y Computadoras.
ICOM5217 - Interconexión de Microprocesadores

Experiment #3 - Report

Timers and Applications

Juan Lebrón Betancourt
Anthony Llanos Velázquez
Profesor: Manuel Jimenez
Instructor: Luis Malavé
7 de octubre de 2013

Exercise:

Timer Interrupts

•

Planning and Code

Timer Interrupts

A buzzer was connected as in the instructions of the experiment. After this, we search for the way the timers are activated in the Tiva Microprocessor:

```
    // Set the clocking to run directly from the crystal.
    ROM_SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC
                       | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);

    // Enable the timer peripherals.
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);

    // Enable processor interrupts.
    ROM_IntMasterEnable();

    // Configure the two 32-bit periodic timers.
    ROM_TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
    ROM_TimerLoadSet(TIMER0_BASE, TIMER_A, ROM_SysCtlClockGet()/2000);

    //
    // Setup the interrupts for the timer timeouts.
    ROM_IntEnable(INT_TIMER0A);
    ROM_TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    // Enable the timers.
    ROM_TimerEnable(TIMER0_BASE, TIMER_A);
```

Once the timers are activated, in the method TimerIntEnable() we are able to set the timer to trigger an interrupt. The frequency at which the timer is set is computed by using a method SysCtlClockGet which returns a frequency value for the processor such that it is 0.5s or 2 Hz. By dividing over 2000, we obtain the desired frequency of 1000 Hz.

The interrupt vector table has a method which is the handler for the interrupts of the timer. This handler merely toggles the output pin in which the buzzer is connected.

```
void Timer0IntHandler(void){
    // Clear the timer interrupt.
    ROM_TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    // Toggle the flag for the first timer.
    HWREGBITW(&g_ui32Flags, 0) ^= 1;
    // Use the flags to Toggle the PIN for this timer.
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, g_ui32Flags << 1);
}
```

After this an additional interrupt handler was configured for the switches that were used to increase and decrease the flags by a factor of 10.

Handler for Frequency Modification.

```
void inc(){

    // Wait. Software debouncing.
    SysCtlDelay(20000);

    // Get contents of Raw Interrupt Status to determine
    // which pin caused the interrupt.
    int ris = HWREG(0x40007414) ;
    // Right pin?
    if(ris & 0x00000001){
        // Multiplu by ten.
        freq = (freq * 10);
        freq = (freq > 20000) ? 2 : freq ;
        ROM_TimerLoadSet(TIMER0_BASE, TIMER_A,
                                ROM_SysCtlClockGet()/freq);
    }
    // Left pin.
    else{
        // Divide
        freq = (freq / 10);
        freq = (freq <= 1) ? 20000 : freq ;
        ROM_TimerLoadSet(TIMER0_BASE, TIMER_A, ROM_SysCtlClockGet()/freq);
    }
    SysCtlDelay(2000000);
    IntFinish();
}
```

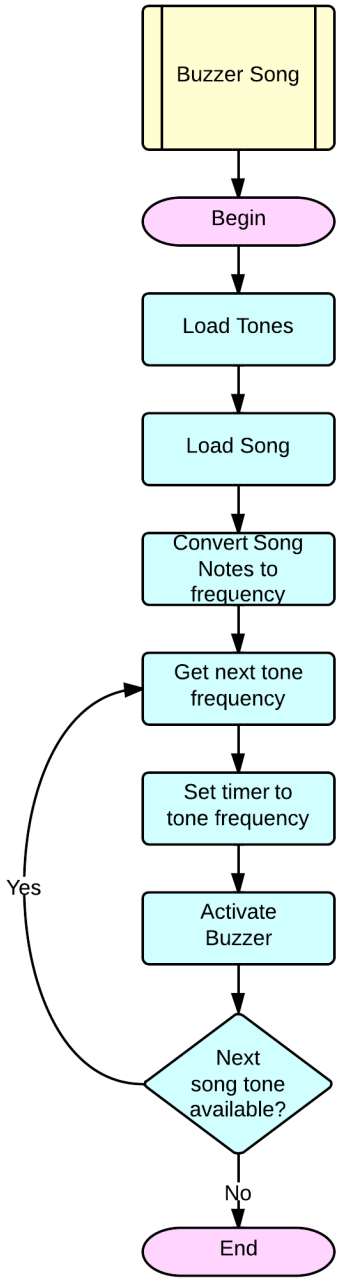
After this, a program was written such that it played a song. This song is name Fur Elise. A library was written to facilitate the configuration process of the buzzer. The implementation is included below with the notes library omitted:

```
int freq = 1000;
int note = 0;
int note2 = 0;
int note_count = 35;
int notesOctave[];
float song[1000];
char eliseSong[2000] = "E D# E D# E B D C A C E A B E G# B C E E D# E D# E B
D C A C E A B E C B A A B C D E G F E D F E D C E D C B C E A E E D# E D# E B
D C A A E A C E A E B E G# E C B A A E A B C D C E G C G F E G D G B F E D A
C E A E D C E B E E E E E E E E D# E D# E D# E D# E D# E D# E B D C A A E A C
E A E B E G# E G# B A C E A E E D# E D# E B D C A A E A C E A E B E G# E C B
A A E A Bb C E C A C F C G C E Bb C F C A F A C A C F A E F E Bb D D Bb D Bb
Bb A F A E G F G Bb F E E F G Bb D E C F Bb A C A A C A Bb G A A Bb F C A C A
C D A D# E E A C A E D D F F A G C F G F G D G B C E C G G G A G F G B G E G
C G D F G D G C E G E G C B F A A G F E G B D G F D C C G G G A G F G B G E G
C G D F G D G C E G E G C B F A A G F E G B D G F D G# B E F E D# E B E D# E
B E D# E B E D# E B E D# E D# E D# E D# E D# E B D C A A E A C E A E B E G# E
G# B A C E A E E D# E D# E B D C A A E A C E A E B E G# E C B A A E A B C D C
E G C G F E G D G B F E D A C E A E D C E B E E E E E E E E D# E D# E D# E D#
E D# E D# E B D C A A E A C E A E B E G# E G# B A C E A E E D# E D# E B D C A
A E A C E A E B E G# E C B A A A A A A A A E G Bb C# A A A A A A F A D A A A
A C# E A D F A G# D F A A A A G# D F A A A C E A A A A A D A F D D A D A D A
D A E C D A D B D A C F # A D A D A D A D A C A D A E A C A E A E A E C E A E
```

```
G# D B E G # A A C A A A A A A A A E G Bb C# A A A A A A A F A D A A A A C# E A D
F A D F A A A A D F A Bb D F Bb Bb Bb Bb Bb Bb G Eb Bb Bb Bb Bb FD Bb Eb C Bb
D F Bb Bb Bb Bb Bb D F A Bb B D F Ab B B B B D F Ab B C C E A E G# E B A A C
E A C E A C E D C B A C E A C E A C E A C E A C E D C B A C E A C E A C E A C E D C
B A C E Bb A G# G F# F E D# D C# C B Bb A G# G F# F E D# E B D C A A E A C E
A E B E G# E G# B A C E A E E D# E D# E B D C A A E A C E A E B E G# E C B A
A E A B C D C E G C G F E G D G B F E D A C E A E D C E B E E E E E E E E D#
E D# E D# E D# E D# E D# E D# E B D C A A E A C E A E B E G# E G# B A C E A E E D#
E D# E B D C A A E A C E A E B E G# E C B A A A ";
char sillySong[100] = "E D C D E E E D D D E E E E D C D E E E D D E D C ";
int counteri = 0;
int pushed = 0;

void inc(){

      int ris = HWREG(0x40007414) ;
      int one_hertz =  ROM_SysCtlClockGet()/2;
      int notesOctave2[8]={16.35,18.35,20.60,21.83,24.52,27.50,30.87,32.70};


      if(ris & 0x00000001){

            pushed = 1;

      }
      else{
            ROM_TimerLoadSet(TIMER0_BASE, TIMER_A,
                              one_hertz/(notesOctave2[note2]*32));
            note2 = (note2 + 1) % 8;
      }

      SysCtlDelay(2000000);
      IntFinish();

}

void convertStringToNotes(){
      int x = 0;
      int add = 0;
      while(add < 888){
            switch(eliseSong[add]){
            case 'C': switch(eliseSong[ add + 1]){
                              case ' ': song[x] = notes_c;
                                          add += 2;
                                          break;
                              case '#': song[x] = notes_c_sharp;
                                          add += 3;
                                          break;
                        }
            break;
            case 'D': switch(eliseSong[ add + 1]){
                              case ' ': song[x] = notes_d;
                                          add += 2;
                                          break;
                              case '#': song[x] = notes_d_sharp;
                                          add += 3;
                                          break;
                              case 'b': song[x] = notes_d_flat;
```

```
                                        add += 3;
                                        break;
                        }
                break;
                case 'E': switch(eliseSong[ add + 1]){
                                case ' ': song[x] = notes_e;
                                        add += 2;
                                        break;
                                case 'b': song[x] = notes_e_flat;
                                        add += 3;
                                        break;
                        }
                break;
                case 'F': switch(eliseSong[ add + 1]){
                                case ' ': song[x] = notes_f;
                                        add += 2;
                                        break;
                                case '#': song[x] = notes_f_sharp;
                                        add += 3;
                                        break;
                        }
                break;
                case 'G': switch(eliseSong[ add + 1]){
                                case ' ': song[x] = notes_g;
                                        add += 2;
                                        break;
                                case '#': song[x] = notes_g_sharp;
                                        add += 3;
                                        break;
                                case 'b': song[x] = notes_g_flat;
                                        add += 3;
                                        break;
                        }
                break;
                case 'A': switch(eliseSong[ add + 1]){
                                case ' ': song[x] = notes_a;
                                        add += 2;
                                        break;
                                case '#': song[x] = notes_a_sharp;
                                        add += 3;
                                        break;
                                case 'b': song[x] = notes_a_flat;
                                        add += 3;
                                        break;
                        }
                break;
                case 'B': switch(eliseSong[ add + 1]){
                                case ' ': song[x] = notes_b;
                                        add += 2;
                                        break;
                                case 'b': song[x] = notes_b_flat;
                                        add += 3;
                                        break;
                        }
                break;
                }
                x++;
}
```

```
}

void nextTone(){
      int one_hertz =  ROM_SysCtlClockGet()/2;
      ROM_TimerLoadSet(TIMER0_BASE, TIMER_A,  one_hertz/(song[counteri]*32));
      //SysCtlDelay(1000000);
}

void nextToneWithTime(int time){
      int one_hertz =  ROM_SysCtlClockGet()/2;
      ROM_TimerLoadSet(TIMER0_BASE, TIMER_A,  one_hertz/(song[counteri]*32));
      SysCtlDelay(time);
}
```



Exercise:

Random Number Guess

Codenamed: “Guessimal”

•

Planning and Code

Guessimal

A game was implemented such that it provided an interface for a user to play a game of guessing a number between 0 and 9. To implement this game, a random number had to be generated. This random number was obtained by initializing a timer from which its count value was extracted and reduced to the range 0 and 9 using the modulus function. This was done in a function that resets the entire game:

```
void gameReset(){
	tries = 3 ;
	selection = 0;
	target = TimerValueGet(TIMER0_BASE,TIMER_A) % 10;
	gameInit();
}
```

After this the logic of the game was implemented such that the user could see the numbers from 0 to 9 and select his/her preferred guess.

```
void switchPressed(){
	//Wait for other button.
	//Software Debouncing and Delay.
	SysCtlDelay(2000000);
	if(gamePressedBoth() == 0x06){
		if(gameCheckDecision()){
			gameReset();
		}
		else{
			gameTryAgain();
		}
	}
	else if(gamePressedRight()){
		gameSelectionRight();
	}
	else{
		gameSelectionLeft();
	}
	SysCtlDelay(2000000);
	IntFinish();
}

//Selection Right? Move up in circular numeration.
void gameSelectionRight(){
	selection=(selection+1)%10 ;
	gameWriteSelection(selection+48);
}
//Selection left? Move down in circular numeration.
void gameSelectionLeft(){
	selection=(selection-1);
	if(selection<0) selection= 9 ;
	gameWriteSelection(selection+48) ;
}
//Check Raw Interrupt Status for right button.
int gamePressedRight(){
	return HWREG(0x40024414) & 0x02 ;
}
```

```
//Check Raw Interrupt Status for left button.
int gamePressedLeft(){
	return HWREG(0x40024414) & 0x04 ;
}

//Check Raw Interrupt Status for both buttons.
int gamePressedBoth(){
	return HWREG(0x40024414) & 0x06 ;
}

//Writes the selection to the LCD using omitted library
//created from first, second laboratories.
void gameWriteSelection(int selection){
	lcdCursorHomeDown();
	lcdWriteString("     < .");
	lcdWriteLetter(selection);
	lcdWriteString(" >      .");
}
```

The desicion process of the game was implemented in the following method which let the user try three times before losing. Once the user fails three times to guess the number, then the game resets by displaying a welcome message and restarting the game with another number.

```
//Check if guess is correct.
int gameCheckDecision(){

	if(gameValidate()){
		lcdWriteLineUp(" You win! :D.");
		SysCtlDelay(20000000);
		return 1 ; //Correct desicion: reset.
	}
	else{
		//UI Stuff.
		lcdClearLine(LCD_LINE_DOWN);

		//Tries left.
		if(tries > 0){
			tries--;
			lcdWriteLineUp(" Try again :/.");
			SysCtlDelay(4000000);
			return 0; //Incorrect desicion but has more tries.
		}
		//No tries left.
		else{
			gameLose();
			SysCtlDelay(20000000);
			return 1; //Incorrect desicion but
				  //has no tries left: reset.
		}
	}

}
```

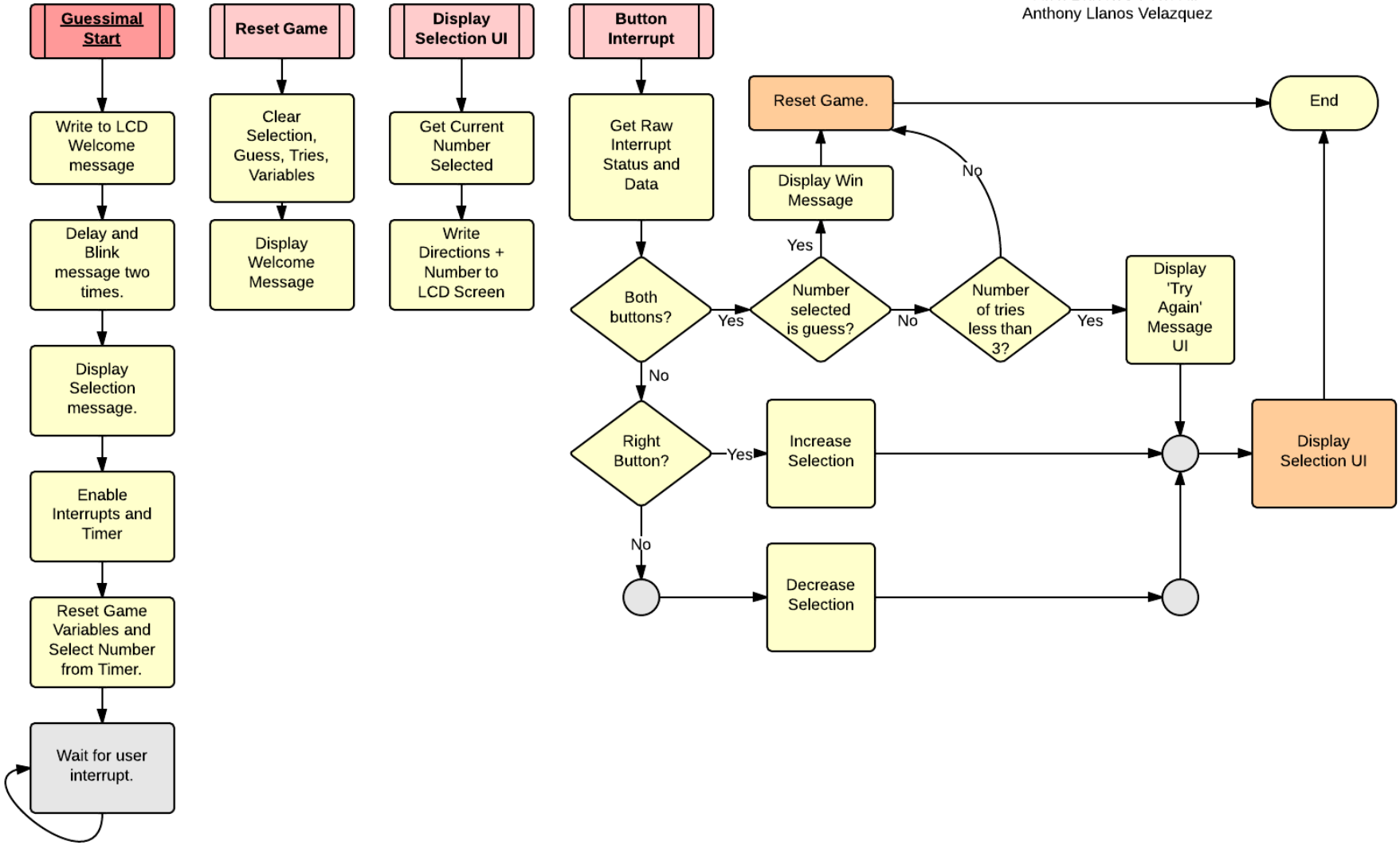
```
//Reset number back to 0.
void gameTryAgain(){
      selection = 0 ;
      lcdWriteLines("Select Number:.","     < 0 >.");
}

//Display the number.
void gameLose(){
      lcdWriteLineUp(" You lose :( .");
      lcdClearLine(LCD_LINE_DOWN);
      lcdCursorHomeDown();
      lcdWriteString("Number: .");
      lcdWriteLetter(target+48);
}

//Reset game.
void gameInit(){
      lcdClear();
      lcdCursorHome() ;
      //UI Only.
      gameBlinkingMessage("  Welcome to .","   Guessimal  .", 2);
      lcdWriteLines("Select Number:.","     < 0 >.");
}
```

The **timer** is inited with a standard frequency of 1000 Hz.

```
void initTimerModule(){
      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
      SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC
                        | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
      TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
      TimerLoadSet(TIMER0_BASE, TIMER_A,
                        ROM_SysCtlClockGet()/(TIMER_FREQUENCY*2));
      TimerEnable(TIMER0_BASE, TIMER_A);
}
```

Experiment 3 - Part III - Software Plan

Juan Lebron Betancourt
Anthony Llanos Velazquez



Exercise:

Homework

•

Code

Tacometer

For the homework, a tacometer was implemented using the photosensors RPR-220 used in the previous laboratory. The idea was to give an output of the RPM of the wheel. For this we had to used timers such that we could keep track of the time. In the implementation, a timer was set such that it executed every millisecond. In every execution, the timer incremented a variable named milliseconds. Once the milliseconds were greater than 1000, the variable was reset to 0 and a variable seconds was incremented. On every full turn which was the equivalent of 7 interrupts, the elapsed time was taken from these variables and the rpm computed according to the following formula:

```
//Store everything in terms of milliseconds.
ms = (seconds*1000 + milliseconds);

//Revolutions per minute.
rpm = (((float)60)*((float)1))/((float)ms/(float)1000);
= (60 seconds / 1 minute)(1 revolution completed)/(x seconds transcurred)
```

The code used for detecting movement of the wheel was implemented in the second laboratory. The components added were the following.

```
//Initiate timer with 1000 Hz frequency or millisecond frequency.
void initTimerModule(){
	SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
	SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN
				| SYSCTL_XTAL_16MHZ);
	TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
	TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet()/1000);//
	IntEnable(INT_TIMER0A);
	TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);//
	TimerEnable(TIMER0_BASE, TIMER_A);
}

//Init the interrupt framework.
void initInterruptModule(){
	IntMasterEnable(); //Interrupts on controller.
	IntEnable(INT_GPIOE); //Interrupts on port E.
	HWREG(0x40024408)= 0x06; //Interrupts on both edges.
	IntMaskEnable();  //Disable the mask.
}
```

```
int milliseconds;
int seconds;
int write;
int stopped;

int fourth = -1 ;
int third = -1 ;
int second = -1;

int direction ;
int count = 0;
int rpm = 0;

//Wheel moved.
void movedABlock(){
	SysCtlDelay(20000);
	//Extract both data and raw interrupt status for double verification.
	int ris = HWREG(0x40024414) & 0x06;
	int data = HWREG(0x400243FC) & 0x06;

	//If both are equal then it moved a block.
	if(data == 0x06 || ris == 0x06){
		count++;
		//A complete turn was done.
		if(count >= 7){
			//Update UI.
			int ms = (seconds*1000 + milliseconds);
			rpm = (((float)60)*((float)1))/((float)ms/(float)1000);
			count = 0;
			seconds = 0;
			milliseconds = 0;
			stopped = 0;// To reset speed if
					// a second has transcurred without
					// rotation. Here it indicates it is rotating.
		}
	}
	//Moved right.
	else if(ris == 0x04){
		direction = 1 ; //Set direction.
	}
	//Moved left.
	else if(ris == 0x02){
		direction = -1; //Set direction.
	}

	//Reset interrupt flags..
	IntFinish();
}
```

```
//A millisecond transcurred.
void timerCount();
	TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); //Clear the interrupt
	milliseconds++ ; //Increment milliseconds.
	if(milliseconds >= 1000){
		seconds++; //Add second.
		milliseconds = 0;
		if(stopped == 1){
			// To reset speed if a second has transcurred without
			// rotation.
			rpm = 0 ;
		}
	}
	if(milliseconds == 500){
		// To reset speed if a second has transcurred without rotation.
		stopped = 1 ;
	}
}
//Write the rpm to the LCD.
void tacometerWriteRPM(){
	lcdCursorHome();
	lcdWriteString("RPM: .");
	int t_rpm = rpm;
	//Fourth Digit.
	if(t_rpm >=1000){
		fourth = t_rpm/1000;
		lcdWriteLetter(fourth+48);
		fourth*=1000;
		t_rpm-=fourth;
	}
	else
		lcdWriteLetter(0+48);
	//Third Digit.
	if(t_rpm >= 100){
		third = t_rpm/100;
		lcdWriteLetter(third+48);
		third*=100;
		t_rpm-=third;
	}
	else{
		lcdWriteLetter(0+48);
	}
	//Second Digit.
	if(t_rpm >= 10){
		second=t_rpm/10;
		lcdWriteLetter(second+48);
		second*=10;
		t_rpm-=second;
	}
	else
		lcdWriteLetter(0+48);

	//First Digit.
	lcdWriteLetter(t_rpm+48);

}
```

```
//To write the direction it simply uses the direction variable.
void tacometerWriteDirection(){
      lcdCursorHomeDown();
      lcdWriteString("      .");
      if(direction == -1) lcdWriteLetter('<');
      else lcdWriteLetter(' ');
      lcdWriteString("  .");
      //CW
      if(direction == 1) lcdWriteLetter('>');
      else lcdWriteLetter(' ');

}
```

Experiment 3 - Tacometer - Software Plan

Juan Lebron Betancourt
Anthony Llanos Velazquez