PIT. 44

CENTRO DE INGENIERIA Y DESARROLLO INDUSTRIAL

Proyecto Industrial Terminal

"Diseño y programación de un panel de control en la superficie para la tele-operación del robot submarino KAXAN del CIDESI"

PARA OBTENER LA ESPECIALIDAD EN

"TECNOLOGO EN MECATRONICA"

PRESENTA

Alumno: Ing. Luis Alberto Rendón Delgado

Tutor de Planta: Dr. Tomas Salgado Jiménez

Tutor Académico: M. en C. Luciano Nava Balanzar.

QUERETARO,QRO. 2012

TECNOLOGO EN MECATRONICA

Índice

Indice de Anexos.

CAPÍTULO 1. ANTECEDENTES.

La robótica submarina es un área de investigación de actual interés en nuestro país, debido a la inminente explotación de recursos petroleros en aguas profundas mexicana Este proyecto pretende probar que las instituciones de investigación mexicana pueden ayudar al desarrollo de tecnología para la explotación de petróleo en aguas profundas, y así disminuir la importación de tecnología extranjera que pudiera elevar los costos de producción del petróleo mexicano. Actualmente CIDESI está desarrollando un prototipo de robot submarino teleoperado conocido en el medio como ROV`s (Remotely Operated Vehicle).

Cabe mencionar que el ROV (ver Figura 1) ha contribuido al desarrollado de estudiantes de ingeniería, maestría, doctorado de diferentes áreas, con asesoría técnica de personal del CIDESI. Este proyecto es financiando con recursos propios de CIDESI.

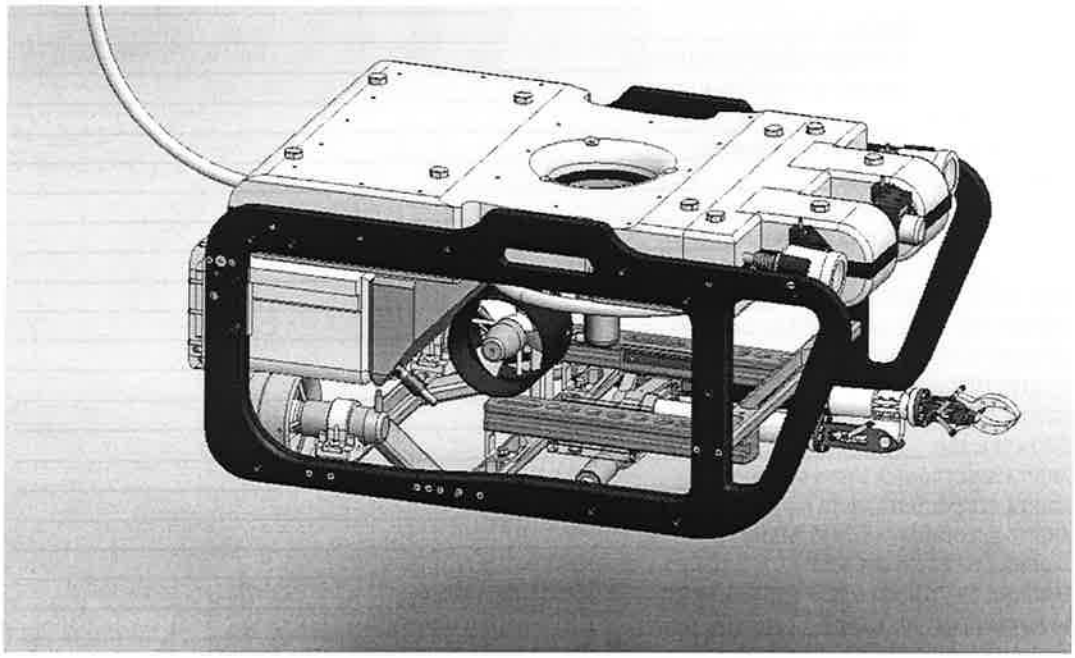


Figura 1 Robot Tele-operado de CIDESI.

Los robots submarinos *son una* herramienta indispensable para la planeación, construcción, operación y mantenimiento de las instalaciones en aguas profundas, *debido a que en aguas profundas la presión ejercida en los cuerpos cuando estos ganan profundidad es considerable, lo cual hacen imposible el uso de personal con equipo de*

Indice de Figuras.

1.1. DEFINICIÓN DEL TEMA.

Se desarrollará un panel o caja de control, que estará compuesto de dispositivos de control como son; palancas, botones y un microcontrolador. La salida de los dispositivos (palancas y botones) es adquirida por el microcontrolador, el cual es encargado de adquirir y procesar dichas señales para posteriormente enviarlas por medio del protocolo serial hacia una computadora industrial.

Debido a que se requiere visualizar la información se debe se debe generar una aplicación en LabView con dos objetivos; el primero es ver la información del panel de control y el segundo es que los movimientos que se presentan en el panel de control se transmitan de matera transparente o fiel a los movimientos del robot submarino.

1.2. JUSTIFICACIÓN.

Los robots submarinos tele-operados son controlados por un usuario humano desde la superficie con una consola de control. En dicha consola de control existe algunos instrumentos como son; una computadora, un monitor y un panel de control, este último es la interfaz con el usuario, dicha interfaz tiene que ser transparente es decir; tiene que transmitir los movimientos del panel de control a la computadora y de la computadora al ROV en tiempo real y 100% fieles, para lograr la operación del robot. Esté proyecto tiene la finalidad de crear el panel de control y la interfaz de adquisición de datos del panel de control a la computadora, actualmente se tiene como panel de control un control SONY PS3 (Dual Shock3 Sixaxis) y se pretende desarrollar un panel de control más robusto y que se adapte a las necesidades del ROV.

1.3. OBJETIVOS.

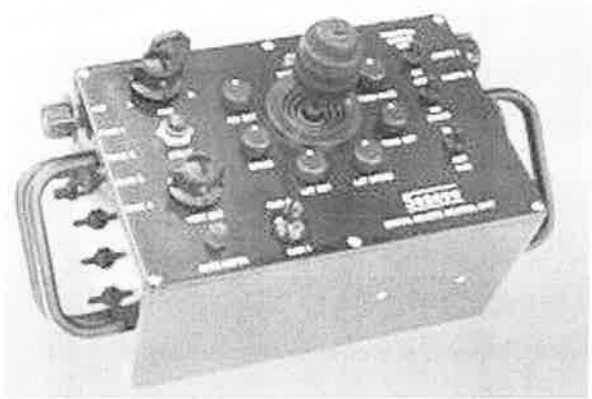
1.3.1. OBJETIVO GENERAL

- Diseñar y programar un panel de control en la superficie para la tele-operación del robot submarino KAXAN, para controlar los movimientos del robot submarino, con la ayuda de una interfaz gráfica en LabView.

buceo. Las principales aplicaciones de los robots submarinos en el mundo son numerosas, por citar algunas:

- Investigación geofísica y biomarina.
- Cartografía de fondo marino para el trazo de tuberías y cables submarinos.
- Inspección de estructuras y tuberías submarinas.
- Monitoreo de contaminación en el mar.
- Aplicaciones militares, por ejemplo la detección de minas y la vigilancia.
- Arqueología submarina.

En la actualidad existen diversas empresas que se dedican a fabricar este tipo de robots submarinos y todos ellos son de diversos tamaños y formas, al igual que sus unidades de control y controles de mano como se puede observar en las siguientes figuras.



1.3.2. OBJETIVOS ESPECIFICOS

- Diseñar la tarjeta electrónica para adquirir las señales analógicas y digitales.
- Programación del microcontrolador para adquirir datos y enviarlos a la computadora de control vía puerto serial.
- Ensamble e integración de las palancas y botones que componen el panel de control.
- Desarrollo de una interfaz grafica en LabView para desplegar la información de las palancas y botones, que serán adaptadas a los movimientos del robot submarino.

CAPÍTULO 2. Kit de Evaluación LaunchPad.

Este capitulo estudia las características de los Kit de Evaluación LaunchPad de Texas Instruments. Comienza con una exposición de los Kits LaunchPad en el mercado hasta el día de hoy y continua con la descripción del Kit de Evaluación MSP430^{tm}LaunchPad y finalmente se estudia los MSP430G2xx y se dan ejemplos de cómo programarlos

2.1. Descripción de los Kit de Evaluación LaunchPad.

Los kits de evaluación LaunchPad de **T**exas **I**nstruments (**TI**) son una gran herramienta de introducción a los microcontroladores de **TI**. Cuentan con todo lo necesario para iniciar el desarrollo de nuevas aplicaciones. La principal característica de los Kits de Evaluación de TI son su bajo costo y la robustez de sus dispositivos en comparación a otros Kits de otras compañías, es por esto que se opto por un Kit de Texas Instruments a los de Freescale, Microchip, Arduino, Parallax y Picaxe.

Otra ventaja que tiene los Kits de LaunchPad son los Booster Packs que son tarjetas modulares plug-in que se ajustan en la parte superior de los zócalos del LaunchPad. Estos módulos introducen nuevas funcionalidades a los kits de evaluación LaunchPad incluyendo wireless, sensores capacitivos, iluminación LED y mucho más. Muchos de estos BoosterPacks son compatibles con otros BoosterPacks lo que hace que se pueden colocar en diferentes muelles o apilados verticalmente (Ver Figura 2). Esto es especialmente útil cuando los BoosterPacks comparten una comunicación serial.

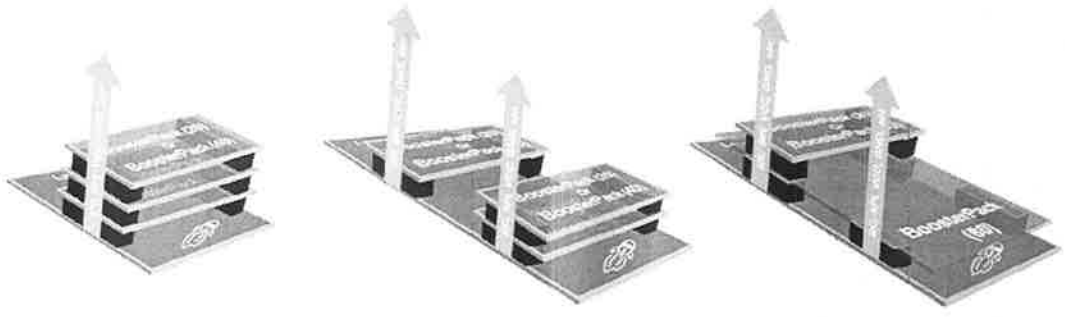


Figura 2 BoosterPacks para los Kits de Evaluación LaunchPAd.

Los MSP430 LaunchPad se apoya en varias herramientas de software para escribir y depurar el código que se va a cargar en el MCU (Microcontroladores). Actualmente existen tres software que recomienda Texas Instrumentes que son el Code Composer Studio, IAR Embedded Workbench y el MSPGCC (ver Figura 3). Siendo el IAR Embedded Workbench el software selecto para realizar el presente proyecto debido a que ya se contaba con conocimiento de dicho software.

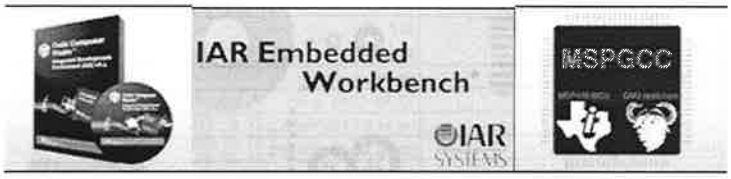


Figura 3 De izquierda a derecha: Code Composer Studio, IAR Embedded Workbench y el MSPGCC

En la Figura 4 se muestran en forma comparativa los tres Kits de Evaluación LaunchPad, especificando sus principales características.

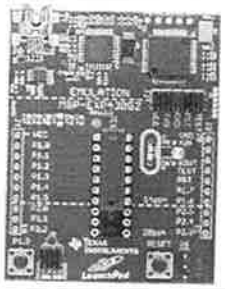
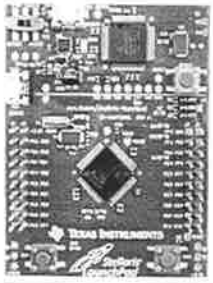
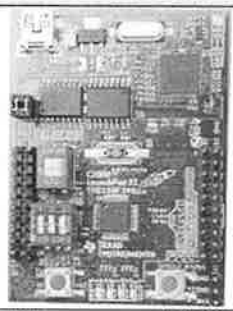
			
	Basado en los MSP430G2X gama de MCUs de ultrabajo consumo e integración analogica. EL original LauchPad	Basado en el C2000 Piccolo MCUs de Control en tiempo real	Basado en el Stellaris LM4F MCUs de Control de señales digitales. El 1er LauchPad basado en ARM
	$4.30 USD $55,18	$17 USD $218.15	$4.99 USD (por introducción) $64.03
	16 MHz	60 MHz	80 MHz
	16-bit MSP430 core	32-bit C28x core	32-bit ARM Cortex-M4F core
	Up to 16 kB (Flash)	Up to 64 kB (Flash)	up to 256KB Flash
	Up to 0.5 kB (SRAM)	Up to 12 kB (SRAM)	up to 32KB SRAM
	8ch, 10-bit ADC (200 ksps) 2x 16-bit Timers (with PWM capability) Interrupt-capable Input/Output pins Capacitive Touch I/O UART, SPI & I2C	8ch PWMs (4 hi-res PWM) 12-bit ADC, 12ch (4.6 Msps) Dual Comparator Interrupt-capable Input/Output pins UART, SPI & I2C	2KB EEPROM 12x 32-bit & 12x 16-bit Timers 11ch into 2x12-bit ADCs (1Msps) 2x Analog Comparators UART, I2C, SPI, CAN, USB Dev
	Cumple con 20-pin estándar BoosterPack	Cumple con 40-pin estándar BoosterPack	Cumple con 40-pin estándar BoosterPack
			
	LEDs y un switch de proposito general Switch de Reset Emulador para la programación / depuración	LEDs y un switch de proposito general Switch de Reset Emulador para la programación / depuración	Switches 2 usaurio y 1 de reset, LEDs: 1 RGB y 1 Alimentación Conectores BoosterPack y micro USB Emulador para la programación / depuración

Figura 4 Plataformas de evaluación LaunchPad.

Como se puede observar en la Figura 4 el sistema que más se adapta a nuestro proyecto es el MSP430 LaunchPad debido a sus periféricos: Ocho canales del ADC de 10bits, de los cuales solo se requiere de 6 canales; seis entradas/salidas digitales, de los cuales se requieren 6 y finalmente la comunicación serial UART. Aunque los otros Kits de evaluación cumplen con los periféricos se opto por el MSP430 LaunchPad debido a su bajo costo en comparación a los otros dos kits de desarrollo LaunchPad y a que los otros Kits tiene características que no se van a utilizar en un futuro.

2.2. Descripción del KIT MSP430 LaunchPad

El MSP430 LaunchPad es un kit de desarrollo que provee de todo el hardware y software que permite evaluar la gama MSP430G2x53 y además puede completar un proyecto entero en base al mismo. El MSP430 LaunchPad usa el IAR Embedded Workbench Integrated Development Environment (IDE) que provee la programación y la emulación completa con la opción de diseñar un sistema autónomo o utilizar la tarjeta que contiene al microcontrolador para integrarlo a un diseño existente. El puerto USB provee la energía para alimentar el MSP430 de ultra baja potencia.

Las 20 terminales del MSP430G2553 son accesibles en la tarjeta de pruebas MSP430 LaunchPad para una fácil corrección de errores e interactiva con los dispositivos periféricos. Adicionalmente, dos terminales digitales de I/O está conectadas a un diodo emisor de luz para una indicación visual y cuenta con dos switch uno de reset y otro conectado a una terminal digital de I/O. La Figura 5 muestra una fotografía del kit de desarrollo MSP430 LaunchPad, incluido el microcontrolador MSP430G2553, como se puede observar la tarjeta de evaluación cuenta con un socket DIP de 20-pines, el cual acepta todos los dispositivos de la línea MSP430G2xx de encapsulado DIP de 14 o 20 pines. Esto hace que sea fácil para sacar un dispositivo programado y colocarlo en una placa para una máxima flexibilidad.

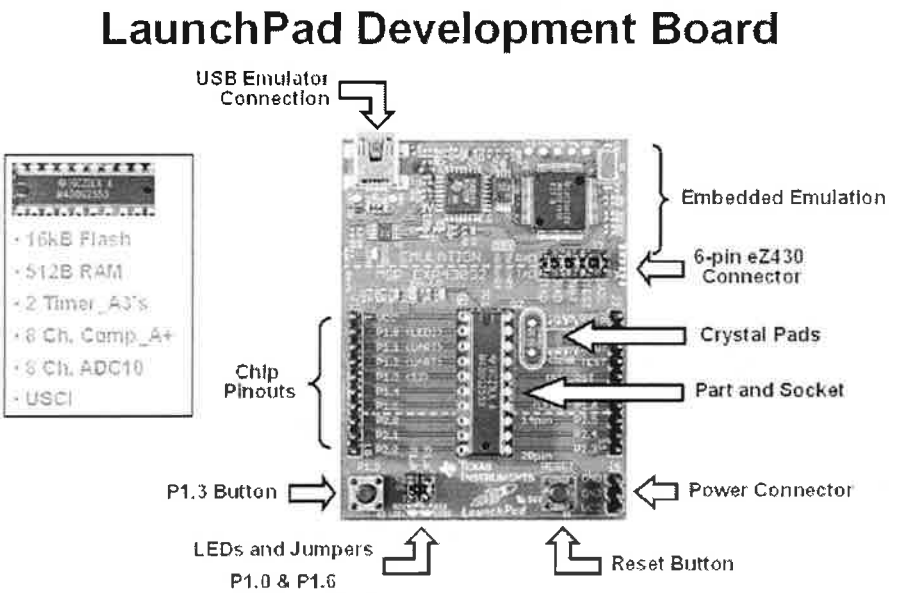


Figura 5 MSP430 LaunchPad

2.3. Contenido del KIT MSP430 LaunchPad.

EL MSP430 LaunchPad incluye todo lo necesario para iniciar a desarrollar aplicaciones, el software y los driver se puden descargar de la pagina oficial de TI http://www.ti.com/ww/en/launchpad/msp430_head.html, donde también se encuentran los ejemplos de código para este MCU y algunos recursos interesante. El kit de desarrollo contiene lo siguiente:

- LaunchPad emulador (MSP-EXP430G2)
- Dos dispositivos flash MSP430
- MSP430G2553: bajo consumo de 16-bit del microcontrolador MSP430 con 8 canales de ADC 10-bit, en el chip comparador, toque de sentido permitió I/Os, interfaz universal de comunicación serial, memoria flash de 16 kB y 512 bytes de RAM (precargado con un programa de ejemplo)
- MSP430G2452: bajo consumo de 16-bit del microcontrolador MSP430 con 8 canales de 10-bit ADC, en el chip comparador, toque de sentido activar E / S, interfaz serie universal, la memoria flash de 8 KB, y 256 bytes de SRAM
- 32.768 Khz. cristal reloj de Micro Cristal (http://www.microcrystal.com)
- Mini USB-B cable, 0,5 m
- Dos conectores hembra PCB de 10-pines.
- Cristal externo 32kHz
- Guía de inicio rápido
- Dos LaunchPad pegatinas

2.3.1. Descripción técnica del microcontrolador.

La arquitectura del MSP430 incluye una CPU tipo RISC (Reduced Instruction Set Computer) de 16 bits que se interconecta al resto de dispositivos mediante dos buses: El primero es el bus de direccionamiento de memoria común (llamado MAB - Memory Address Bus) y el segundo es el bus común para datos (llamado MDB - Memory Data Bus), estos buses permiten transmitir y recibir datos desde cualquiera de los dispositivos

integrados en el microcontrolador, tales como: La memoria RAM, La memoria Flash, Puertos 1 y 2, Timer_A2, USI, etc.. Además posee un sistema de reloj muy flexible que permite seleccionar de manera muy sencilla la fuente de reloj para la CPU y para los dispositivos periféricos. Todos los dispositivos periféricos son modulares y están correlacionados entre sí mediante los buses MAB y MDB. Las características más importantes de la familia MSP430x2xx son las siguientes:

- Tecnología de bajo consumo que extiende la vida de la batería.
- 0.8 uA en modo de reloj en tiempo real
- 250 uA al activarse el micro
- Sistema analógico de gran rendimiento ideal para mediciones de precisión.
- Temporizadores/Comparadores con capacidad de interrupción, es decir que pueden provocar la generación de interrupciones al alcanzar un valor determinado.
- CPU de arquitectura RISC de 16-bits que permite nuevas aplicaciones usando una fracción del tamaño de código.
- El diseño compacto del núcleo principal reduce el consumo de energía y el costo.
- Optimizado para la programación moderna de alto nivel.
- Solamente 27 instrucciones de núcleo y siete modos de direccionamiento.
- Capacidad de interrupción-vectorial extensiva.
- La Flash programable en sistema permite cambios al código, actualizaciones in situ y adquisición de datos de estado.

En la Figura 6 se muestra la arquitectura en forma general de los microcontroladores de la familia MSP430 y en la Figura 7 se muestra el esquema funcional en bloques del microcontrolador MSP430x20x3, con los periféricos específicos de este modelo.

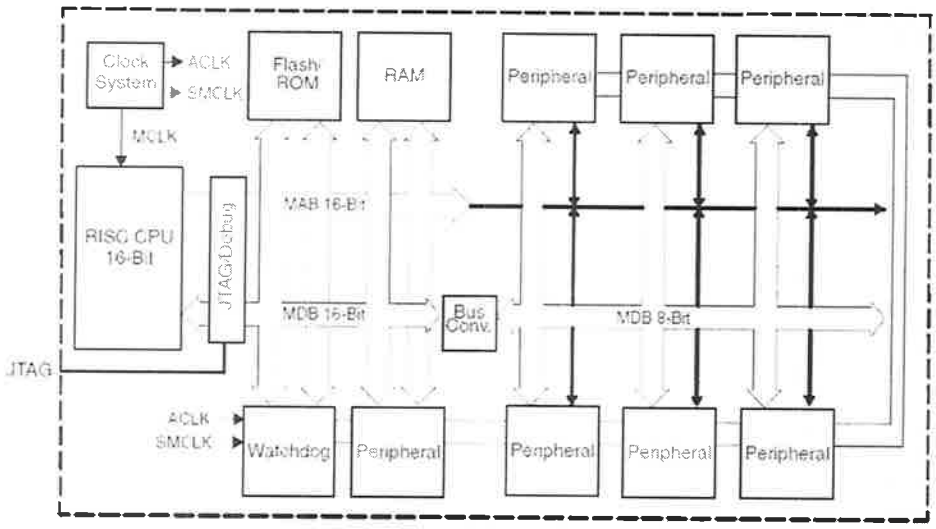


Figura 6 Arquitectura del MSP430.

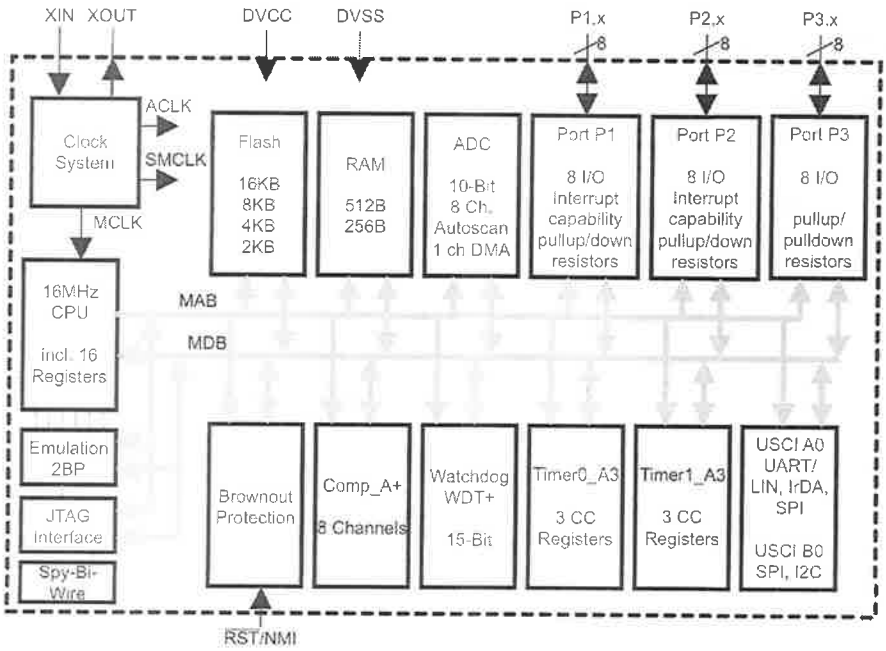


Figura 7 Esquema funcional en bloques del MSP430G2452

2.3.2. Terminales del dispositivo MSP430G2452.

La Figura 8 muestra la configuración de las terminales del MSP430G2452, que coincide con el microcontrolador usado en este proyecto, que es el MSP430G2452.

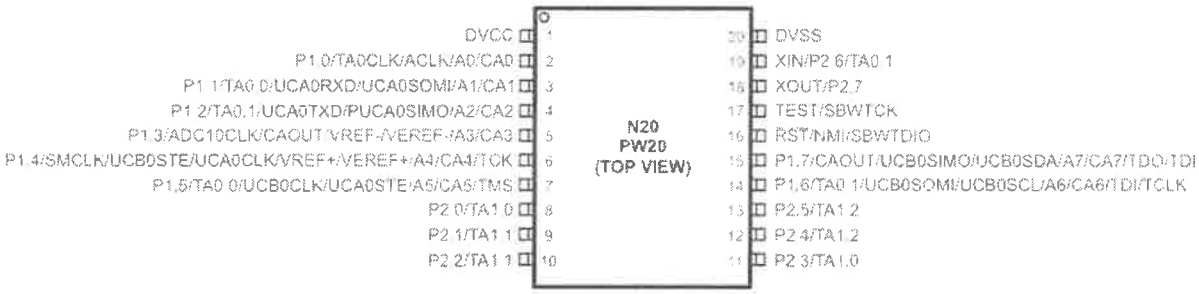


Figura 8 Detalle de las terminales del MSP430G2452

2.3.3. Programación del MSP430G2452.

A continuación se muestra un ejemplo de cómo configurar el convertidor analógico-digital ADC10 del MCU ver Figura 9. Para este ejemplo se utiliza el sensor de temperatura interno del MSP.

```
#include <msp430g2553.h>

#ifndef TIMER0_A1_VECTOR
#define TIMER0_A1_VECTOR     TIMERA1_VECTOR
#define TIMER0_A0_VECTOR     TIMERA0_VECTOR
#endif

volatile long tempRaw;

void FaultRoutine(void);

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;           // Stop watchdog timer
  P1DIR = 0x41;                       // P1.0&6 outputs
  P1OUT = 0;                          // LEDs off

  if (CALBC1_1MHZ ==0xFF || CALDCO_1MHZ == 0xFF)
   FaultRoutine();                    // If cal data is erased
                                      // run FaultRoutine()
  BCSCTL1 = CALBC1_1MHZ;              // Set range
  DCOCTL = CALDCO_1MHZ;               // Set DCO step + modulation

  BCSCTL3 |= LFXT1S_2;                // LFXT1 = VLO
  IFG1 &= ~OFIFG;                     // Clear OSCFault flag
  BCSCTL2 |= SELM_0 + DIVM_3 + DIVS_3; // MCLK = DCO/8

  while(1)
  {
   ADC10CTL1 = INCH_10 + ADC10DIV_0;   // Temp Sensor ADC10CLK
   ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON;
   _delay_cycles(5);                  // Wait for ADC Ref to settle
   ADC10CTL0 |= ENC + ADC10SC;        // Sampling & conversion start

   P1OUT = 0x40;                      // green LED on
   _delay_cycles(100);

   ADC10CTL0 &= ~ENC;
   ADC10CTL0 &= ~(REFON + ADC10ON);
   tempRaw = ADC10MEM;

   P1OUT = 0;                         // green LED off
   _delay_cycles(125000);
  }
}

void FaultRoutine(void)
 {
   P1OUT = 0x01;                      // red LED on
   while(1);                          // TRAP
 }
```

Figura 9 Programa de ADC10 en el MSP430G2553.

CAPÍTULO 3. Kit de Evaluación EZ430-RF2500-SE.

En este capítulo se estudia el Kit de Evaluación EZ430-RF2500-SE de Texas Instruments. Comienza con la descripción del Kit de Evaluación y se describe brevemente cada uno de sus componentes y finalmente se estudia el protocolo de comunicación SimpliciTI de Texas Instrumentes.

3.1. Descripción del Kit de Evaluación EZ430-RF2500

El eZ430-RF2500 es un sistema completo basado en un MSP430 USB de desarrollo inalámbrico que proporciona todo el hardware y software para evaluar el microcontrolador MSP430F2274 y el CC2500, transceptor inalámbrico de 2.4GHz. El eZ430-RF2500 utiliza el IAR Embedded Workbench como entorno de desarrollo integrado (IDE) o Código Esenciales Compositor (CCE) para escribir, descargar y depurar la aplicación.

La placa de destino eZ430-RF2500T es un sistema inalámbrico que se puede utilizar con la interfaz de depuración USB, como un sistema independiente con o sin sensores externos, o se puede incorporar en un diseño existente.

La nueva interfaz USB permite la depuración eZ430-RF2500 para enviar y recibir datos de forma remota desde su PC utilizando el MSP430 Aplicación UART.

A continuación se mencionan algunas características sobresalientes del eZ430-RF2500:

- Depuración de USB y una interfaz de programación que ofrece una instalación sin drivers y aplicaciones backchannel
- 21 pines disponibles para desarrollo
- MSP430 MCU de ultra baja potencia con el rendimiento de 16MHz
- Dos LEDs uno verde y otro rojo para la retroalimentación visual conectados a dos pines de propósito general E/S digital.
- Un switch de propósito general conectado a una entrada digital.
- Alcance de hasta 450 pies(137.16metros) a 10 Kbps.

3.2. Contenido del KIT EZ430-RF2500.

En primer lugar se detalla todo el material que viene con el kit eZ430- RF2500 que aparte de un CD con distintos programas como el IAR y ejemplos de códigos para el microcontrolador MSP430 se encuentran todos los elementos de hardware que podemos ver en la Figura 10 y la Figura 11.

Hardware incluidos:

- 2 eZ430-RF2500T tarjeta de destino
- 1 eZ430-RF USB interfaz de depuración
- 1 Soporte de baterías AAA con tarjeta de expansión (baterías incluidas)
- 1 CD-ROM contiene la documentación, herramientas de desarrollo y software de monitorización para el eZ430-RF2500.

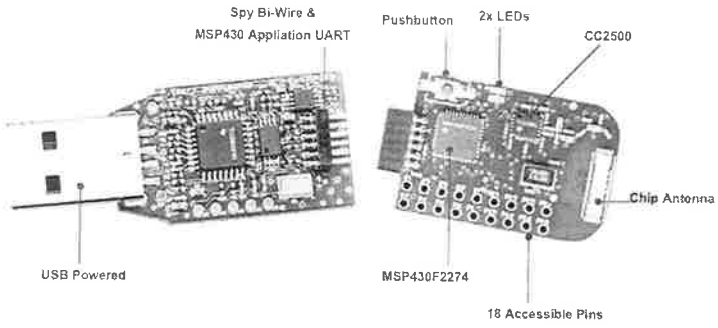


Figura 10 eZ430-RF2500

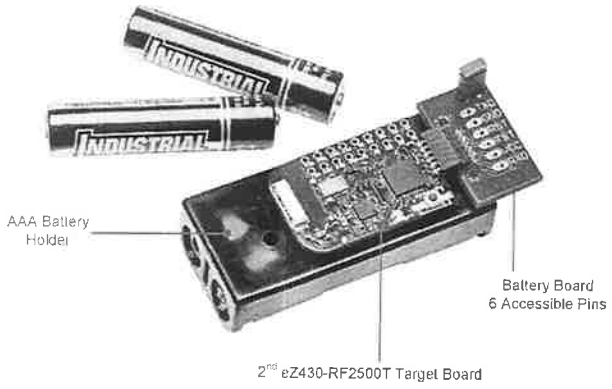


Figura 11 eZ430-RF2500 porta pilas

Contenido del CD-ROM:

- Guía de usuario de la familia MSP430F2xx, SLAU144
- MSP-FET430 FLASH Emulation Tool User's Guide, SLAU138
- eZ430-RF2500 Guía de usuario, SLAU227
- Code Composer Essentials (CCE), SLAC063
- IAR Embedded Workbench (Kickstart Version), SLAC050
- eZ430-RF2500 Sensor Monitor (Code and Visualizer), SLAC139

3.3. Descripción técnica del microcontrolador MSP430F2274.

Este microcontrolador es de la familia MSP430 de ultra baja potencia de Texas Instruments. Estos pueden ser usados en una gran variedad de aplicaciones. La arquitectura, se puede combinar en cinco modos de conservación de energía, optimizando la durabilidad de sus baterías, ayudando de este modo, a que su utilidad se extienda a una gran variedad de posibilidades.

En el dispositivo destaca una CPU de 16 bits RISC, registros de 16 bits y generadores constantes que contribuyen a maximizar la eficiencia del código.

El oscilador controlado digitalmente (DCO, Digitally Controlled Oscillator) permite cambiar el estado de los módulos de bajo consumo al modo activo en menos de 1μs.

La serie de MSP430x22x4 se basa en microcontroladores de ultra baja potencia, con dos temporizadores de 16 bits, un interfase de comunicación universal, un convertidor analógico-digital de 10 bits que utilizaremos en nuestro caso para tratar la señal analógica, un controlador de datos de transferencia (DTC, Data Transfer Controller), dos amplificadores operacionales de uso general en los dispositivos de MSP430x22x4 y 40 pines I/O ver Figura 12.

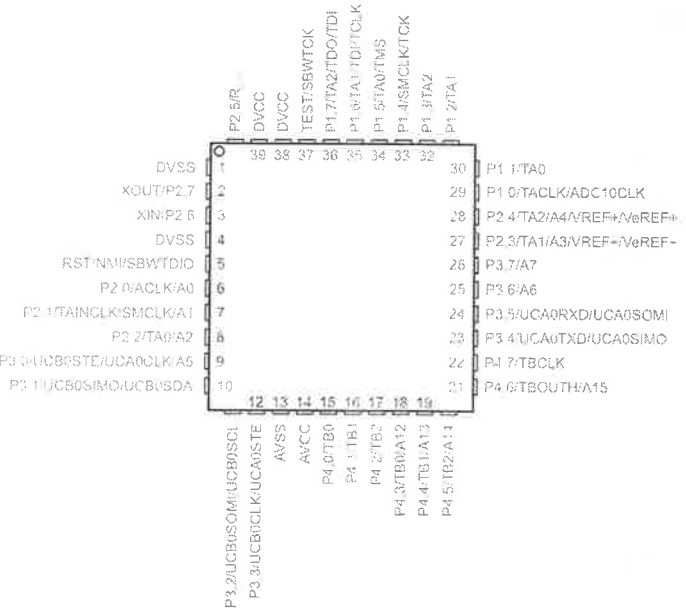


Figura 12 Detalle de las terminales del MSP430F2274

En la Figura 13 se muestran algunas características sobre salientes de MCU. Como se menciono anteriormente este MCU tiene 40 pines de los cuales solo 18 se encuentran disponibles en la tarjeta de evaluación. De los 18 pines 3 son ocupados para la energizar la tarjeta y 6 están disponibles para testear la comunicación entre el F2274 y el CC2500; por lo tanto solamente se tienen disponibles 9 puertos para desarrollo. Ver Figura 14 y Figura 15.

PARAMETER	MIN	TYP	MAX	UNIT
OPERATING CONDITIONS:				
Operating supply voltage	1.8		3.6	V
Operating free-air temperature range	-40		85	°C
CURRENT CONSUMPTION				
Active Mode at 1Mhz, 2.2V		270	390	µA
Standby Mode		0.7	1.4	µA
Off Mode with RAM Retention		0.1	0.5	µA
OPERATING FREQUENCY				
V_{cc} ≥ 3.3V			16	MHz

Figura 13 Especificaciones eléctricas.

Pin	Function	Description
1	GND	Ground reference
2	VCC	Supply voltage
3	P2.0 / ACLK / A0 / OA0I0	General-purpose digital I/O pin / ACLK output / ADC10, analog input A0
4	P2.1 / TAINCLK / SMCLK / A1 / OA0O	General-purpose digital I/O pin / ADC10, analog input A1 Timer_A, clock signal at INCLK, SMCLK signal output
5	P2.2 / TA 0 / A2 / OA0I1	General-purpose digital I/O pin / ADC10, analog input A2 Timer_A, capture: CCI0B input/BSL receive, compare: OUT0 output
6	P2.3 / TA 1 / A3 / VREF – / VeREF – / OA1I1 / OA1O	General-purpose digital I/O pin / Timer_A, capture: CCI1B input, compare: OUT1 output / ADC10, analog input A3 / negative reference voltage output/input
7	P2.4 / TA 2 / A4 / VREF + / VeREF + / OA1I0	General-purpose digital I/O pin / Timer_A, compare: OUT2 output / ADC10, analog input A4 / positive reference voltage output/input
8	P4.3 / TB0 / A12 / OA0O	General-purpose digital I/O pin / ADC10 analog input A12 / Timer_B, capture: CCI0B input, compare: OUT0 output
9	P4.4 / TB1 / A13 / OA1O	General-purpose digital I/O pin / ADC10 analog input A13 / Timer_B, capture: CCI1B input, compare: OUT1 output
10	P4.5 / TB2 / A14 / OA0I3	General-purpose digital I/O pin / ADC10 analog input A14 / Timer_B, compare: OUT2 output
11	P4.6 / TBOUTH / A15 / OA1I3	General-purpose digital I/O pin / ADC10 analog input A15 / Timer_B, switch all TB0 to TB3 outputs to high impedance
12	GND	Ground reference
13	P2.6 / XIN (GDO0)	General-purpose digital I/O pin / Input terminal of crystal oscillator
14	P2.7 / XOUT (GDO2)	General-purpose digital I/O pin / Output terminal of crystal oscillator
15	P3.2 / UCB 0SOMI / UCB 0SCL	General-purpose digital I/O pin USCI_B0 slave out/master in in SPI mode, SCL I2C clock in I2C mode
16	P3.3 / UCB 0CLK / UCA 0STE	General-purpose digital I/O pin USCI_B0 clock input/output / USCI_A0 slave transmit enable
17	P3.0 / UCB 0STE / UCA 0CLK / A5	General-purpose digital I/O pin / USCI_B0 slave transmit enable / USCI_A0 clock input/output / ADC10, analog input A5
18	P3.1 / UCB 0SIMO / UCB 0SDA	General-purpose digital I/O pin / USCI_B0 slave in/master out in SPI mode, SDA I2C data in I2C mode

*Pins 13 to 18 may be used to test the connection between the MSP430F2274 and CC2500.

Figura 14 Pines de la tarjeta de Evaluación.

Battery Board Pinouts

Pin	Function	Description
1	P3.4 / UCA 0TXD / UCA 0SIMO	General-purpose digital I/O pin / USCI_A0 transmit data output in UART mode (UART communication from 2274 to PC), slave in/master out in SPI mode
2	GND	Ground Reference
3	#RST/SBWTDIO	Reset or nonmaskable interrupt input. Spy-Bi-Wire test data input/output during programming and test
4	TEST/SBWTCK	Selects test mode for JTAG pins on Port1. The device protection fuse is connected to TEST. Spy-Bi-Wire test clock input during programming and test
5	VCC (3.6V)	Supply Voltage
6	P3.5 / UCA 0RXD / UCA 0SOMI	General-purpose digital I/O pin / USCI_A0 receive data input in UART mode (UART communication from 2274 to PC), slave out/master in in SPI mode

Figura 15 Pines de la tarjeta con porta pilas.

3.4. Descripción técnica del Transceptor CC2500.

El CC2500 es un transceptor diseñado para aplicaciones de bajo consumo a 2.4Ghz. Su circuitería trabaja dentro de la banda de frecuencia ISM para usos industriales, científicos y médicos (entre 2400MHz a 2483.5MHz) ver Figura 16.

Soporta varios tipos de modulación llegando hasta 500Kbaudios. Es capaz de almacenar datos, trabajar con paquetes e indicar la calidad del canal de enlace.

Los parámetros principales de las operaciones realizadas en el CC2500 pueden ser controlados con una interfase SPI. La gestión de cola de datos en el buffer del transceptor es gestionada mediante la política FIFO (First In, First Out,). Suele ser usado junto con un microcontrolador y otros componentes pasivos.

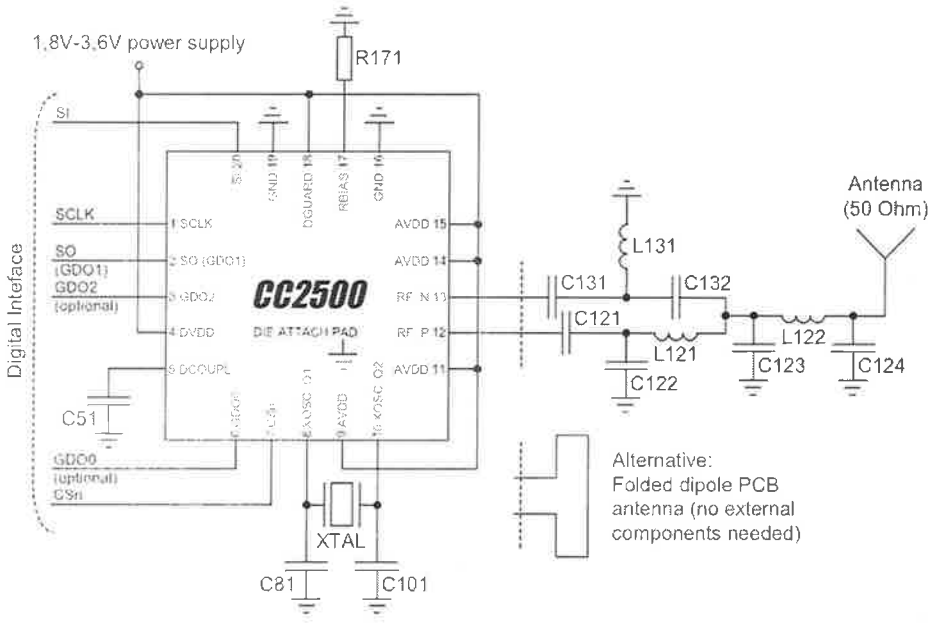


Figura 16 Aplicación típica y Circuito Evaluación (a excepción de los condensadores de desacoplamiento de suministro)

El transceptor CC2500 tiene diferentes estados en los cuales realiza funciones diferentes:

- Transmitir.
- Recibir.
- Idle: preparado para recibir, pero no lo hace (algunas funciones del Hardware se desactivan y consume menos energía).

- Sleep: Todas las partes del transceptor están apagadas. Hay que tener en cuenta que en ocasiones despertar el sistema puede consumir más energía que dejarlo en Idle.

Las características eléctricas del dispositivo se pueden ver en la Figura 17 y las principales características relacionadas con nuestro tipo de envío del transceptor CC2500 (Rendimiento RF) son:

- ♦ Alta sensibilidad (-104 dBm a 2,4 baudios, 1 % de tasa de error).
- ♦ Bajo consumo de corriente (13,3 mA en RX, 250 baudios, de entrada muy por encima de la sensibilidad límite).
- ♦ Potencia de salida programable hasta +1 dBm.
- ♦ Datos programable desde 1,2 a 500 kBaudios.
- ♦ Rango de frecuencia: 2400 a 2483,5 MHz.

PARAMETER	MIN	TYP	MAX	UNIT	CONDITION
OPERATING CONDITIONS:					
Operating supply voltage	1.8		3.6	V	
CURRENT CONSUMPTION					
RX input signal at the sensitivity limit, 250 kbps		16.6		mA	Optimized current
		18.8		mA	Optimized sensitivity
RX input signal 30 dB above the sensitivity limit, 250 kbps		13.3		mA	Optimized current
		15.7		mA	Optimized sensitivity
Current consumption TX (0 dBm)		21.2		mA	
Current consumption TX (-12 dBm)		11.1		mA	
RF CHARACTERISTICS					
Frequency range	2400		2483.5	MHz	
Data rate (programmable)	1.2		500	kbps	
Output power (programmable)	-30		0	dBm	
Sensitivity, 10 kbps		-99		dBm	Optimized current, 2-FSK, 230 kHz RX filter bandwidth, 1% PER
		-101		dBm	Optimized sensitivity
Sensitivity, 250 kbps		-87		dBm	Optimized current, 500 kHz RX filter bandwidth, 1% PER
		-89		dBm	Optimized sensitivity

Figura 17 Características Eléctricas.

3.5. Protocolo de red SimpliciTI.

Los transceptores CC2500 utilizan para su comunicación de radiofrecuencia el protocolo SimpliTI, protocolo propiedad de Texas Instruments. El protocolo de la red SimpliciTI fue diseñado para una fácil implementación con un mínimo de requisitos de recursos del microcontrolador. El protocolo se ejecuta con un MSP430 de ultra baja potencia de TI y con alguno de los varios transceptores de RF de TI.

SimpliciTI es un protocolo de radiofrecuencia que utiliza baja potencia y es capaz de integrar redes que pueden llegar a los 30 nodos. Al poder permanecer dormidos durante largos intervalos de tiempo hace que SimpliciTI sea un protocolo de baja potencia, también es de bajo coste al hacer uso de memorias flash inferiores a los 4 kBytes y memorias RAM menores de 512 Bytes. Por lo tanto, se puede decir que SimpliciTI reúne las principales propiedades que caracteriza ZigBee que son gran número de nodos y muy bajo consumo.

El protocolo de red SimpliciTI se puede utilizar en una amplia gama de aplicaciones de baja potencia, incluyendo alarmas de seguridad (detectores de humo, detectores de rotura de cristales, sensores de monóxido de carbono, y sensores de luz), la lectura automática de contadores (contadores de gas y medidores de agua), la automatización del hogar (electrodomésticos, puertas de garaje y dispositivos ambientales) y RFID activo.

El protocolo de red SimpliciTI se proporciona como código fuente bajo una licencia libre, sin derechos de autor. Para obtener información sobre compatibilidad, actualizaciones y la última versión del protocolo SimpliciTI, visite www.ti.com / SimpliciTI.

SimpliciTI soporta dispositivos finales denominados ED (End Device) que son los dispositivos que se comunican con el AP (Acces Point) encargado de identificar a todos los EDs de la red asignándoles un identificador cuando entran en comunicación con él, de este modo cuando un ED se comunique para enviar paquetes sabrá su procedencia según el identificador que anteriormente le ha otorgado.

Los mensajes serán almacenados por el AP. Una opción que proporciona SimpliciTI es extender la red mediante extensores de rango (RE) que permite ampliar la distancia con hasta cuatro saltos. El protocolo SimpliciTI viene completamente integrado al kit EZ430-RF2500 y viene con un ejemplo de una red que mide la temperatura del sensor interno del MSP430. En este caso la red creada está basada en la topología de una red UART

inalámbrica en la que se establece una conexión punto a punto de forma automática sin intervención del usuario. Ver Figura 18

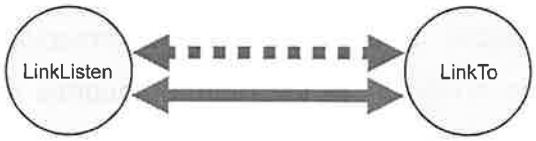


Figura 18 Esquema de la red UART inalámbrica creada con SimpliciTI

La Figura 19 muestra la arquitectura del protocolo, el modelo de SimpliciTI usa arquitectura de puerto, parecida al protocolo TCP/IP para comunicarse con la capa de red, que permite el manejo de funciones, luego esta el BSP (Board Support Package) para controlar la interfaz radio.

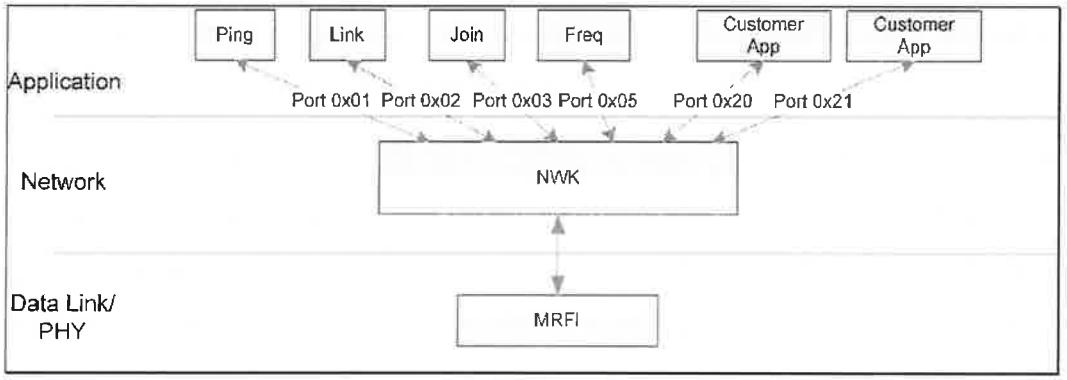


Figura 19 Protocolo SimpliTI

No hay nada regulado en la capa física, por lo tanto la frecuencia, el data rate y la modulación utilizada son libres a elegir por el diseñador del programa según el hardware que se utilice.

CAPÍTULO 4. Protocolo de comunicación RS-232C.

Este capitulo estudia el protocolo de comunicación serial. Comienza con una exposición de los conceptos básicos sobre la transmisión de información serie, sus formatos, parámetros e interfaces. A continuación se estudian los puertos serie disponibles en los microcontroladores y se dan ejemplos de cómo programarlos.

4.1. Interfaz RS-232C

El establecimiento de una comunicación a distancia requiere la participación de varios equipos que pueden agruparse en: Equipo terminal de datos (DTE: Data Terminal Equipment): Son los equipos que producen la señal de datos o son los receptores finales de la señal de datos. Equipo de comunicación de datos (DCE: Data Comunications Equipment): Son los equipos que adecuan la señal de datos al medio de transmisión utilizado o reciben esta señal del medio de transmisión ofreciéndola de forma apropiada al receptor final.

Un equipo terminal de datos muy común es la computadora, que puede, por ejemplo, generar la señal de datos con el formato asíncrono. Si esta señal ha de transmitirse a otro ordenador a través de un canal telefónico, entonces hay que utilizar algún equipo intermedio para adecuar la señal de datos al canal telefónico en el transmisor y en el receptor. Este equipo es denominado *módem* (**mod**ulator – **dem**oluator) y esta clasificado dentro de los equipos de comunicación de datos.

Según la anterior clasificación de los equipos, el esquema general de un sistema de comunicación de datos es ilustrado en la Figura 20.

La conexión entre el DTE y el DCE fue normalizada en el sexto decenio del siglo XX por el entonces llamado Comité Consultivo Internacional de Telefonía y Telegrafía (CCITT), hoy incorporado a la Unión Internacional de Telecomunicaciones (UIT). Una de las normas más utilizadas es la relativa a la interfaz de comunicación en modo asíncrono para velocidades bajas y medias, conocida como la Recomendación V.24 del Libro Blanco de 1969. Esta interfaz es conocida popularmente como RS-232C (Recommended Estándar número 232 revisión C) pues fue propuesta originalmente en 1962 por la EIA (Electrónic

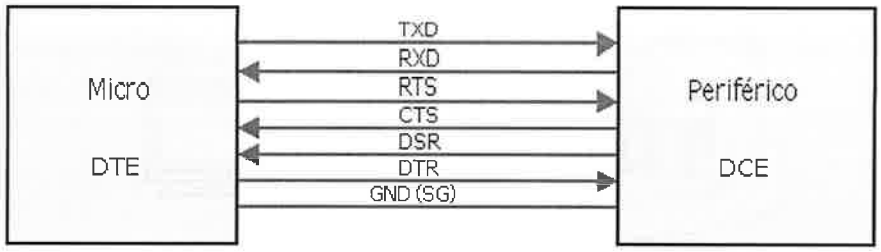
Industries Association) de los Estados Unidos, para la conexión entre equipos de datos a corta distancia (originalmente menos de 50 pies o 15.24metros) en un entorno ruidoso y la velocidad máxima de transmisión es de 20,000 bps. La revisión C es del año 1969; se han hecho otras revisiones, como la D en 1986, la E en 1991 y la F en 1997, pero a menudo se mantiene la designación RS-232C a pesar de los cambios. Esta interfaz se ha popularizado de tal forma que hasta hace poco, prácticamente todas las laptops estaban equipadas con una interfaz "RS-232" y su conector para la comunicación con un *módem* u otro periférico serie. Desde 2004, los únicos puertos serie de las laptops suelen ser USB (**U**niversal **S**erial **B**us).

Las señales de la interfaz RS-232C utiliza lógica negativa y no son compatibles con los niveles lógicos TTL, deben situarse dentro de los siguientes rangos:

- Nivel lógico 0: entre +3v y +15V con carga, hasta +25V sin carga.
- Nivel lógico 1: entre -3v y -15V con carga, hasta -25V sin carga.
- El área muerta entra +3 V y -3 V está designada para absorber ruido de línea.

Se utilizan conectores de 25 pines (DB25) o de 9 pines (DB9) siendo asignado el conector macho al DTE y el conector hembra al DCE. Las señales más utilizadas de la Interfaz RS-232C se muestran en la Figura 20.

Requerimientos de señal para mínima conexion full duplex



Requerimientos de señal típica para comunicacion full duplex

Figura 20 Señales más utilizada de la comunicación RS-232C.

Para una comunicación *full duplex* desde el USART del PIC se debe conectar un mínimo numero de señales, TX y RX así como la masa(GND). Los PIC utilizan señales TTL en el modulo USART por lo que debe utilizarse un conversor de nivel a RS-232, como el MAX232

En a mayoría de los Laptops, están desapareciendo los puertos serie. Como solución se puede utilizar cables de conversión SERIE-USB (ver Figura 21) que utilizan el Universal Serial Port (USB), el cual no se debe confundir con la utilización del modulo USB integrado en el PIC con gestión de comunicación USB

Figura 21 Convertidor USB- RS232

La comunicación serial es popular debido a que la mayoría de los computadores poseen uno o más puertos seriales, por lo que no se requiere hardware adicional a un cable para conectar el instrumento a un computador o dos computadores juntos ver Figura 22.

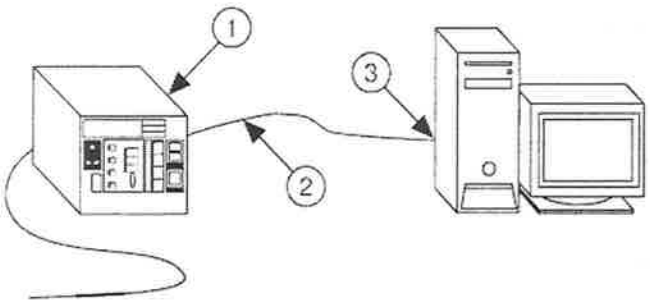


Figura 22 1. Instrumento RS-232. 2 Cable RS-232. 3 Puerto Serial

La comunicación serial requiere que usted especifique los cuatro parámetros siguientes:

- La velocidad en baudios de la transmisión
- El número de bits de datos codificados a carácter
- La sensibilidad del bit opcional de paridad
- El número de bits de parada

Cada carácter transmitido es empaquetado en un marco de carácter que consiste de un solo bit de inicio seguido por los bits de datos, el bit opcional de paridad y el bit o bits de parada. La Figura 23 muestra un marco típico de carácter codificando la letra m.

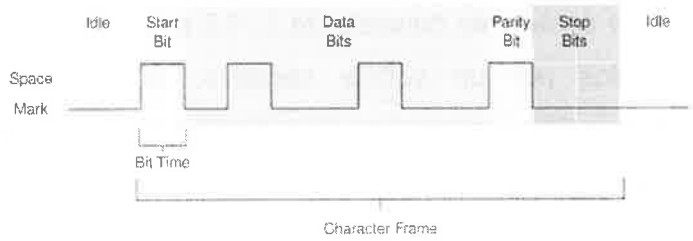


Figura 23 Trama de transmisión.

La velocidad en baudios es una medida de cuan rápido los datos son movidos entre instrumentos que emplean comunicación serial. RS-232 emplea solo dos estados de voltaje. En un esquema de codificación como el de dos estados, la velocidad en baudios es idéntica al máximo número de bits de información, incluyendo bits de control, que son transmitidos por segundo.

Un bit de inicio indica el principio de cada marco de carácter. Esta es una transición desde un voltaje negativo hasta uno positivo. Su duración en segundos es un recíproco de la velocidad en baudios. Si el instrumento está transmitiendo a 9,600 baudios, la duración del bit de inicio y cada subsecuente bit esta cerca de 0.104 ms. El marco total de carácter de once bits debe ser transmitido en cerca de 1.146 ms.

Los bits de datos son transmitidos al revés y hacia atrás. Esto es, se emplea lógica inversa y el orden de la transmisión es desde el bit menos significativo (LSB) hasta el bit más significativo (MSB). Para interpretar los bits de datos en un marco de caracteres, debe leer de derecha a izquierda y leer 1 para voltaje negativo y O para voltaje positivo. Esto

produce 1101101 (binario) o 6D (hexadecimal). Una tabla de conversión ASCII muestra que esta es la letra m.

Un bit de paridad opcional sigue los bits de datos en un marco de carácter. El bit de paridad, si está presente, también sigue lógica inversa, 1 para voltaje negativo y 0 para voltaje positivo. Este bit es incluido como un simple medio de control de error. Usted especifica el período de tiempo donde la paridad del instrumento debe ser par o impar. Si la paridad se elige impar, entonces el transmisor fija el bit de paridad en forma tal que se realice un número par de unos a través de los bits de datos y el bit de paridad. Esta transmisión emplea paridad impar. Existen cinco unos a través de los bits de datos, ya hay un número impar, así el bit de paridad se fija en 0.

La última parte del marco de caracteres consiste de 1, 1.5 o 2 bits de parada. Estos bits están siempre representados por un voltaje negativo. Si no se adelantan más transmisiones de caracteres, la línea permanece en condición negativa. La transmisión del siguiente marco de carácter, si hay, se anuncia por un bit de inicio de voltaje positivo.

4.2. Interfaz RS-232C en LabView.

Las funciones VISA Write y VISA Read trabajan con cualquier tipo de comunicación de instrumentos y son las mismas independiente que realice comunicación GPIB o serial. Sin embargo, debido a que la comunicación serial requiere de una configuración de parámetros extras, debe iniciar una comunicación del puerto serial con el VI VISA Configure Serial Port.

El VI VISA Configure Serial Port inicializa el puerto identificado por **VISA resource name** con las configuraciones especificadas. **timeout** fija el valor de tiempo fuera para la comunicación serial. **baud rate**, **data bits**, **parity** y **flow control** determinan los parámetros específicos de puerto serial. Los clusters de **error in** y **error out** dan las condiciones de error para este VI.

El siguiente ejemplo muestra como enviar el comando de solicitud de identificación *IDN? al instrumento conectado al puerto serial COM2. El VI VISA Configure Serial Port abre la comunicación con COM2 y lo fija a 9,600 baudios, 8 bits de datos, paridad impar, un bit de parada y software handshaking XON/XOFF. Entonces la función VISA Write envía el

comando. La función VISA Read lee el retorno hasta 200 bytes en el buffer de lectura y el VI Simple Error Handler verifica la condición de error.

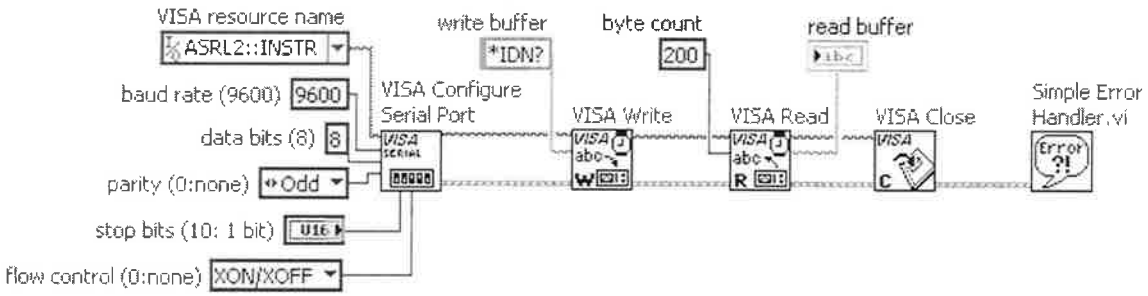


Figura 24 Configuración del puerto serial en LabView.

Nota Los VIs y funciones localizadas en la paleta **Functions»All Functions»Instrument I/O»Serial** también son empleadas para comunicación por puerto paralelo. Usted especifica el nombre de recurso VISA para que sea uno de los puertos LPT. Por ejemplo, puede emplear MAX para determinar que LPT 1 tiene un nombre de recurso VISA de ASRL10: : INSTR.

4.3. Interfaz RS-232C en MCU de Texas Instruments.

El estándar RS232 lleva desde hace ya mucho tiempo, es un protocolo sencillo, practico y conformado por pocos cables (en configuración mínima solo se requieren 2 cables, **RX** y **TX**), el programador ez430-RF2500 tiene un puente RS232 ver Figura 25 listo para ser usado, también el programador MSP40 LaunchPad cuenta con uno.

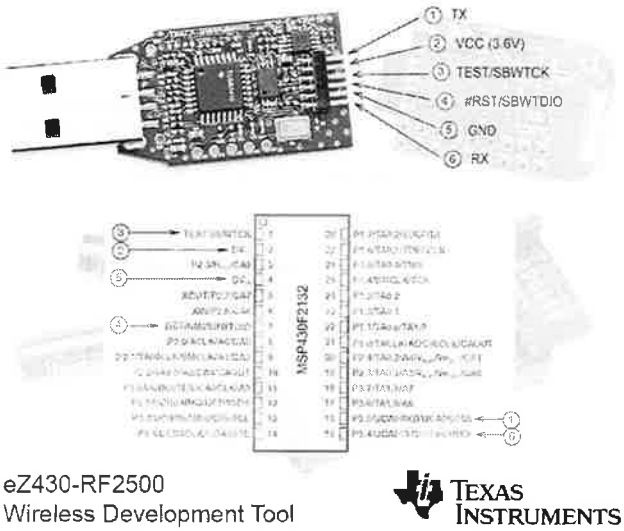


Figura 25 Herramienta de desarrollo eZ430-RF2500.

Para hacer uso de la interfaz hay un sinfín de programas que pueden hacer uso del mismo sin complicaciones, desde los más sencillos (Hyperterminal) hasta los más complejos (LabVIEW). Para este ejercicio usaremos el (Hyperterminal incluido en Windows XP):

Se abre la HyperTerminal y se crea una nueva conexión ver Figura 27. En nombre, se puede poner lo que sea, damos OK y obtendremos una ventana como esta:

Figura 26 Hyperterminal

Después se abre una ventana como se muestra en la Figura 27. En ella seleccionaremos el puerto COM que corresponde a nuestro programador (se puede ver en el Administrador de Dispositivos de Windows) para este caso es el COM13.

Figura 27 Configuración de la HyperTerminal.

Después de eso aparecerá otra ventana de configuración ver Figura 28.

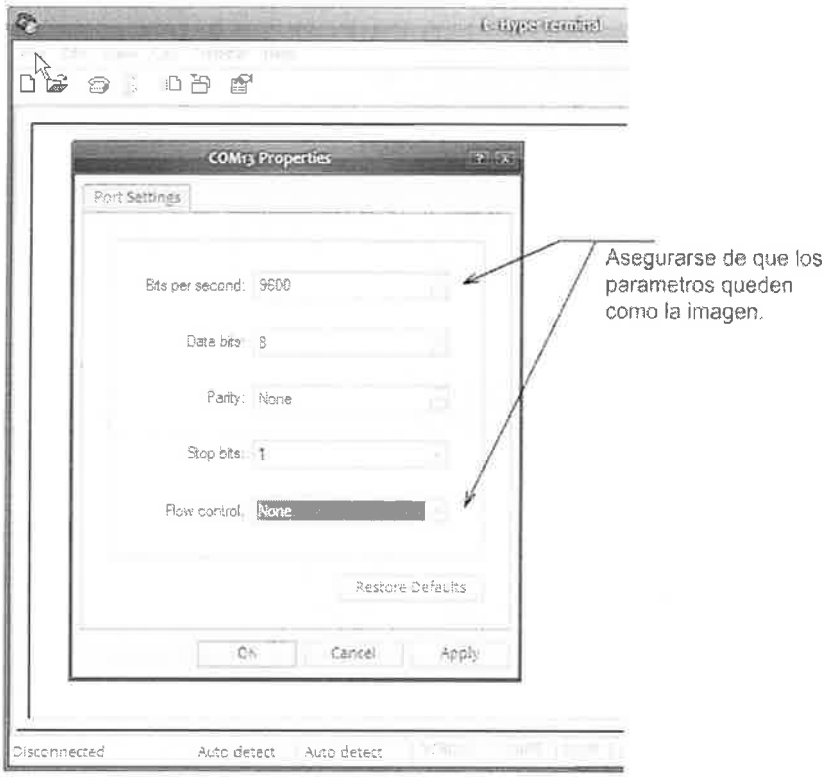


Figura 28 Configuración del COM

Finalmente presionaremos aplicar y después aceptar. Lo único que nos quedará será una pantalla en blanco, en ella podremos ver los mensajes que mandemos desde el microcontrolador por RS232.

En la Figura 29 se muestra un ejemplo de cómo programar un MCU con una comunicación UART (comunicación serial). El programa funciona como un repetidor, ya que envía (Tx) lo mismo que recibe(Rx). Para visualizarlo se puede utilizar la HyperTerminal o LabView, como se menciono anteriormente.

```
//******************************************************************************
//  MSP430F21x2 Demo - USCI_A0, 115200 UART Echo ISR, DCO SMCLK
//
//  Description: Echo a received character, RX ISR used. Normal mode is LPM0.
//  USCI_A0 RX interrupt triggers TX Echo.
//  Baud rate divider with 1MHz = 1MHz/115200 = ~8.7
//  ACLK = n/a, MCLK = SMCLK = CALxxx_1MHZ = 1MHz
//
//               MSP430F21x2
//            -----------------
//        /|\|              XIN  |
//         | |                   |
//         --|RST            XOUT|-
//           |                   |
//           |P3.4/UCA0TXD|------------>
//           |                   | 115200 - 8N1
//           |P3.5/UCA0RXD|<------------
//
//******************************************************************************
#include  "msp430x21x2.h"

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  if (CALBC1_1MHZ==0xFF || CALDCO_1MHZ == 0xFF)
  {
    while(1);                               // If calibration constants erased
                                            // do not load, trap CPU!!
  }
  BCSCTL1 = CALBC1_1MHZ;                    // Set DCO
  DCOCTL = CALDCO_1MHZ;
  P3SEL = 0x30;                             // P3.4,5 = USCI_A0 TXD/RXD
  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 = 8;                              // 1MHz 115200
  UCA0BR1 = 0;                              // 1MHz 115200
  UCA0MCTL = UCBRS2 + UCBRS0;               // Modulation UCBRSx = 5
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt

  __bis_SR_register(LPM0_bits + GIE);       // Enter LPM0, interrupts enabled
}

// Echo back RXed character, confirm TX buffer is ready first
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
  while (!(IFG2&UCA0TXIFG));                // USCI_A0 TX buffer ready?
  UCA0TXBUF = UCA0RXBUF;                    // TX -> RXed character
}
```

Figura 29 Ejemplo de comunicación RS232.

CAPÍTULO 5. Desarrollo

En este capitulo se presenta como fue que se desarrollo el prototipo del panel de control. En el primer subtema se presenta todo lo relacionado con el prototipo del panel de control alambrico, mientas que en el segundo subtema se presenta lo relacionado con el prototipo de panel inalámbrico desarrollado.

5.1. Diseño del panel de control.

Unos de los objetivos de crear el panel de control es el diseñar un control robusto y de alta calidad en base a los dispositivos con los que se cuentan. Dicho control debe tener dos joystick analógicos: uno de 2 ejes y otro de 3 ejes con dos botones, adicionalmente también deberá de contar con un potenciómetro lineal con su respectiva perilla, y 4 botones en el panel.

5.1.1. Lista de Material.

En la lista de material que se muestra a continuación se presenta las partes más importantes que forman el panel de control, así como un valor aproximado del costo total del panel, despreciando los material más significativos y el equipo que se utilizo para ensamblar el panel; como fue cable, termófil, soldadura, pasta, brocas, taladro, cautín, etc.

No. de Línea	*No. de Parte de Newark*	*Cantidad*	*Número De Parte*	*Descripción Del Producto*	*Precio*
1	89F4157	1	CONXALL 3282-4SG-3XX	CIRCULAR CONNECTOR FEMALE SIZE 20, 4POS CABLE; Connector Type:Circular Industrial; Series:Multi-Con-X; Connector Body Material:Nylon; Gender:Receptacle; Contact Gender:Socket; Connector Mounting:Cable; Connector Shell Size:20 ;RoHS Compliant: Yes	$12.29

2	 89F4158	1	CONXALL 3282-4PG-3XX	CIRCULAR CONNECTOR PLUG, SIZE 20, 4POS, CABLE; Connector Type:Circular Industrial; Series:Multi-Con-X; Connector Body Material:Nylon; Gender:Plug; Contact Gender:Pin; Connector Mounting:Cable; Connector Shell Size:20 ;RoHS Compliant: Yes	$5.88
3	 20M1714	1	VISHAY SPECTROL 132-0-0-203	POT, WIREWOUND, 20KOHM, 3%, 2.75W; Track Resistance:20kohm; Track Taper:Linear; No. of Turns:1; Shaft Diameter:6.34mm; Shaft Length:22.22mm; Resistance Tolerance:± 3%; Power Rating:2.75W; Potentiometer Mounting:Panel; No. of Gangs:1 ;RoHS Compliant: Yes	53.01
4	 91F3918	1	BUD INDUSTRIES PN-1335-DG	ENCLOSURE, JUNCTION BOX, PLASTIC, GRAY; Enclosure Type:Junction Box; Enclosure Material:ABS; Body Color:Grey; External Height - Imperial:10.43"; External Width - Imperial:7.28"; External Depth - Imperial:3.74"; IP Rating:66 ;RoHS Compliant: Yes	39.56
5	 90F742	2	BUD INDUSTRIES H-9111-B	Chrome Plated Equipment Handle; Fixing Centers:4"; Handle Material:Steel; Diameter:0.375"; Material:Steel ;RoHS Compliant: Yes	$10.38

6	 34C9068	1	EHC 407D2B	ROUND SKIRTED KNOB W/ ARROW IND, 6.35MM; Knob / Dial Style:Round Skirted with Indicator Arrow; Shaft Diameter:6.35mm; Knob Diameter:18.288mm; Shaft Type:Round; Knob Material:ABS; Body Material:ABS Plastic ;RoHS Compliant: Yes	$7.92
7	 679-2277-ND	1	MS-54W00S000	JOYSTICK SERIES III 2 BUTTONS	$364.42
8	 679-2269-ND	1	HFX2-11S10	JOYSTICK SERIES2 2AXIS NO BUTTON	$197.65
				USD:	$562.07
			PESOS:	$8,419.29	

5.1.2. Ensamble.

A la caja se le tomaron medidas para poder obtener una buena distribución de todos los componentes que integrarían el panel de control, la distribución por la que se opto se muestra en la Figura 30. Posteriormente de seleccionar la mejor distribución se siguió a marcar los lugares que ocuparía cada componente, para después realizar las perforaciones e instalación de cada uno de los dispositivos como se puede ver en la Figura 31.

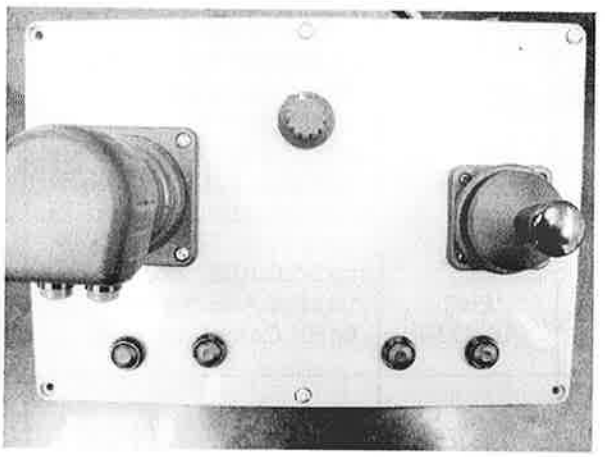


Figura 30 Panel de control. Vista Superior.

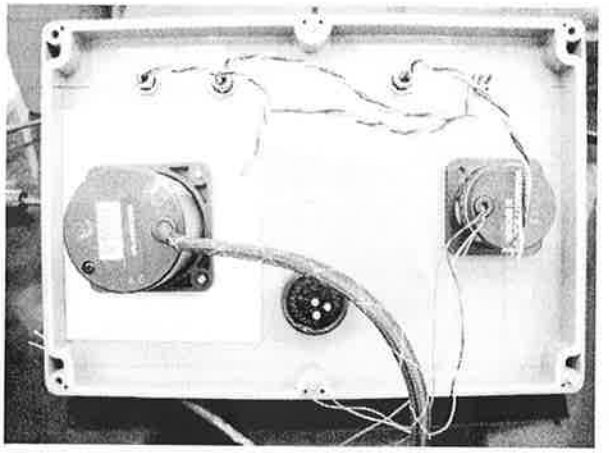


Figura 31 Panel de control. Vista Posterior.

Posteriormente a la instalación de cada uno de los dispositivos se prosiguió a verificar cada uno de ellos y sus respectivas señales que entregan. Después de la revisión se noto que el Joystick de 3 ejes y 2 botones se movía mucho, es decir era muy inestable, por lo que se opto por colocar una placa en la base del mismo para distribuir la fuerza a la que es sometido cuando se esta operando el joystick (ver Figura 31). En la Figura 32 se muestra el panel de control terminado con su conector y cable de alimentación. El panel de control es el mismo para el control alambrico y el inalámbrico, la única diferencia entre ellos es que cuando se utiliza alambricamente el MCU utiliza el protocolo de comunicación RS-232C y utiliza el cable ya que por el se envía la alimentación de 9V y el Tx y Rx necesarios para el protocolo de comunicación. Mientras que para el modo inalámbrico el cable que se muestra se utiliza para alimentar toda la electrónica que contiene el panel de control. En un futuro se piensa instalar unas baterías recargables y utilizar el cable para recargas las baterías, sin necesidad de destapar el panel, y al momento de operar el panel se usa sin el cable.

Figura 32 Panel de Control Terminado

5.2. Diseño del control alambrico.

El panel de control alambrico esta basado en el MSP430 LaunchPad este kit de desarrollo como ya se había mencionado es el que mejor se ajusta a las necesidades del proyecto. El MCU que se utilizo fue el M430G2553 debido a sus características que tiene, a continuación se menciona los módulos utilizados dentro del MCU y como se conectaron.

- Entradas/Salidas Digitales: Se utilizaron entradas digitales para poder conectar cada uno de los botones que tienen el panel de control. Se utilizo todo el puerto 2 del microcontrolador para ello.

E/S Digital	Pin MSP430G2553	Dispositivo conectado
Entrada Digital	**P2.0**	**Joystick 3 ejes; Botón 1**
Entrada Digital	**P2.1**	**Joystick 3 ejes; Botón 2**
Entrada Digital	**P2.2**	**Botón 1**
Entrada Digital	**P2.3**	**Botón 2**
Entrada Digital	**P2.4**	**Botón 3**
Entrada Digital	**P2.5**	**Botón 4**

- El ADC10: Se utilizaron 6 canales del ADC10 para convertir la señal analógica a una digital. El orden en que se conectaron cada uno de ellos fue el siguiente:

Canal de ADC10	Pin MSP430G2553	Dispositivo conectado
A0	**P1.0**	**Potenciómetro lineal**
A3	**P1.3**	**Joystick 3 ejes; Eje X**
A4	**P1.4**	**Joystick 3 ejes; Eje Y**
A5	**P1.5**	**Joystick 3 ejes; Eje Z**
A6	**P1.6**	**Joystick 2 ejes; Eje X**
A7	**P1.7**	**Joystick 2 ejes; Eje Y**

- El UART: Se utilizo el protocolo de comunicación serial UART para mandar todos los datos hacia una computadora.

UART	Pin MSP430G2553	Dispositivo conectado
TX	**P1.1**	**MAX232**
RX	**P1.2**	**MAX232**

En la Figura 33 se muestra el diagrama de flujo del programa cargado al MSP430G2553 y en el Anexo 1 se muestra el código desarrollado en IARD Workbench.

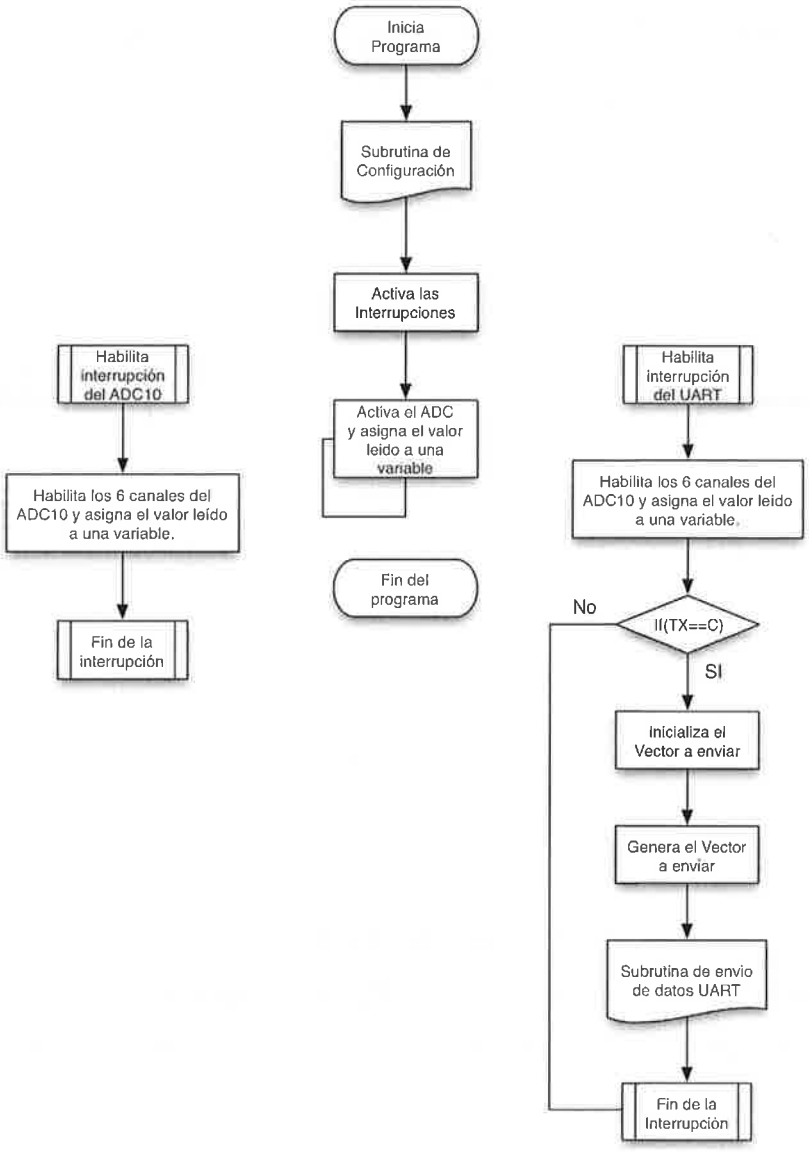


Figura 33 Diagrama de Flujo del MSP430 LaunchPad

Para poder adquirir las señales de los diferentes dispositivos se diseño una tarjeta electrónica, la cual cumple con la mayoría de las especificaciones de los Booster Pack, ya que se desea mejorarla para poder subir el diseño y compartirlo con la comunidad en la página de Internet de TI dentro de los BoosterPacks.

Para el diseño de la tarjeta electrónica se utilizo el software de Altium donde se realizo el diagrama electrónico o diagrama esquemático, y se diseño el circuito impreso (PCB). En la

Figura 34 se muestra el diagrama esquemático, el cual se diseño en base a las hojas de especificaciones de cada uno de los componentes que integran la tarjeta electrónica.

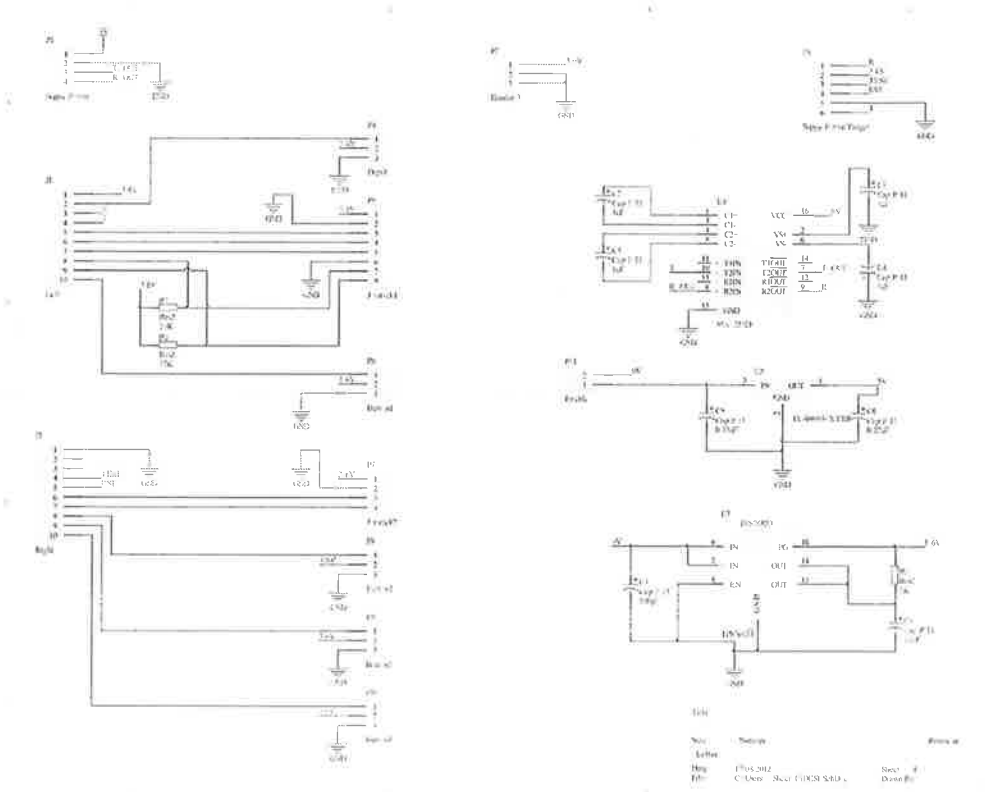


Figura 34 Diagrama electrónico

El diseño del circuito impreso se presenta en la Figura 35, dicho diseño se baso en las normas de diseño para Booster Pack de Texas Instruments ya que se pretende compartir con la comunidad ingenieril.

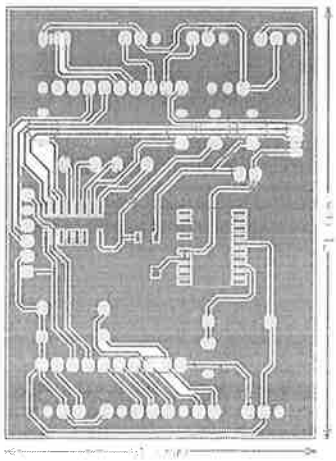


Figura 35 Circuito Impreso (PCB)

Bill f Materials <Parameter Title n tf und>

S urce Data Fr m: PCB_CIDESI.PrjPcb
Pr jecl: PCB_CIDESI.PrjPcb
Variant: N ne

Creati n Dale: 17/08/2012 10:18:53
Print Dale: 41138 41138.42982

F tprint	C mment	LibRef	Designat r	Descripti n	Quantity
CAPPR5-5x5	Cap P l1	Cap P l1	C1, C2, C3, C4, C5, C6, C8	P larized Capacit r (Radial)	7
C0805	Cap P l3	Cap P l3	C7	P larized Capacit r (Surface M unl)	1
HDR1 10H	Left	Header 10H	J1	Header, 10-Pin, Right Angle	1
HDR1 10H	Right	Header 10H	J2	Header, 10-Pin, Right Angle	1
HDR1 4	Suppy P wer	Header 4	P1	Header, 4-Pin	1
HDR1 3	Header 3	Header 3	P2	Header, 3-Pin	1
HDR1 6	Suppy P wer Targel	Header 6	P3	Header, 6-Pin	1
HDR1 3	Depth	Header 3	P4	Header, 3-Pin	1
HDR1 8	J ystick1	Header 8	P5	Header, 8-Pin	1
HDR1 3	Butt n1	Header 3	P6	Header, 3-Pin	1
HDR1 4	J ystick2	Header 4	P7	Header, 4-Pin	1
HDR1 3	Butt n2	Header 3	P8	Header, 3-Pin	1
HDR1 3	Butt n3	Header 3	P9	Header, 3-Pin	1
HDR1 3	Butt n4	Header 3	P10	Header, 3-Pin	1
MHDR1 2	Enable	MHDR1 2	P11	Header, 2-Pin	1
A IAL-0.4	Res2	Res2	R1, R2, R3	Resist r	3
D016_N	MA 232D	MA 232D	U1	Dual EIA-232 Driver/Receiver	1
PK03	TL780-05CKTTR	µA78L05CPKR	U2	P sitive-V ltage Regulal r	1
DW020_L	TPS76833	TPS76833	U3		1
					27

Appr ved N tes

Figura 36 Lista de Materiales.

El PCB se fabrico en la máquina de maquinado de PCBs, la cual se encuentra dentro del CIDESI en el laboratorio de Mecatrónica, en la Figura 37 se muestra la tarjeta maquina. En la Figura 38 y la Figura 39 se muestra la tarjeta electrónica terminada.

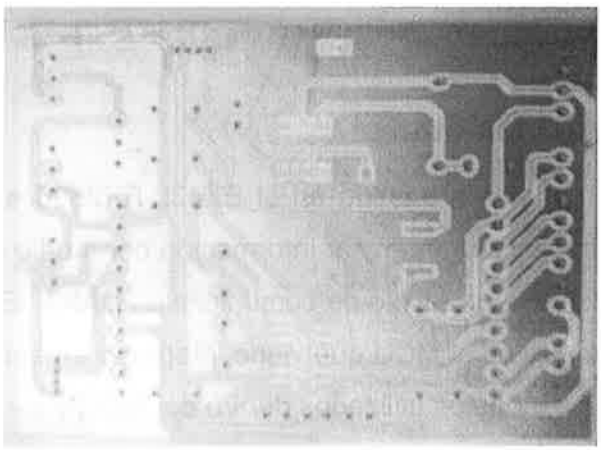


Figura 37 PCB fabricado.

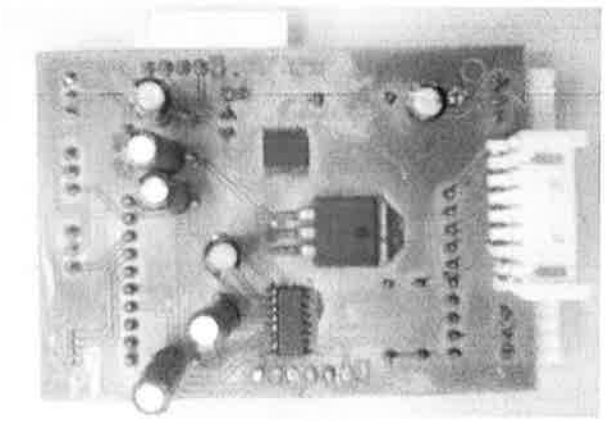


Figura 38 Tarjeta electrónica Vista superior.

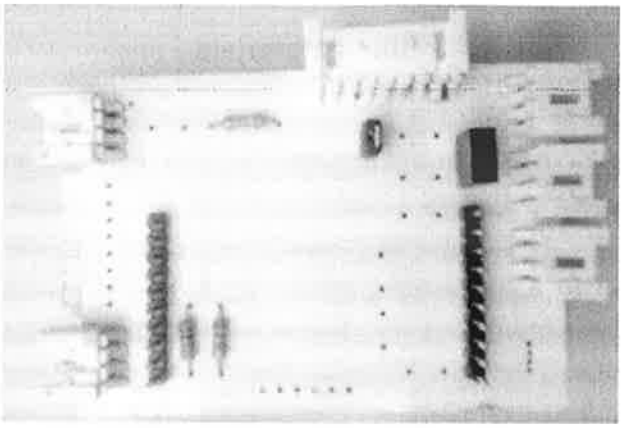


Figura 39 Tarjeta electrónica vista inferior.

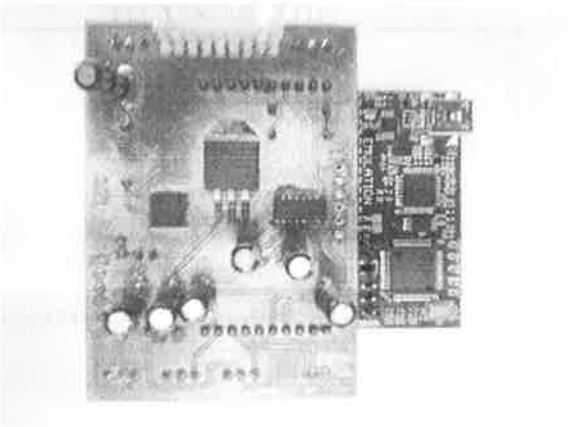


Figura 40 Tarjeta electrónica con el MSP430

5.3. Diseño del control Inalámbrico.

El panel de control inalámbrico esta basado en el EZ430-RF2500 este kit de desarrollo como ya se sabía tiene la capacidad de enviar información por medio de radiofrecuencia a otro MCU y este a su vez tiene el protocolo de comunicación UART. El MCU que se utilizo fue el M430F2274 debido a que es el MCU que viene integrado en este kit de desarrollo, a continuación se menciona los módulos utilizados dentro del MCU y como se conectaron.

Target Board. Acces Point (AP).

La tarjeta de evaluación del Acces Point(AP) se conecta directamente a la tarjeta debugger USB, no necesita de algún otro dispositivo, solo se necesita cargar el programa que se muestra en el Anexo 2. A continuación se muestra como se conecto.

- El UART: Se utilizo el protocolo de comunicación serial UART para mandar todos los datos hacia una computadora.

UART	Pin MSP430F2274	Dispositivo conectado
Tx	**P3.4**	**Debugger USB**
Rx	**P3.5**	**Debugger USB**

Target Board. End Point (EP).

Para la tarjeta de evaluación End Point(EP) utilizaron los siguientes módulos del MCU y se conectaron de la siguiente manera.

- Entradas/Salidas Digitales: Se utilizaron entradas digitales para poder conectar cada uno de los botones que tienen el panel de control. Se utilizo solo algunos pines del puerto 4 del microcontrolador para ello, debido a que la tarjeta de evaluación solo cuenta con eso pines disponibles, ya que tienes otros pines accesibles, pero son solo para monitorear la comunicación SPI.

E/S Digital	Pin MSP430F2274	Dispositivo conectado
Entrada Digital	**P4.4**	**Joystick 3 ejes; Botón 1**
Entrada Digital	**P4.5**	**Botón 1**
Entrada Digital	**P4.6**	**Joystick 3 ejes; Botón 2**

- El ADC10: Se utilizaron 6 canales del ADC10 para convertir la señal analógica a una digital. El orden en que se conectaron cada uno de ellos fue el siguiente:
-

Canal de ADC10	Pin MSP430F2274	Dispositivo conectado
A0	**P2.0**	**Joystick 3 ejes; Eje X**
A1	**P2.1**	**Potenciómetro lineal**
A2	**P2.2**	**Joystick 3 ejes; Eje Y**
A3	**P2.3**	**Joystick 2 ejes; Eje X**
A4	**P2.4**	**Joystick 3 ejes; Eje Z**
A12	**P4.3**	**Joystick 2 ejes; Eje Y**

- El UART: Se utilizo el protocolo de comunicación serial UART para mandar todos los datos hacia una computadora.

UART	Pin MSP430F2274	Dispositivo conectado
Tx	**P3.4**	**Debugger USB**
Rx	**P3.5**	**Debugger USB**

En la Figura 41 se muestra el diagrama de flujo del programa cargado al MSP430F2274 para el Acces Point y en el Anexo 2 se muestra el código desarrollado en IARD Workbench. En la Figura 42 se muestra el diagrama de flujo del End Point y en el Anexo 3 se muestra el código que se le cargo.

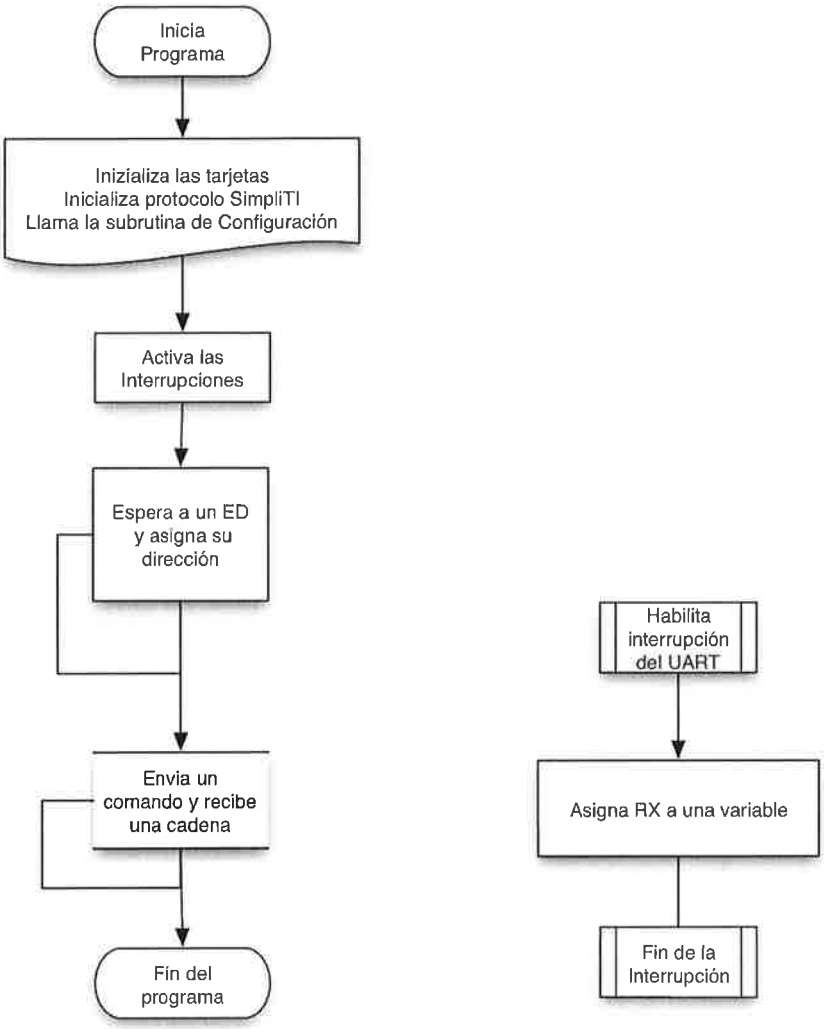


Figura 41 Diagrama de Flujo del MSP430F2274 EZ430-RF2500 (Acces Point)

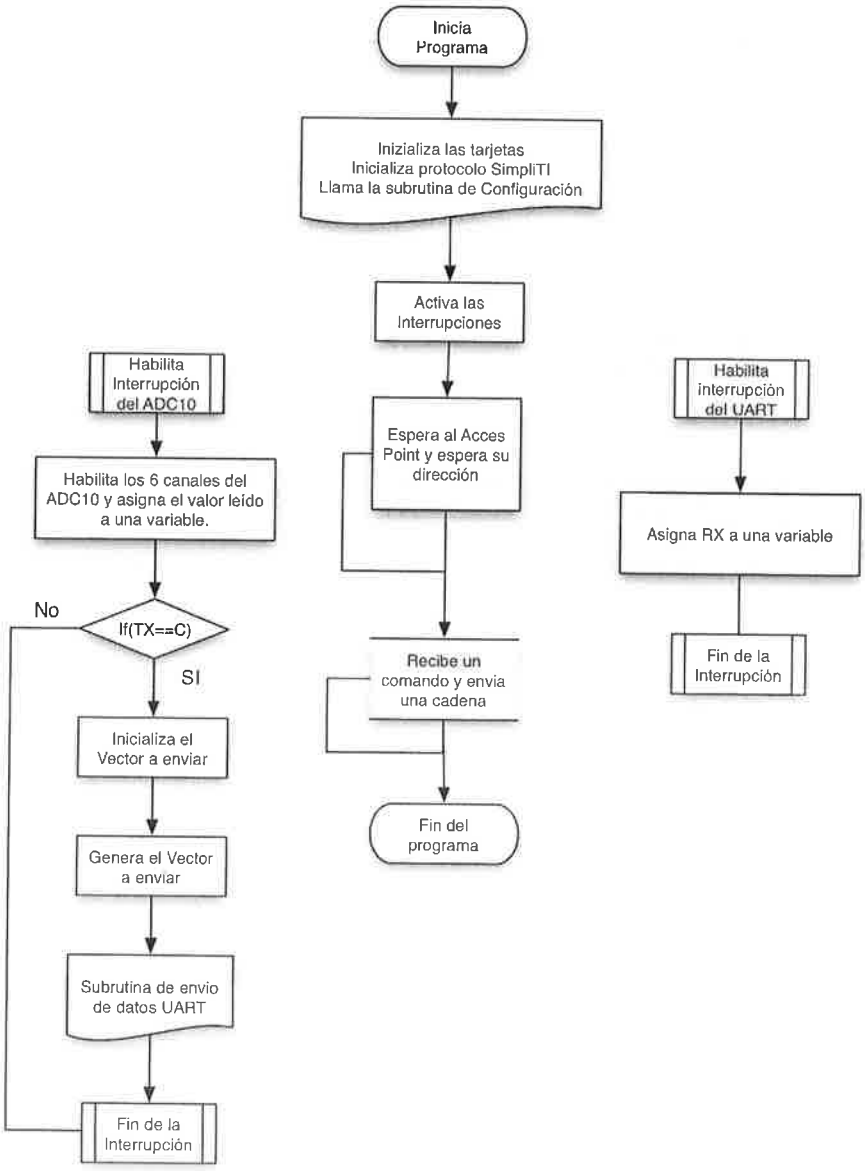


Figura 42 Diagrama de Flujo del MSP430F2274 EZ430-RF2500 (End Point)

Para poder adquirir toda la información que proviene de cada uno de los componentes que integran el panel de control se diseño una tarjeta electrónica, para ello se utilizo el software de Altium donde se realizo el diagrama electrónico o diagrama esquemático, y se diseño el circuito impreso (PCB). En la Figura 43 se muestra el diagrama esquemático, el cual se diseño en base a las hojas de especificaciones de cada uno de los componentes que integran la tarjeta electrónica.

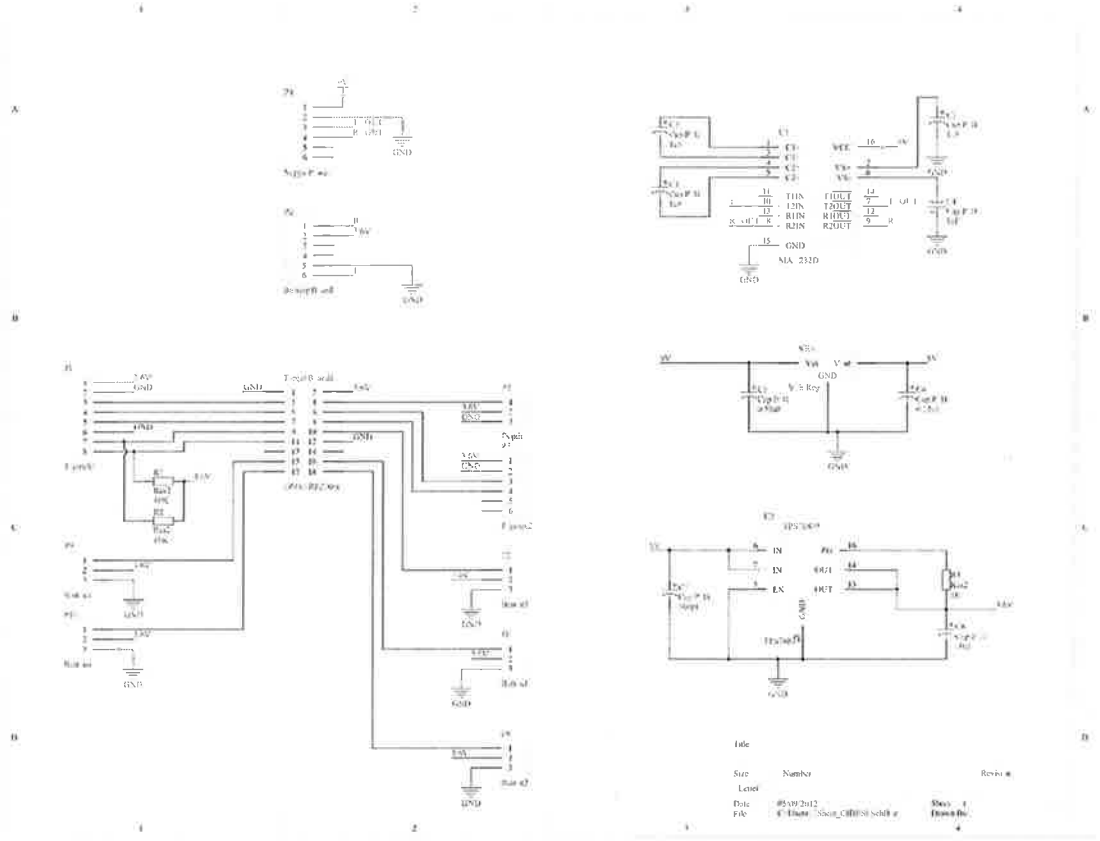


Figura 43 Diagrama electrónico

El diseño del circuito impreso se presenta en la Figura 44, dicho diseño se trato de hacer lo mas compacto posible y se dejo unos conectores de más por si en un futuro se llegaran a utilizar y en la Figura 45 se muestra la lista de los materiales utilizados.

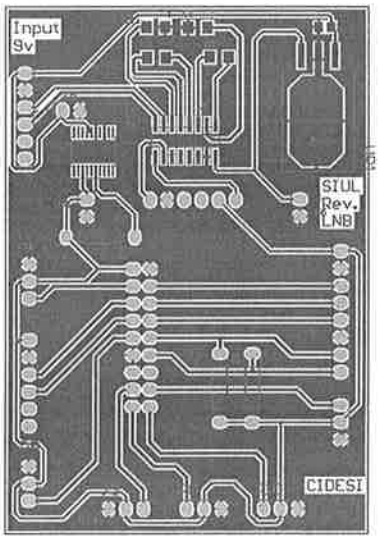


Figura 44 Circuito Impreso (PCB)

Bill f Materials <Parameter Title n t f und>

S urce Data Fr m: PCB_CIDESI.PrjPcb
Pr ject: PCB_CIDESI.PrjPcb
Variant: N ne

Creati n Date: 05/09/2012 12:20:44 p.m.
Print Date: 41157 41157.51448

F tprint	C mment	LibRef	Designat r	Descripti n	Quantity
C1206	Cap P l1	Cap P l1	C1, C2, C3, C4	P larized Capacit r (Radial)	4
HDR1 2	Cap P l1	Cap P l1	C5, C6, C8	P larized Capacit r (Radial)	3
C0805	Cap P l3	Cap P l3	C7	P larized Capacit r (Surface M unt)	1
HDR1 8	J ystick1	Header 8	J1	Header, 8-Pin	1
HDR1 3	Butt n1	Header 3	J3	Header, 3-Pin	1
HDR1 6	Suppy P wer	Header 6	P1	Header, 6-Pin	1
HDR1 6	Battery B ard	Header 6	P2	Header, 6-Pin	1
HDR1 3	Depth	Header 3	P3	Header, 3-Pin	1
HDR1 3	Butt n2	Header 3	P5, P8	Header, 3-Pin	2
HDR1 6	J ystick2	Header 6	P7	Header, 6-Pin	1
HDR1 3	Butt n3	Header 3	P9	Header, 3-Pin	1
HDR1 3	Butt n4	Header 3	P10	Header, 3-Pin	1
A IAL-0.4	Res2	Res2	R1, R2, R3	Resist r	3
HDR2 9	eZ430-RF2500t	Header 9 2	Target B ard1	Header, 9-Pin, Dual r w	1
D016_N	MA 232D	MA 232D	U1	Dual EIA-232 Driver/Receiver	1
TPS76850	TPS76833	TPS76833	U2		1
TL78005	V lt Reg	V lt Reg	VR1	V ltage Regulat r	1
					25

Appr ved N tes

Figura 45 Lista de Materiales.

El PCB se fabrico en la máquina de maquinado de PCBs, la cual se encuentra dentro del CIDESI en el laboratorio de Mecatrónica, en la Figura 46 se muestra la tarjeta maquina. En la Figura 48 y la Figura 49 se muestra la tarjeta electrónica terminada.

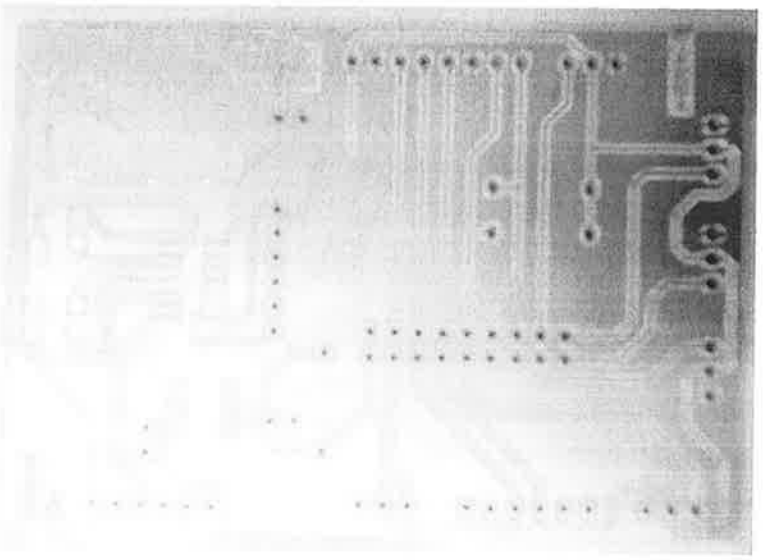


Figura 46 PCB fabricado.

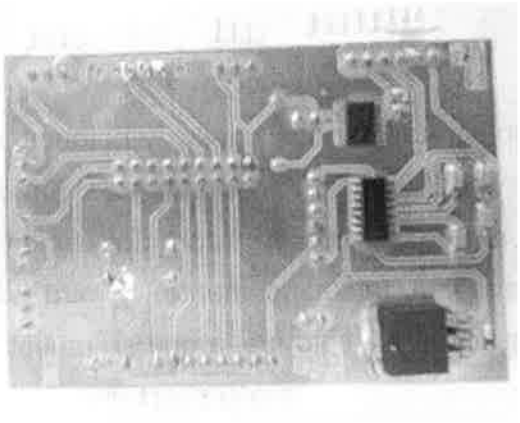


Figura 47 Tarjeta electrónica Vista superior.

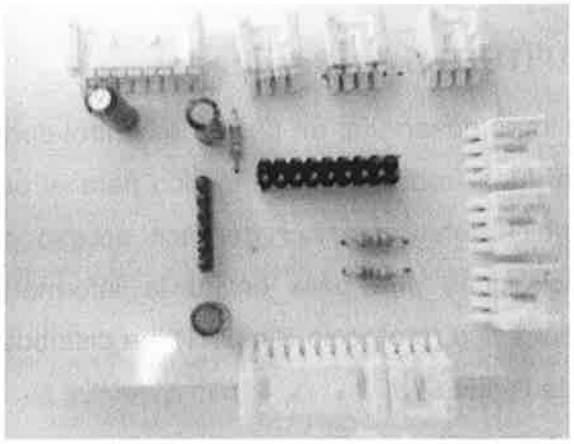


Figura 48 Tarjeta electrónica vista inferior.

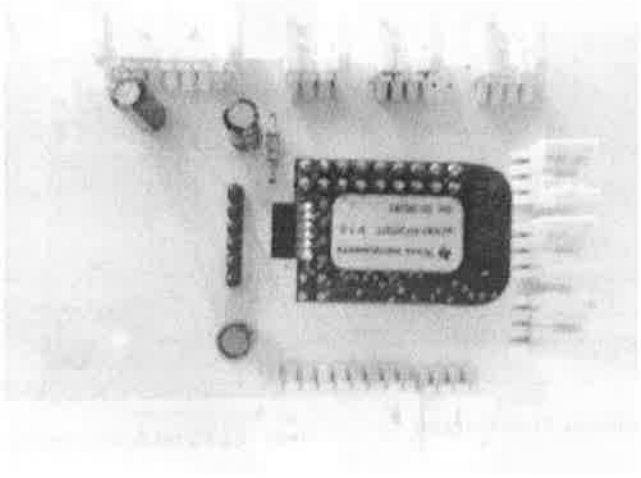


Figura 49 Tarjeta electrónica con el MSP430

CAPÍTULO 6. Resultados.

Se logro desarrollar un panel de control donde se integraran los dos joystick y botones, el cual fuera robusto y ergonómico para el operador ver Figura 50, dicho panel de control también tenia que ser de fácil acceso debido a que se desarrollaron dos tarjetas electrónicas una para enviar la información vía serial (alambricamente) y otra vía inalámbrica (protocolo SimpliTI). La distribución por la cual se opto fue la que se muestra en la Figura 51.

Figura 50 Panel de Control Ensamblado. Vista Exterior

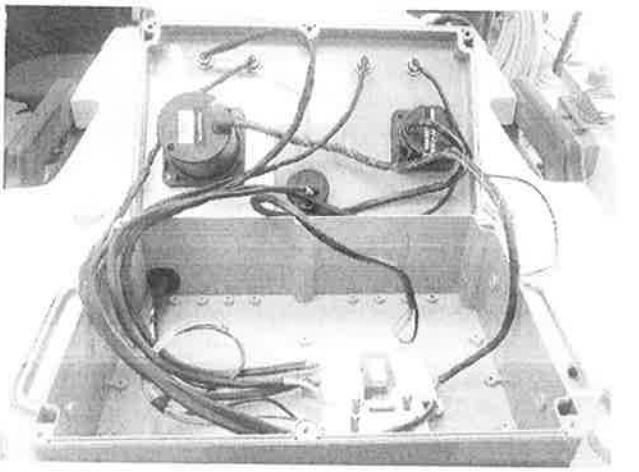


Figura 51 Panel de Control Ensamblado. Vista Interior

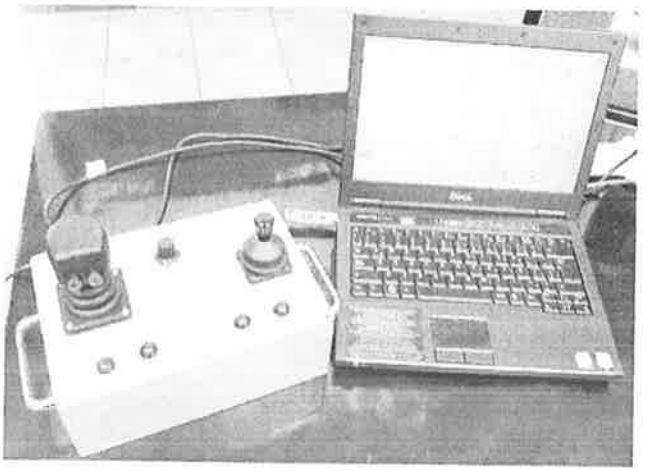


Figura 52 Panel de Control y PC

Figura 53 PC con Debugger USB

En la Figura 52 y la Figura 53 se muestra el panel de control que se desarrollo y la computadora que se utiliza para el control del robot submarino, la cual tiene cargada la aplicación que se diseño en LabView para la visualización de los movimientos que se realizan en el panel de control, dicha aplicación se muestra en el Anexo 4. Cabe señalar que es la misma aplicación para el control alambrico que para el inalámbrico, solo se cambia el puerto de comunicación. Para el control alambrico durante las pruebas se obtuvo una comunicación constante y fiel a los movimiento en el panel de control, casi en tiempo real, pero para el control inalámbrico se obtuvo una comunicación algo inestable debido a que presentaba fallas ya que de vez en cuando se perdía la comunicación, es por ello que no se obtenían movimientos fieles al panel de control. En la siguiente tabla se muestran cada uno de los dispositivos utilizados y la función que realiza.

Función a Realizar	Dispositivo conectado	Salida
Propulsor de Profundidad (Subiendo/Bajando)	Potenciómetro lineal	100% a -100% 5Volts a -5Volts
Propulsores de Movimiento (Adelante/Atrás)	Joystick 3 ejes; Eje X	100% a -100% 5Volts a -5Volts
Propulsor de Movimiento (Izquierda/Derecha)	Joystick 3 ejes; Eje Y	100% a -100% 5Volts a -5Volts
Propulsores de Rotación (Izquierda/Derecha)	Joystick 3 ejes; Eje Z	100% a -100% 5Volts a -5Volts
Brazo (Arriba/Abajo)	Joystick 2 ejes; Eje X	Digital 0 – 1
Brazo (Izquierda/Derecha)	Joystick 2 ejes; Eje Y	Digital 0 – 1
Abrir Pinza	Joystick 3 ejes; Botón Derecha	Digital 0 – 1
Cerrar Pinza	Joystick 3 ejes; Botón Izquierda	Digital 0 – 1
Lámparas (-)	Botón 1	Digital 0 – 1
Lámparas (+)	Botón 2	Digital 0 – 1
No utilizado	Botón 3	No utilizado
No utilizado	Botón 4	No utilizado

En la Figura 54 y la Figura 55 se muestran dos capturas de pantallas de cuando se estaban realizando pruebas con el control. En la primera figura se puede apreciar que se presiona un botón y se activa el indicador en la interfaz grafica y en la segunda se activa el joystick de los propulsores y en la interfaz grafica se muestra como se desplaza hacia atrás los dos propulsores

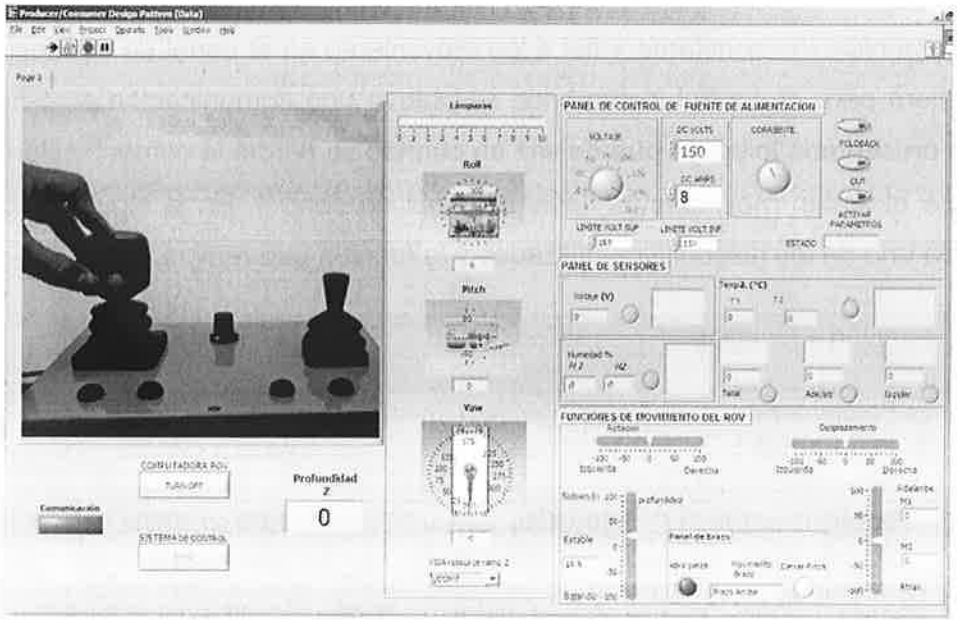


Figura 54 Prueba 1.

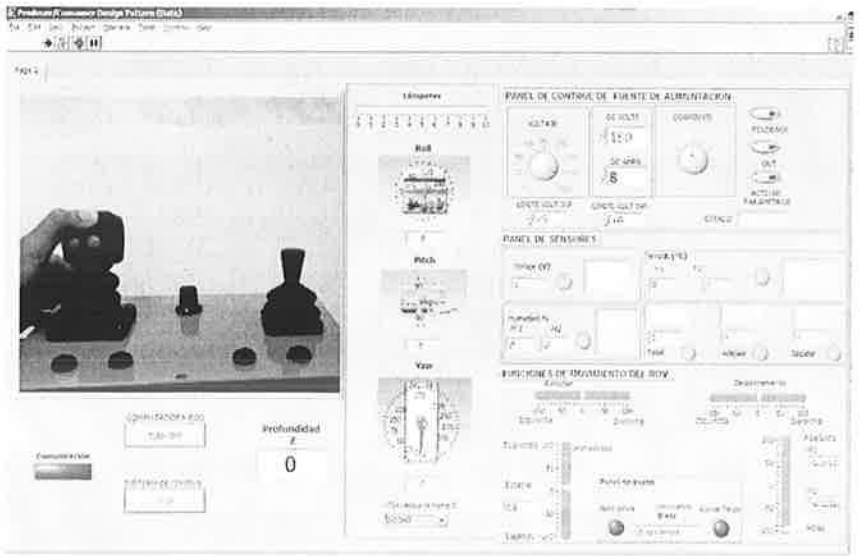


Figura 55 Prueba 2.

CAPÍTULO 7. Conclusiones.

Los objetivos que se propusieron al iniciar el trabajo se cumplieron en su mayoría, ya que se logro desarrollar un panel de control para la tele-operación del robot submarino KAXAN, en el cual se hizo una integración de las palancas y botones.

Se diseñaron dos tarjetas electrónicas para adquirir las señales analógicas y digitales, debido a que se programaron dos microcontroladores (MSP Texas Instruments) para adquirir las señales, procesarlas y enviarlas. Una de las tarjetas electrónicas y un kit de desarrollo de Texas Instruments se utilizo para hacer un panel de control alambrico, ya que la información que recolectaba el MSP la enviar vía serial (protocolo RS-232) y la otra tarjeta electrónica con el otro kit de desarrollo envía la información inalambricamente por medio del protocolo de Texas Instruments SimpliTI.

El primer control con el Kit MSP430 LaunchPad que fue el alambrico en vía los datos adquiridos a una computadora por el puerto serial, la computadora tiene cargada una interfaz grafica en LabView, la cual se diseño para desplegar la información de las palancas y botones que integran el panel de control, las cuales son adaptadas a los movimientos del robot submarino, en dicha aplicación se obtuvo una comunicación transparente y fiel a los movimientos del panel de control.

En el segundo control con el Kit de desarrollo de Texas Instruments el eZ430-RF2500 con su respectiva tarjeta electrónica, se enviaron los datos adquiridos a una computadora de manera inalámbrica hacia un puerto serial de la computadora. En la computadora se le cargo una aplicación en LabView la cual es la misma que la del control alambrico, pero en ella se presentaron algunas fallas, ya que no se obtuvo una comunicación 100% transparente, debido a que se presentaba un pequeño retardo en los movimientos del panel de control con los visualizados en LabView y había veces que se perdía la comunicación.

CAPÍTULO 8. Trabajo a futuro.

Las posibilidades de este tipo de tecnologías son muy extensas y variadas.

En primer lugar centrándonos en el control alámbrico se podría depurar aun más el programa para aprovechar el desempeño del MSP430, ya que el MSP430G2553 pertenece a la familia de los microcontroladores de ultra baja potencia y se podría disminuir el consumo significativamente. Por otro lado se podría crear un Booster Pack aparir de la tarjeta diseñada, debido que actualmente en la pagina de Texas Instruments no se encuentra ningún diseño de algún Booster Pack que adquiera señales analógicas de los 8 canales ADC, las 6 entradas digitales del puerto 2 y que a su vez envíe la información adquirida de forma serial hacia la computadora, por medio del UART. Es por ello que podría ser una área de oportunidad, para comercializar un Booster Pack o simplemente para compartir el conocimiento adquirido con la comunidad ingenieril.

En el segundo diseño del control que fue el control inalámbrico basado en el Kit de evaluación, EZ430-RF2500 se podría igualmente depurar el código para aprovechar las características de ultra baja potencia del MSP430F2274. Adicionalmente este control necesitara del diseño de un sistema de administración de energía debido a que es inalámbrico, y se podría mejorar el diseño de la tarjeta electrónica añadiendo dicho sistema. El sistema de administración de energía deberá contar con un control de carga rápida para unas baterías recargables que llevara el control y también deberá contar con una función que le permita operar el equipo mientras se encuentra cargando las baterías.

Por otro lado sería buena idea probar con otro tipo de tecnología inalámbrica, como la tecnología Bluetooth, debido a que el tipo de tecnología utilizado durante el desarrollo del proyecto SimpliTI, tuvo algunos problemas de comunicación, los cuales se podrían resolver por medio de programación, pero en cuanto a seguridad deja mucho que desear debido a que es un protocolo con muy baja seguridad o nula a comparación de la tecnología Bluetooth.

Por ultimo se recomienda ampliamente añadirle al diseño del control unas guías de restricción de movimiento para los joystick, debido a que existen ciertos movimientos restringidos por la seguridad del ROV. Para ello es necesario diseñar unas guías mecánicas que se tendrían que instalar a cada uno de los joystick, o en su defecto reemplazar los joystick, por alguno otro que

si cuente con guías para evitar movimientos indeseados. Adicionalmente seria muy recomendable cambiar el potenciómetro lineal de precisión, el cual sirve para controlar la profundidad del ROV, por algún potenciómetro digital de no contacto, debido a que el potenciómetro convencional se llega a dañar más fácilmente que uno de no contact.ANEXOS.

Anexo 1 Código C IAR Workbench para el MSP430G24 del LaunchPad.

C:\IAR Proyect\Proyecto CIDESI - Copy\ADC_Serial_P16.c

```
//******************************************************************************
//  MSP430G2553 - USCI_A0, 19200 UART Echo ISR, DCO SMCLK, XIN=  A1,A4,A5,A6,A7
//
//  Description: Echo a received character, RX ISR used. Normal mode is LPM0.
//  USCI_A0 RX interrupt triggers TX Echo.
//  Baud rate divider with 1MHz = 1MHz/19200 =  52.1
//  ACLK = n/a, MCLK = SMCLK = CALxxx_1MHZ = 1MHz
//
//                    MSP430G2553
//                  -----------------
//              /|\|                 XIN|-
//               | |                    |  32KHZ
//               --|RST             XOUT|-
//                 |                    |
//                 |      P1.2/UCA0TXD|------------>
//                 |                    | 19200 - 8N1
//                 |      P1.1/UCA0RXD|<------------
//                 |                    |
//             --->|P1.0/A0        P1.5/A5|<-----
//             --->|P1.3/A3        P1.6/A6|<-----
//             --->|P1.4/A4        P1.7/A7|<-----
//                 |                    |
//             --->|P2.0                |
//             --->|P2.3                |
//             --->|P2.5                |
//                 |                    |
//                  --------------------
//
//  D. Dang
//  Texas Instruments Inc.
//  May 2011
//  Built with IAR Embedded Workbench Version: 5.10.6
//******************************************************************************

#include  "msp430g2553.h"

unsigned int  ADC7, ADC6, ADC5, ADC4, ADC3, ADC2, ADC1, ADC0;
unsigned char MSB, LSB, P2;

void adc();
void configura();
void TXString();

//------------------------------------------------------------------------------
//                            Main
//------------------------------------------------------------------------------
void main(void)
{
  configura();
  __bis_SR_register(GIE);
  while( )
  {
  ADC10CTL0 &= ~ENC;
  while (ADC10CTL1 & BUSY);
  ADC10SA =     ';
  __delay_cycles(   );
  ADC10CTL0 |= ENC + ADC10SC;
  }
```

```
}

/*-------------------------------------------------------------------------
                    USCIA interrupt service routine
-------------------------------------------------------------------------*/
#pragma vector=USCIAB0RX_VECTOR          // Echo back RXed character, confirm TX buffer is ready first
__interrupt void USCI0RX_ISR(void)
{
  while (!(IFG2 & UCA0TXIFG));               // USCI_A0 TX buffer ready?
  {
    ADC10CTL0 &= ~ENC;                      // ADC10 Enable Conversion
    while (ADC10CTL1 & BUSY);               // Wait if ADC10 core is active
    ADC10SA = 0x200;                        // ADC10 Data Transfer Start Address
    __delay_cycles(100);                    // Wait 100 clock cycles
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
    if(UCA0RXBUF==0x42)
    {
       char output[] = {"1XX2XX3XX4XX5XX6XX7XxZ"}; // String to send

       output[1] = ADC0&0xFF;                       // Send LSB (Least Significant Bit)
       output[2] = (ADC0>>8)&0xFF;                  // Send MSB (Most Significant Bit)
       output[4] = ADC3&0xFF;;
       output[5] = (ADC3>>8)&0xFF;
       output[7] = ADC4&0xFF;;
       output[8] = (ADC4>>8)&0xFF;
       output[10] = ADC5&0xFF;
       output[11] = (ADC5>>8)&0xFF;
       output[13] = ADC6&0xFF;
       output[14] = (ADC6>>8)&0xFF;
       output[16] = ADC7&0xFF;
       output[17] = (ADC7>>8)&0xFF;
       output[19] = P2IN;                           // Send digital port
       //output[10] = 0x32;                           // Send finish

       TXString(output, 20 );          // Send String
       //__delay_cycles(3000);                   // Wait 100 clock cycles
      }
  }
}


/*-------------------------------------------------------------------------
                    ADC10 interrupt service routine
-------------------------------------------------------------------------*/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
    while (!(1));
    {
  ADC10CTL0 &= ~ENC;                        // ADC10 Enable Conversion
  while (ADC10CTL1 & BUSY);                 // Wait if ADC10 core is active
  ADC10SA = 0x200;                          // ADC10 Data Transfer Start Address
  __delay_cycles(100);                      // Wait 100 clock cycles
  ADC10CTL0 |= ENC + ADC10SC;               // Sampling and conversion start
    }
}

/*-------------------------------------------------------------------------
```

```
                    Configura routine
-------------------------------------------------------------------------*/
void configura()
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

  BCSCTL1 = CALBC1_1MHZ;                    //configuracion de reloj
  DCOCTL = CALDCO_1MHZ;                     // Set DCO
                                            //configuracion rs232 rev. pag 46
  P1SEL = BIT1 + BIT2 ;                     // P1.1 = RXD, P1.2=TXD
  P1SEL2 = BIT1 + BIT2 ;                    // P1.1 = RXD, P1.2=TXD
                                            //revisar pagina 44)
  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
                                            // Compensacion de error rev.
                                            // pag. 435  tabla
  UCA0BR0 = 104;                            // 1MHz 19200
  UCA0BR1 = 0;                              // 1MHz 19200
  UCA0MCTL = UCBRS1;                        // Modulation UCBRSx = 1

  ADC10CTL1 = INCH_7 + CONSEQ_1;            // A3/A2/A1, single sequence
  ADC10CTL0 = ADC10SHT_2 + MSC + ADC10ON + ADC10IE;
  ADC10DTC1 = 0x08;                         // Num of conversions
//SE SELECIONAN 5 CONVERSIONES Y LA ULTIMA CONVERSIÓN SE GUARDA EN EL ADC10MEM
                                            //inicializa
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt

  P2DIR= 0xFF;                              // All Digital Input Port 2

  ADC0 = 0;                                 // Initial Value
  ADC1 = 0;
  ADC2 = 0;
  ADC3 = 0;
  ADC4 = 0;
  ADC5 = 0;
  ADC6 = 0;
  ADC7 = 0;
}

/*-------------------------------------------------------------------------
                    TXString routine
-------------------------------------------------------------------------*/
void TXString( char* string, int length )           // Send string
{
  int pointer;
  for( pointer = 0; pointer < length; pointer++)
  {
    //Volatile int i;
    __delay_cycles(1000);
    UCA0TXBUF = string[pointer];                    // Send character
    __delay_cycles(1000);
    while (!(IFG2&UCA0TXIFG));                      // USCI_A0 TX buffer ready?
  }
  pointer = 0;
}
```

Anexo 2 Código C IAR Workbench del eZ430-RF2500T. Acces Point.

C:\Users\Ricky\Desktop\eZ430-RF2500 Wireless\demo_AP3.c

```
/*******************************************************************************************
Chat Program -- Link Listen
*******************************************************************************************/
#include "bsp.h"
#include "mrfi.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"

char Buffer[20] = {0};
char Key =0;
char Key_Not_Pressed, TxHold = 1;

static void linkFrom(void);
static linkID_t sLinkID2;
static volatile uint8_t  sSemaphore;
volatile char Temp_Size, Temp_General=0;

static uint8_t sRxCallback(linkID_t);
void DisplayCharacter(char[], char);
void tx_char(uint8_t *, int*);
void Configura(void);
void Get_Name (void);
#define DELAY  NWK_DELAY(100)

/*-------------------------------------------------------------------------
                              Main
-------------------------------------------------------------------------*/
void main (void)
{
  BSP_Init();                                   // SimpliciTI Initialization
  SMPL_Init(sRxCallback);
  Configura();
  BSP_TURN_ON_LED1();
  linkFrom();                                   // Go to main process
  while (1);
}

static void linkFrom()
{
  uint8_t  msg[2]={0};
  int count = 0;
  while (1)                                     // Listen for link forever...
  {
    if (SMPL_SUCCESS == SMPL_LinkListen(&sLinkID2))
    {
      break;
    }
  }
  BSP_TURN_OFF_LED1();
  BSP_TURN_ON_LED2();
  Configura();

  int * countptr;
  countptr = &count;
  uint8_t* char_var;
```

```
  char_var = &msg[0];
  uint8_t  rxmsg[2], tid = 0;

  while(1)                                          // Main Loop
  {
    tx_char(char_var, countptr);                    // Transmit function
    if (sSemaphore)                                 // Poll for Received Msgs
     {
       *(rxmsg+1) = ++tid;
       SMPL_Send(sLinkID2, rxmsg, 2);
       sSemaphore = 0;
     }
  }
}

/*-------------------------------------------------------------------------
                          tx_char routine
-------------------------------------------------------------------------*/
void tx_char(uint8_t  *msg_ptr, int *count)
{

  while(Key_Not_Pressed);                   // Wait for a key press from the user
  if( *count >1)
    TxHold = 0;                             // Overflow Condition

  Buffer[*count] = Key;                     // Queue the character
  (*count)++;

  if (!TxHold)
  {                                         // At this point we're ready to tx
    Buffer[*count] = 0x 0;
    for(int i=0; i< *count; i++)            // Send your message
    {
      msg_ptr[0] = Buffer[i];
      //TOGGLE_LED1();
      SMPL_Send(sLinkID2, msg_ptr, 2);
      __delay_cycles(3000);                         // Wait 3000 clock cycles
    }
    TxHold=0;
    *count=0;
  }
  Key_Not_Pressed = 1;
}


/*-------------------------------------------------------------------------
                          sRxCallback routine
-------------------------------------------------------------------------*/
static uint8_t sRxCallback(linkID_t port)       // handle received messages
{
  uint8_t msg[MAX_APP_PAYLOAD], len;

  if (port == sLinkID2)
  {
        if (SMPL_Receive(sLinkID2, msg, &len) == SMPL_SUCCESS)
    {
      DisplayCharacter(msg, 0);
      __delay_cycles(  30);                         // Wait 3000 clock cycles
```

```
      BSP_TOGGLE_LED1();
    }
  }
  return 0;
}


/*-------------------------------------------------------------------------
                        DisplayCharacter routine
-------------------------------------------------------------------------*/
void DisplayCharacter(char array[], char size)
{
  _DINT();                                  // This function enables the UART
  P3SEL |= 0x30;                            // P3.4,5 = USCI_A0 TXD/RXD
  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 = 0x41;                           // 8MHz 9600
  UCA0BR1 = 0x03;                           // 8MHz 9600
  UCA0MCTL = UCBRS1;                        // Modulation UCBRSx = 2
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  for(int count=0; count<size; count++){
    while (!(IFG2&UCA0TXIFG));              // USCI_A0 TX buffer ready?
    UCA0TXBUF = array[count];               // TX -> RXed character
    _EINT();
  }
}

/*-------------------------------------------------------------------------
                        USCIA interrupt service routine
-------------------------------------------------------------------------*/
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
  Key_Not_Pressed = 0;
  Key = UCA0RXBUF;
  TxHold = 0;
}

/*-------------------------------------------------------------------------
                        Configura routine
-------------------------------------------------------------------------*/
void Configura(void)                        // Set up USCI channel A for back-channel UART
{
  UCA0CTL1 |= UCSWRST;                      // Reset State Machine
  UCA0CTL1 &= ~UCSWRST;                     // Init USCI Ch. A for key Press
  P3SEL |= 0x30;                            // P3.4,5 = USCI_A0 TXD/RXD
  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 = 0x41;                           // 8MHz / 9600
  UCA0BR1 = 0x03;                           // 8MHz / 9600 = 1041.6
  UCA0MCTL = UCBRS1;                        // Modulation UCBRSx = 1
  UCA0CTL1 &= ~UCSWRST;                     // Initialize USCI state machine
  IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt
  Key_Not_Pressed = 1;
}
```

```
  while((Key_Not_Pressed));                 //Wait for a key press from the user
  TxHold = 0;

  Buffer[*count] = Key;                     // Assigned Key to Buffer
  (*count)++;

  if (!TxHold)                              // Display Buffer
  {
    Buffer[*count] = 0x02;
    for(int i=0; i< *count; i++)
    {
      msg_ptr[0] = Buffer[i];
      BSP_TOGGLE_LED1();
      SMPL_Send(sLinkID1, msg_ptr, 1);      //Sends data to a peer.
    }
    TxHold=1;
    *count=0;                               // Clear Buffer
  }
  Key_Not_Pressed = 1;                      // Reset Key flag
}

/*-------------------------------------------------------------------------+
                        sRxCallback routine
+-------------------------------------------------------------------------*/
static uint8_t sRxCallback(linkID_t port)          /* handle received frames. */
{
  uint8_t msg[ ], len;

  if (port == sLinkID1)
  {
    if ((SMPL_SUCCESS == SMPL_Receive(sLinkID1, msg, &len)) && len)
    {                                                   //Receive data to a peer
      DisplayCharacter(msg, 1);                         // Send data UART
      if(msg[0] ==0x01)                                 // Selector Keys
        {
          select='A';
        }
      else if(msg[ ] ==0x )
        {
         select='B';
        }
      else if(msg[ ] == 0x )
        {
          select='C';
        }
      else
        {
          select='D';
        }
      ADC10CTL0 |= ENC + ADC10SC;                 // Sampling and conversion start
    }
  }
  return 0;
}

/*-------------------------------------------------------------------------+
                        DisplayCharacter routine
+-------------------------------------------------------------------------*/
```

```
void DisplayCharacter(char array[], char size)
{
  _DINT();                                          // Disable Interrupts
  P3SEL |= 0x30;                                    // P3.4,5 = USCI_A0 TXD/RXD
  UCA0CTL1 |= UCSSEL_2;                             // SMCLK
  UCA0BR0 = 0x41;                                   // 8MHz 9600
  UCA0BR1 = 0x03;                                   // 8MHz 9600
  UCA0MCTL = UCBRS1;                                // Modulation UCBRSx = 2
  UCA0CTL1 &= ~UCSWRST;                             // Initialize USCI state machine
  for(int count=0; count<size; count++){
    while (!(IFG2&UCA0TXIFG));                      // USCI_A0 TX buffer ready?
    UCA0TXBUF = array[count];                       // TX -> RXed character
    _EINT();                                        // Enable Interrupts
  }
}

/*-------------------------------------------------------------------------
                        Configura routine
-------------------------------------------------------------------------*/
void Configura(void)
{
  UCA0CTL1 |= UCSWRST;                          // Reset State Machine
  UCA0CTL1 &= ~UCSWRST;                         // Initialize USCI Ch. A for key P
  P3SEL |= 0x30;                                // P3.4,5 = USCI_A0 TXD/RXD
  UCA0CTL1 |= UCSSEL_2;                         // SMCLK
  UCA0BR0 = 0x41;                               // 8MHz / 9600
  UCA0BR1 = 0x03;                               // 8MHz / 9600 = 1041.6
  UCA0MCTL = UCBRS1;                            // Modulation UCBRSx = 1
  UCA0CTL1 &= ~UCSWRST;                         // **Initialize USCI state machine**
  IE2 |= UCA0RXIE;                              // Enable USCI_A0 RX interrupt

  ADC10CTL0 = SREF_0 + ADC10SHT_2 + MSC + ADC10ON + ADC10IE; // referencia VCC para Vref+ y VSS para Vref- (sin ca
  ADC10AE0 = 0x01;
  ADC10AE1 = 0x10;

  P3DIR |= 0x00;                                // Set digital input port 2
  P4DIR |= 0x00;
  P4SEL |= 0x00;

  ADC0 = 0;
  ADC1 = 0;
  ADC2 = 0;
  ADC3 = 0;
  ADC4 = 0;
  ADC5 = 0;
  ADC6 = 0;
  ADC7 = 0;
}

/*-------------------------------------------------------------------------
                    USCIA interrupt service routine
-------------------------------------------------------------------------*/
#pragma vector=USCIAB0RX_VECTOR                 //  Echo back RXed character, confirm TX buffer is ready first
__interrupt void USCI0RX_ISR(void)
{
  Key_Not_Pressed = 0;                          // Key pressed
  Key = UCA0RXBUF;                              // Asigned RX in Key
  TxHold = 1;
```

```
}

/*-------------------------------------------------------------------------
                    ADC10 interrupt service routine
-------------------------------------------------------------------------*/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
      ADC10CTL1 = INCH_0;                       // Active analog input A0
      ADC10CTL0 |= ENC + ADC10SC;               // Sampling and conversion start
      while (ADC10CTL1 & BUSY);                 // Wait if ADC10 core is active
      __delay_cycles(100);                      // Wait 100 clock cycles
      ADC10CTL0 &= ~ENC;                        // Turn off ADC10
      ADC0 = ADC10MEM;                          // ADC 10 reading assigned

      ADC10CTL1 = INCH_1;                       // Active analog input A1
      ADC10CTL0 |= ENC + ADC10SC;               // Sampling and conversion start
      while (ADC10CTL1 & BUSY);                 // Wait if ADC10 core is active
      __delay_cycles(100);
      ADC10CTL0 &= ~ENC;
      ADC1 = ADC10MEM;

      ADC10CTL1 = INCH_2;                       // Active analog input A2
      ADC10CTL0 |= ENC + ADC10SC;               // Sampling and conversion start
      while (ADC10CTL1 & BUSY);                 // Wait if ADC10 core is active
      __delay_cycles(100);
      ADC10CTL0 &= ~ENC;
      ADC2 = ADC10MEM;

      ADC10CTL1 = INCH_3;                       // Active analog input A3
      ADC10CTL0 |= ENC + ADC10SC;               // Sampling and conversion start
      while (ADC10CTL1 & BUSY);                 // Wait if ADC10 core is active
      __delay_cycles(100);
      ADC10CTL0 &= ~ENC;
      ADC3 = ADC10MEM;

      ADC10CTL1 = INCH_4;                       // Active analog input A4
      ADC10CTL0 |= ENC + ADC10SC;               // Sampling and conversion start
      while (ADC10CTL1 & BUSY);                 // Wait if ADC10 core is active
      __delay_cycles(100);
      ADC10CTL0 &= ~ENC;
      ADC4 = ADC10MEM;

      ADC10CTL1 = INCH_12;                      // Active analog input A12
      ADC10CTL0 |= ENC + ADC10SC;               // Sampling and conversion start
      while (ADC10CTL1 & BUSY);                 // Wait if ADC10 core is active
      __delay_cycles(100);
      ADC10CTL0 &= ~ENC;
      ADC5 = ADC10MEM;

      ADC10CTL1 = INCH_0;                       // Active analog input A0
      ADC10CTL0 |= ENC + ADC10SC;               // Sampling and conversion start
      while (ADC10CTL1 & BUSY);                 // Wait if ADC10 core is active
      __delay_cycles(100);
      ADC10CTL0 &= ~ENC;
      ADC0 = ADC10MEM;

      Digital();                                // Digital sampling
```

```
      char output[] = {"1XX2XX3XX4XX5XX6XX7XZ"};        // String to send

      output[1] = ADC0&0xFF;;                            // Least Significant Bit
      output[2] = (ADC0>>8)&0xFF;                        // Most Significant Bit
      output[4] = ADC1&0xFF;;
      output[5] = (ADC1>>8)&0xFF;
      output[7] = ADC2&0xFF;;
      output[8] = (ADC2>>8)&0xFF;
      output[10] = ADC3&0xFF;
      output[11] = (ADC3>>8)&0xFF;
      output[13] = ADC4&0xFF;
      output[14] = (ADC4>>8)&0xFF;
      output[16] = ADC5&0xFF;
      output[17] = (ADC5>>8)&0xFF;
      output[19] = Digita;                               // Send digital port

  switch (select)
        {
          case 'A':
            {
              TXString(output, 21 );                     // Send UART
              break;
            }
          case 'B':
            {
              TXStringWifi(output, 21 );                 // Send SimpliciTI
              break;
            }
          case 'C':
            {
              TXwifi(output, 21 );                       // Send UART-SimpliciTI
              break;
            }
        default:
          break;
        }
   ADC10CTL0 &= ~ADC10IFG;                               // Rest flag ADC10
}

/*-------------------------------------------------------------------------
                          TXString routine
-------------------------------------------------------------------------*/
void TXString( char* string, int length )               // Send string
{
  int pointer;
  char* msg_ptr;
  pointer = 0;

  for( pointer = 0; pointer < length; pointer++)
  {
    __delay_cycles(8000);
    UCA0TXBUF = string[pointer];                        // Send character UART
    __delay_cycles(3000);
    while (!(IFG2&UCA0TXIFG));                          // USCI_A0 TX buffer ready?
  }
}
```

```
/*-------------------------------------------------------------------
                        TXStringWifi routine
-------------------------------------------------------------------*/
void TXStringWifi( char* string, int length )          // Send string
{
  int pointer;
  char* msg_ptr;
  pointer = 0;

  for( pointer = 0; pointer < length; pointer++)
  {
    msg_ptr[0] = string[pointer];
    SMPL_Send(sLinkID1, msg_ptr, 1);                   // Send character SimpliciTI
    __delay_cycles(8000);
  }
}

/*-------------------------------------------------------------------
                        TXString routine
-------------------------------------------------------------------*/
void TXwifi( char* string, int length )                // Send string
{
  int pointer;
  char* msg_ptr;
  pointer = 0;

  for( pointer = 0; pointer < length; pointer++)
  {
    __delay_cycles(1000);
    UCA0TXBUF = string[pointer];                       // Send character UART
    __delay_cycles(8000);                              // Wait 8000 clock cycles
    msg_ptr[0] = string[pointer];
    SMPL_Send(sLinkID1, msg_ptr, 1);                   // Send character SimpliciTI
    __delay_cycles(1000);
    while (!(IFG2&UCA0TXIFG));                         // USCI_A0 TX buffer ready?
  }
}

/*-------------------------------------------------------------------
                        Digital routine
-------------------------------------------------------------------*/
void Digital()
{
  Digita = 0x00;                                       // Porting IN data

  if((P4IN&0x01)==0x01)
  {
    Digita= 0x01|Digita;                               // Incorporate P4.0 to Digita
  }
  if((P4IN&0x02)==0x02)
  {
    Digita= 0x02|Digita;                               // Incorporate P4.1 to Digita
  }
  if((P4IN&0x04)==0x04)
  {
    Digita= 0x04|Digita;                               // Incorporate P4.2 to Digita
  }
}
```

Anexo 4 Aplicación desarrollada en LabView.

Para poder leer los datos que llegan de forma serial provenientes de la Target Board eZ430-RF2500T, se realizo una interfaz en LabView que fuera capaz de adquirir los datos y procesarlos. A continuación se muestra

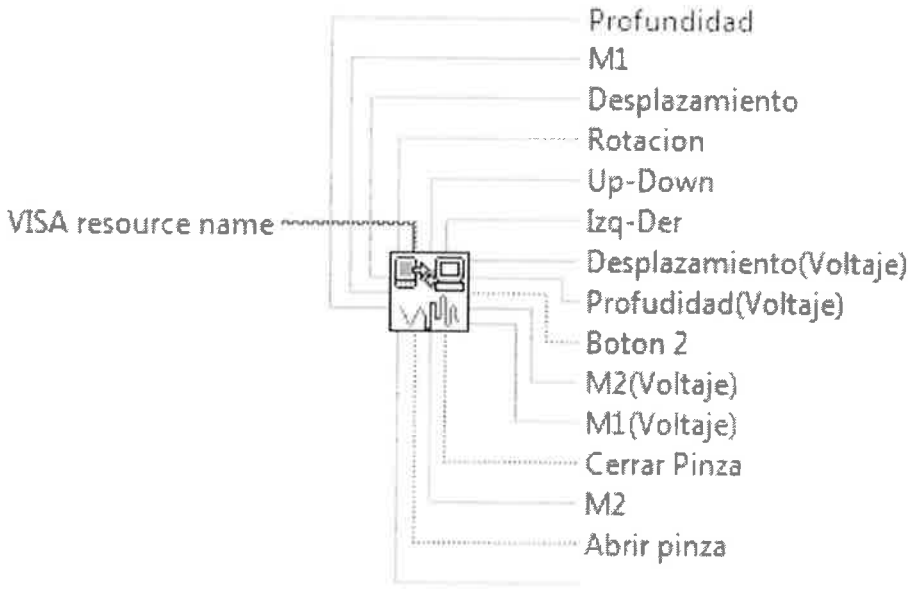
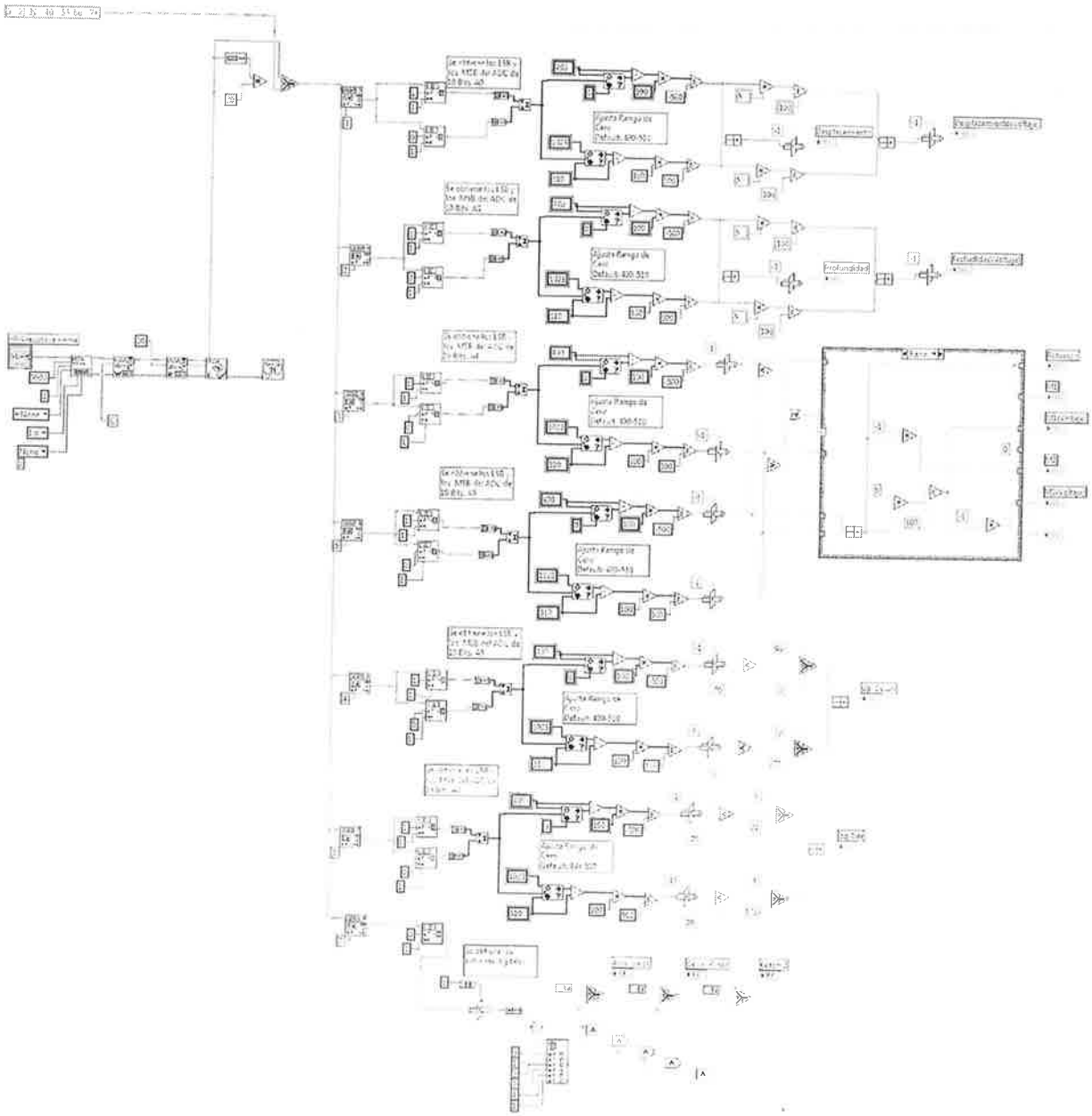


Figura 56 Virtual Instrument creado.



BIBLIOGRAFÍA.

[1] ECA ROBOTICS
http://www.eca-robotics.com/index-sp.htm

[2] SAAB TECHNOLOGIES
http://www.seaeye.com/falcon.html

[3] OCEANNEERING
http://www.oceaneering.com/rovs/rov-systems/spectrum-rov/

[4] MSP430tm Ultra-Low-Power Microcontrollers.
http://www.ti.com/lit/sg/slab034v/slab034v.pdf

[5] Datasheet MSP430g2553
http://focus.ti.com/docs/prod/folders/print/msp430g2553.html

[6] Datasheet MSP430f2274
http://focus.ti.com/docs/prod/folders/print/msp430f2274.html

[7] Datasheet CC2500
http://focus.ti.com/docs/prod/folders/print/cc2500.html

[8] Web oficial SimpliciTi
http://focus.ti.com/docs/toolsw/folders/print/simpliciti.html

[9] eZ340-RF2500 development tools
http://focus.ti.com/docs/toolsw/folders/print/ez430-rf2500-seh.html

[10] MSP430tm Ultra-Low-Power Microcontrollers.
http://www.ti.com/lit/sg/slab034v/slab034v.pdf