Code Review

Habiendo desarrollado el código del SPARC®, se mostró a los diferentes programadores de los demás equipos lo obtenido para con ello hacer una revisión de código en la cual se pudiera enriquecer mutuamente el trabajo de cada equipo, situación que fue de gran ayuda pues sirvió para detectar algunas anomalías en la programación en un corto periodo de tiempo. Los archivos que fueron modificados después de la revisión de código fueron: FuncionesMenu.C, main.C y MotoresXYZ.C, los principales cambios realizados se muestran a continuación:

FuncionesMenu.C:

Antes

```
11 oid moverHaciaXY(void) {

12    uint16_t validarCoordX = 0;
13    uint16_t validarCoordY = 0;
14-   //corregirCoord:
15    printf("X:\n");
16-   coordXCentenas = receiveNum();
17-   coordXDecenas = receiveNum();
18-   coordXUnidades = receiveNum();
19    printf("Y:\n");
20-   coordYCentenas = receiveNum();
21-   coordYDecenas = receiveNum();
22-   coordYUnidades = receiveNum();
23    validarCoordX = ((coordXCentenas - 48)*100)+((coordXDecenas - 48)*10)+(c
24    validarCoordY = ((coordYCentenas - 48)*100)+((coordYDecenas - 48)*10)+(c
25-   if ((validarCoordX > 300) || (validarCoordY > 300)) {
```

Después

```
11 oid moverHaciaXY(void) {
12+   uint8_t charNoNumerico = 0;
13    uint16_t validarCoordX = 0;
14    uint16_t validarCoordY = 0;

15    printf("X:\n");
16+   coordXCentenas = receive();
17+   coordXDecenas = receive();
18+   coordXUnidades = receive();
19    printf("Y:\n");
20+   coordYCentenas = receive();
21+   coordYDecenas = receive();
22+   coordYUnidades = receive();
23    validarCoordX = ((coordXCentenas - 48)*100)+((coordXDecenas - 48)*10)
24    validarCoordY = ((coordYCentenas - 48)*100)+((coordYDecenas - 48)*10)
25+   if (coordXCentenas > 57 || coordXCentenas < 48) {
26+       //Si el caracter recibido no es valido se activa la bandera charN
27+       charNoNumerico = 1;
28+   }
29+   if (coordXDecenas > 57 || coordXDecenas < 48) {
30+       //Si el caracter recibido no es valido se activa la bandera charN
31+       charNoNumerico = 1;
32+   }
33+   if (coordXUnidades > 57 || coordXUnidades < 48) {
34+       //Si el caracter recibido no es valido se activa la bandera charN
35+       charNoNumerico = 1;
36+   }
37+   if (coordYCentenas > 57 || coordYCentenas < 48) {
38+       //Si el caracter recibido no es valido se activa la bandera charN
39+       charNoNumerico = 1;
40+   }
41+   if (coordYDecenas > 57 || coordYDecenas < 48) {
42+       //Si el caracter recibido no es valido se activa la bandera charN
43+       charNoNumerico = 1;
44+   }
45+   if (coordYUnidades > 57 || coordYUnidades < 48) {
46+       //Si el caracter recibido no es valido se activa la bandera charN
```

El cambio que se puede observar en estas líneas de código es que en un inicio cuando se recibía un carácter, se verificaba que este fuera válido y no se aceptaba otro carácter hasta que se obtuviese uno válido, el cambio que se hizo es que ahora el error de validación de los caracteres es mostrado hasta después de haber introducido los 6 números correspondientes a las coordenadas X y Y, si hubo algún error se regresa al menú principal y ya no se queda en espera de un carácter válido como en el código correspondiente al “antes”.

Antes *Después*

```
void modificarZ(void) {
    uint8_t OkEncendido = 0;

    while (OkEncendido == 0) {
        if (PORTAbits.RA5 == 1) {
            __delay_ms(10);
            if (PORTAbits.RA5 == 1) {
                printf("\nSaliendo\n");
                apagarZ();

                OkEncendido = 1;
            }
        }
        if (PORTCbits.RC4 == 1) {
            __delay_ms(10);
            if (PORTCbits.RC4 == 1) {
                printf("MovArriba\n");
                do {
                    moverZArriba();
                } while (PORTCbits.RC4 == 1);

            }
        }
        if (PORTCbits.RC5 == 1) {
            __delay_ms(10);
            if (PORTCbits.RC5 == 1) {
                printf("MovABajo\n");
                do {
                    moverZAbajo();
                } while (PORTCbits.RC5 == 1);

            }
        }
    }
```

```
void modificarZ(void) {
    uint8_t OkEncendido = 0;
    enableMotoresZ = 1;
    while (OkEncendido == 0) {
        if (PORTAbits.RA5 == 1) {
            __delay_ms(10);
            if (PORTAbits.RA5 == 1) {
                printf("\nSaliendo\n");
                apagarZ();
                enableMotoresZ = 0;
                OkEncendido = 1;
            }
        }
        if (PORTCbits.RC4 == 1) {
            __delay_ms(10);
            if (PORTCbits.RC4 == 1) {
                printf("MovArriba\n");
                do {
                    moverZArriba();
                } while (PORTCbits.RC4 == 1);
                apagarZ();
            }
        }
        if (PORTCbits.RC5 == 1) {
            __delay_ms(10);
            if (PORTCbits.RC5 == 1) {
                printf("MovABajo\n");
                do {
                    moverZAbajo();
                } while (PORTCbits.RC5 == 1);
                apagarZ();
            }
        }
    }
```

La segunda corrección mayor que se le hizo al programa fue cuando uno de los compañeros noto que en las rutinas de inicialización de los motores se habilitaba el enable pero después de mover o accionar los motores se dejaba encendido y en ninguna línea se apagaba, en la imagen del “después” se puede observar como ahora se deja el enable de los motores de Z en 0 al salir de la función para ajustar la base, adicional a ello en la programación anterior si se presionaba el botón de subir o bajar la base subía o bajaba hasta que se presionaban el botón de OK, ahora la base solo sube o baja mientras se presione el botón y se regresa al menú principal hasta que se presiona el botón de OK.

Main.C:

Antes *Después*

```
    if (overFlowTMR0 == 1) {
        if (destinoHomeY == 0) {
            PWM_DutyCycleCCP2(0);
        }
        CurrentPosY = coordinates.yWanted; //Se actualiza el valor actual
        printf("Interrupcion TMR0, llegaste a coordenada deseada Y\n");

        sparcEnMovimientoY = 0;

        overFlowTMR0 = 0;
    }
    if (overFlowTMR1 == 1) { //Si el TMR1 registro OverFlow
        if (destinoHomeX == 0) {
            PWM_DutyCycleCCP1(0);
        }
        CurrentPosX = coordinates.xWanted; //Se actualiza el valor actual
        printf("Interrupcion TMR1, llegaste a coordenada deseada X\n");

        sparcEnMovimientoX = 0;

        overFlowTMR1 = 0; //Se apaga la interrupcion del TMR1
    }
    if (intLimitSwitch3Esquinas) {
        __delay_ms(10);
        if (pinLimitSwitch3Esquinas == 1) {
            printf("Ocurrio la interrupcion limitSwitch3Esquinas\n");
        }
        moverHomeX();
        moverHomeY();

        intLimitSwitch3Esquinas = 0; //Se apaga la bandera INT0 de interru
    }
    if (intLimitSwitchHomeX == 1) {
        __delay_ms(10);
        if (pinlimitSwitchHomeX == 1) {
            printf("Ocurrio la interrupcion limitSwitchHomeX\n");
        }
        PWM_DutyCycleCCP1(0);
        CurrentPosX = 0;
```

```
    if (overFlowTMR0 == 1) {
        if (destinoHomeY == 0) {
            PWM_DutyCycleCCP2(0);
        }
        CurrentPosY = coordinates.yWanted; //Se actualiza el valor actual
        printf("Interrupcion TMR0, llegaste a coordenada deseada Y\n");
        printf("Ok,coorY\n");
        sparcEnMovimientoY = 0;
        enableMotorY = 1;
        overFlowTMR0 = 0;
    }
    if (overFlowTMR1 == 1) { //Si el TMR1 registro OverFlow
        if (destinoHomeX == 0) {
            PWM_DutyCycleCCP1(0);
        }
        CurrentPosX = coordinates.xWanted; //Se actualiza el valor actual
        printf("Interrupcion TMR1, llegaste a coordenada deseada X\n");
        printf("Ok,coorX\n");
        sparcEnMovimientoX = 0;
        enableMotorX = 1;
        overFlowTMR1 = 0; //Se apaga la interrupcion del TMR1
    }
    if (intLimitSwitch3Esquinas) {
        __delay_ms(10);
        if (pinLimitSwitch3Esquinas == 1) {
            printf("Ocurrio la interrupcion limitSwitch3Esquinas\n");
        }
        moverHomeX();
        moverHomeY();
        OcurrioIntEsq = 1;
        intLimitSwitch3Esquinas = 0; //Se apaga la bandera INT0 de interru
    }
    if (intLimitSwitchHomeX == 1) {
        __delay_ms(10);
        if (pinlimitSwitchHomeX == 1) {
            printf("Ocurrio la interrupcion limitSwitchHomeX\n");
        }
        PWM_DutyCycleCCP1(0);
        CurrentPosX = 0;
```

La corrección que se hizo en esta parte del main fue muy parecida a la hecha en la imagen pasada, cuando se llega a la coordenada X y Y deseada se deshabilita el enable para los motores de X y Y como medida de protección para el usuario y para el SPARC®.

MotoresXYZ.c:

Antes

```
    //Procedimiento Y

    coordinates.yWanted = ((coordYCentenas - 48)*100)+((coordYDecenas - 48)
    yToAdvance = (abs(coordinates.yWanted - CurrentPosY))*5; //Y por avanza
    if (coordinates.yWanted > CurrentPosY) { //Si el valor por recorrer es
        dirMotorY = 0; //El motor se mueve hacia un lado
    } else if (coordinates.yWanted < CurrentPosY) {//Si el valor por recorr
        dirMotorY = 1; //El motor se mueve al otro lado
    }
    if (coordinates.yWanted != CurrentPosY) {
        printf("yToAdvance is: ");
        send(yToAdvance);
        send('\n');
        sparcEnMovimientoY = 1;
        //Se activa la bandera de que se estara moviendo el eje Y
        setNumPasosY(yToAdvance);
        //Se utiliza el timer 0 como contador de eventos para contar los pu
        //por el PWM para avanzar la distancia necesaria
        //PWM_DutyCycleCCP1(0);
        //Se deja el ciclo de trabajo de X en 0
        PWM_DutyCycleCCP2(50);
        //Se deja el ciclo de trabajo de Y EN 50 %
    } else printf("E,coorAcY\n");
}

void moverHaciaX(uint8_t coordXCentenas, uint8_t coordXDecenas, uint8_t coo
    while (sparcEnMovimientoX == 1) {
        //No se hace nada hasta que termine de moverse a la coordenada ante
    }
    //Procedimiento X
```

Después

```
    //Procedimiento Y
    enableMotorY = 0;
    coordinates.yWanted = ((coordYCentenas - 48)*100)+((coordYDecenas - 48)
    yToAdvance = (abs(coordinates.yWanted - CurrentPosY))*5; //Y por avanza
    if (coordinates.yWanted > CurrentPosY) { //Si el valor por recorrer es
        dirMotorY = 1; //El motor se mueve hacia un lado
    } else if (coordinates.yWanted < CurrentPosY) {//Si el valor por recorr
        dirMotorY = 0; //El motor se mueve al otro lado
    }
    if (coordinates.yWanted != CurrentPosY) {
        printf("yToAdvance is: ");
        send(yToAdvance);
        send('\n');
        sparcEnMovimientoY = 1;
        //Se activa la bandera de que se estara moviendo el eje Y
        setNumPasosY(yToAdvance);
        //Se utiliza el timer 0 como contador de eventos para contar los pu
        //por el PWM para avanzar la distancia necesaria


        PWM_DutyCycleCCP2(50);
        //Se deja el ciclo de trabajo de Y EN 50 %
    } else printf("E,coorAcY\n");
}

void moverHaciaX(uint8_t coordXCentenas, uint8_t coordXDecenas, uint8_t coo
    while (sparcEnMovimientoX == 1) {
        //No se hace nada hasta que termine de moverse a la coordenada ante
    }
    //Procedimiento X
    enableMotorX = 0;
```

Las correcciones o mejoras hechas en esta parte del código es que se alinean las direcciones de los motores para que concuerden con las conexiones eléctricas hechas, así como también se habilita el enable de cada motor cuando es llamada la función de movimiento, este enable posteriormente es apagado por la rutina de interrupción que se muestra en el main.C en imágenes pasadas.

Antes

```
void presionarPantalla(uint8_t presionarZCentenas, uint8_t presionarZDecena
    coordinates.timesToPress = ((presionarZCentenas - 48)*100)+((presionarZ
    //Se calcula el numero de veces a presionar la pantalla
    if (coordinates.timesToPress != 0) {

        //ciclo para activar y desactivar el piston
        for (uint8_t toques = 0; toques < coordinates.timesToPress; toques+


            piston = 1;
            __delay_ms(100);
            piston = 0;
            __delay_ms(100);
        }
    }
```

Después

```
void presionarPantalla(uint8_t presionarZCentenas, uint8_t presionarZDecena
    coordinates.timesToPress = ((presionarZCentenas - 48)*100)+((presionarZ
    //Se calcula el numero de veces a presionar la pantalla
    if (coordinates.timesToPress != 0) {
        if (coordinates.timesToPress < 16) {
            //ciclo para activar y desactivar el piston
            for (uint8_t toques = 0; toques < coordinates.timesToPress; toq
                piston = 0;
                __delay_ms(500);
                piston = 1;
                __delay_ms(500);
                piston = 0;
            }
        }
    }
```

Por ultimo y gracias a que un compañero se dio cuenta de que el pistón se activaba y retraía demasiado rápido se aumentó el retardo entre cada activación y retracción del pistón, adicional a ello se agregó una condición en la que se limita el valor máximo de veces a accionar y retraer a 16, esto debido a que el pistón se calentaba mucho y podría traer problemas al resto del hardware.

Se da las gracias a los compañeros que contribuyeron a mejorar el software del SPARC®: Daniel Trevizo, Jesús Valenzuela y Arael Álvarez.