ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"Optimización de algoritmos para seguidor de línea utilizando técnicas de control PID con el robot Pololu 3pi e incorporación de control inalámbrico por radio frecuencia."

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN ELECTRICIDAD ESPECIALIZACIÓN EN ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL.

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES.

Presentado por:

Diego David Bósquez Granja

David Miguel Martínez Rodríguez

GUAYAQUIL – ECUADOR

AÑO 2011

AGRADECIMIENTO

A Dios.

A todas las personas que contribuyeron en el desarrollo de este trabajo.

Al Ing. Carlos Valdivieso.

A las personas que en conjunto forman a la ESPOL por brindarnos la oportunidad de adquirir conocimientos y formarnos profesionalmente.

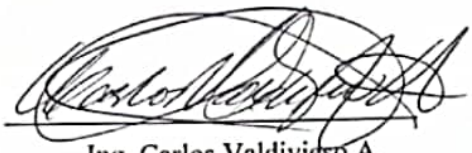
DEDICATORIA

A Dios, nuestro creador por su amor y protección.

A nuestros maestros, por su paciencia, bondad y generosidad por iluminar nuestra senda con la luz del conocimiento, por enseñarnos a ser útiles a los demás.

A nuestros amigos y compañeros, por la alegría compartida, por ser espíritu de solidaridad, por animarnos constantemente hacia la culminación de nuestra carrera.

TRIBUNAL DE SUSTENTACIÓN



Ing. Carlos Valdivieso A.

Profesor del Seminario de Graduación

Ing. Hugo Villavicencio V.

Delegado del Decano

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este trabajo, nos corresponde exclusivamente; y el patrimonio intelectual del mismo a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL".

(Reglamento de exámenes y títulos profesionales de la ESPOL)

Diego David Bósquez Granja

David Miguel Martínez Rodríguez

RESUMEN

El objetivo de este proyecto de tesis es lograr la implementación de un robot seguidor de línea con lazo de control PID para la posición, con la finalidad de optimizar el tiempo de reacción de los sensores infrarrojos en seguidor de línea.

Lo que realiza este sistema es una detección de la posición del Pololu 3Pi mediante el uso de sensores infrarrojos y el manejo del mismo a través de una tarjeta Butterfly con joystick a distancia que tienen integrado dispositivos de radiofrecuencia, teniendo en cuenta de que tanto el Pololu 3Pi como la tarjeta Butterfly tienen integrado pantallas LCD's, podemos programar una sencilla interfaz entre el usuario y el robot.

Este proyecto resalta la aplicación de la programación que se realiza mediante el microcontrolador ATMEL 'atmega328p'

ÍNDICE GENERAL

INDICE DE FIGURAS

INTRODUCCIÓN

En los siguientes capítulos de este informe, se explicará sobre la programación e implementación del controlador ATmega328P del Pololu 3Pi un pequeño robot autónomo, usado para competiciones de seguimiento de línea que es capaz de alcanzar velocidades mayores a 100 cm/s mientras realiza vueltas y cambios de sentido que no varían el voltaje de las baterías.

El primer capítulo contiene una descripción del proyecto, las partes que lo componen y las funciones que es capaz de realizar, además contiene las aplicaciones más comunes en las cuales es utilizado el proyecto y sistemas de similares características.

El segundo capítulo contiene una descripción general del software y el hardware utilizado para la elaboración del proyecto; entre las herramientas de software de programación, tenemos el AVR Studio 4, con sus respectivas características, el simulador de circuitos Proteus con sus funciones principales. Entre las herramientas de Hardware se encuentran: La pantalla gráfica LCD Butterfly de 4x25 segmentos, el Pololu 3Pi y el dispositivo de Radiofrecuencia.

El tercer capítulo contiene todo lo referente a la descripción e implementación del proyecto y las pruebas realizadas.

El cuarto capítulo, se muestra la simulación final en PROTEUS y los esquemas utilizados para el correcto funcionamiento del proyecto. Además se adjuntan los diagramas y esquemas electrónicos.

CAPÍTULO 1

1.- DESCRIPCIÓN GENERAL DEL PROYECTO

1.1. Antecedentes

En empresas muy importantes, podemos observar la aplicación de robots como herramientas para facilitar o realizar ciertas tareas que tienen un grado de peligro y dificultad para los humanos sin influir de manera negativa en la calidad de los productos.

Las aplicaciones más comunes son el transporte de carga en la industria, desactivadores de explosivos, exploración de terrenos de difícil acceso e incluso se han utilizado en puertos de descarga.

En la rama de la instrumentación, la familiaridad con el control PID es constante. Es común observar la instalación, sintonización y utilización de estos controladores. La presencia de algoritmos cuya estructura puede ser mejorada, es la parte sustancial para este proyecto con control PID. Entendimiento y comprensión son los elementos claves

para optimizar el comportamiento del lazo de control. Se requiere un conocimiento específico del proceso así como del control.

Entre las ventajas de usar el PID, están la unión de las características proporcional que provoca el producto entre la señal de error y la constante proporcional para anular el efecto del error y establece una rápida respuesta a cambios o perturbaciones, una integral que tiene como propósito restringir el error en estado estacionario, y la parte derivativa se manifiesta cuando hay un cambio en el valor absoluto del error.

Un robot seguidor de línea se caracteriza por el empleo de mecánica, transductores que varían en número dependiendo de la complejidad del proyecto y electrónica por lo que se encuentra en el campo de la robótica móvil cuyo funcionamiento se basa en el movimiento, sensores y controles que les permitan decidir. Su principal misión es seguir una línea marcada en el suelo.

Las variedades de estos dispositivos van desde el seguidor de una línea hasta los robots que recorren laberintos, entre todos comparten partes similares.

1.2. Descripción del proyecto

El funcionamiento del sistema se basa en un robot seguidor de línea dependiendo de la complejidad del recorrido. Los seguidores de línea negra más simples utilizan 2 sensores, ubicados en la parte inferior de la estructura, uno junto al otro, en nuestro caso existen 5 sensores. Cuando uno de los sensores detecta el color blanco, significa

que el robot está saliendo de la línea negra por ese lado. El robot debe girar hacia el lado contrario hasta que retome su posición sobre la línea

El 3pi de Pololu es un pequeño robot autónomo de alto rendimiento, que puede competir contra otros. El 3pi puede alcanzar velocidades por encima de los 100cm/s en línea recta y da la posibilidad de cambios de sentido.

Podemos afirmar que un robot seguidor de línea en general posee:

Sensores infrarrojos, dos micro motores de corriente continua, ruedas que son movidas por los motores que son anti-deslizantes para evitar fallas de tracción, tarjeta de control que se encarga de la toma de decisiones y el control de los motores están generalmente a cargo de un microcontrolador. La tarjeta de control contiene dicho elemento, junto a otros componentes electrónicos básicos que requiere el microcontrolador.

1.3. Aplicaciones

Entre las aplicaciones de los robots móviles se encuentran el trasporte de la carga en la industria, robots desactivadores de explosivos, exploración de terrenos no aptos para el hombre; entre otras dependiendo (del tamaño de estos).

El campo de acción de este proyecto está relacionado con la optimización de algoritmos de control para robot seguidores de línea, es decir, mejoramiento de lazos de control PID para un mejor desempeño.

1.4. Descripción del problema

Mejorar el lazo de control PID del robot seguidor de línea Pololu 3pi, mediante el uso de un sistema de control a través de sensores infrarrojos, estos sistemas son muy utilizados en las industrias en diferentes procesos en la aplicación de nuestro proyecto se optimiza el lazo de control PID.

Pero debemos tener en cuenta que para diferentes procesos existen diferentes variables así que los lazos de control deberán ser relacionas con el trabajo a realizar y los resultados que se deberán obtener mediante el uso de esta técnica.

1.5. Proyectos Similares

1.5.1 Robot Araña

El diseño de este modelo fue inspirado en los movimientos de una araña, con sus múltiples patas ofrece buena estabilidad al trasladarse, aunque en demostraciones se ha observado un movimiento lento en el avance, el cual se muestra en la Figura 1-1.

Figura1-1: Robot Araña

1.5.2 Combate

Este tipo de robots se han diseñado para combates con otros modelos robóticos simplemente para demostración o diversión de sus creadores, poniendo a prueba sus avances, mejoras y diseños, además de que con cada diseño y mejora los creadores hacen actualizaciones. El cual se muestra en la figura 1-2.

Figura 1-2: Robot Combate

CAPÍTULO 2

2. FUNDAMENTO TEÓRICO

En el presente capítulo se demostrará el uso de las herramientas de Software y Hardware utilizadas para nuestro proyecto. Además se indicarán ciertas características embebidas en el desarrollo del tema:

El Compilador GCC, el simulador Proteus, la pantalla LCD Gráfica butterfly que incluye joystick, el microcontrolador AVR ATmega128 que se programará para controlar el robot POLOLU 3PI, y la interfaz de radiofrecuencia “Emisor, Receptor”.

2.1. Herramientas de Software

2.1.1. Software AVR Studio 4

El programa AVR Studio 4, desarrollado por ATMEL; es un entorno de programación desarrollado para microcontroladores ATMEL, y se lo puede utilizar de mejor forma que la programación tradicional de assembler.

Características del AVR Studio 4:

- Un núcleo del lenguaje simple, con funcionalidades añadidas importantes, como funciones matemáticas y de manejo de archivos, proporcionadas por bibliotecas.
- Es un lenguaje muy flexible que permite programar con múltiples estilos. Uno de los más empleados es el estructurado "no llevado al extremo"
- Usa un lenguaje de pre procesado, el preprocesador de C, para tareas múltiples archivos de código fuente.
- Acceso a memoria de bajo nivel mediante el uso de punteros.
- Un conjunto reducido de palabras clave.
- Por defecto, el paso de parámetros a una función se realiza por valor.
- Plataforma de trabajo, Windows, Linux.
- Genera un código de maquina compatible con simuladores como Proteus
- Punteros a direcciones de memoria, que permiten el uso de la información necesaria.

Se muestra la ventana principal del entorno de trabajo en la Figura 2-1.

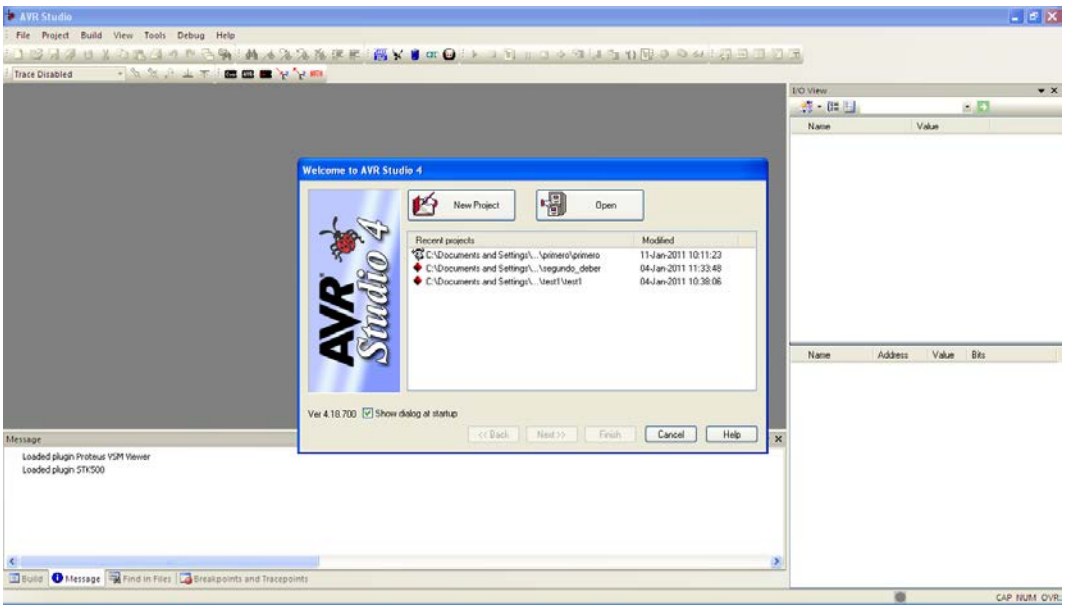


FIGURA 2-1: Entorno AVR Studio 4

2.1.2. PROTEUS (Versión 7.7)

PROTEUS versión 7.7, desarrollado por Labcenter Electronics; es un entorno integrado diseñado para la realización completa de proyectos de construcción de equipos electrónicos en todas sus etapas: diseño, simulación, depuración y construcción. Este emulador consta de dos programas principales: Ares e Isis, y los módulos VSM y Electra.

- ISIS.- *Intelligent Schematic Input System* (Sistema de Enrutado de Esquemas Inteligente); es la herramienta que permite simular el esquema eléctrico del proyecto; incorporando una librería de más de 6.000 modelos de dispositivos digitales y analógicos como son: resistencias,

microcontroladores, leds, relés, microprocesadores, incluyendo fuentes de alimentación, generadores de señales, etc.

- ARES.- AdvancedRouting and Editing Software (Software de Edición y Ruteo Avanzado); es la herramienta de enrutado, ubicación y edición de componentes, se utiliza para la fabricación de placas de circuito impreso (PCB's). En la siguiente Figura se muestra la ventana principal Figura 2-2.

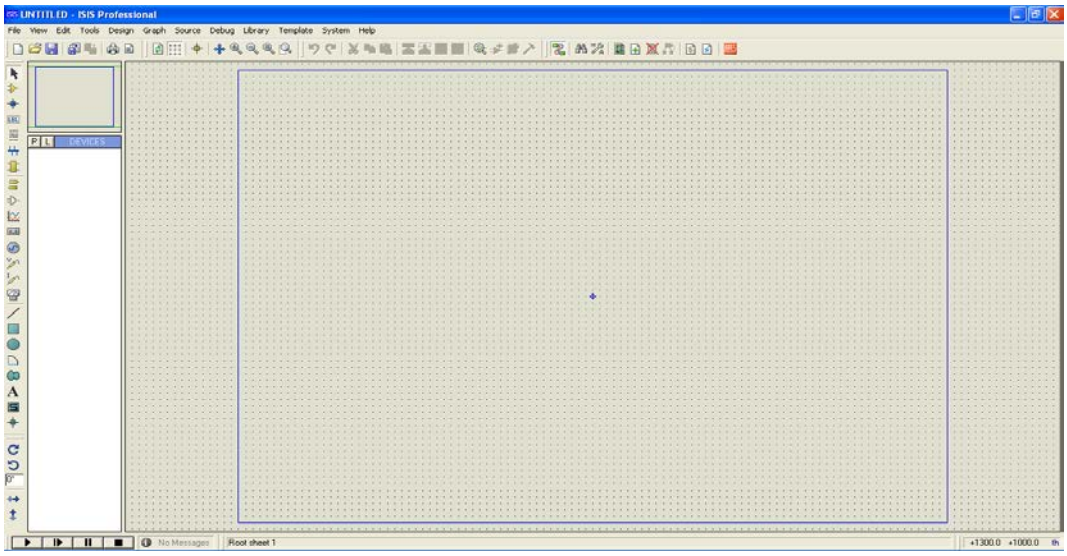


FIGURA 2-2: Entorno PROTEUS V 7.7

2.2. Herramientas de Hardware

2.2.1 Tarjeta Butterfly

La Pantalla LCD Gráfica viene dada en segmentos (4x25) la cual tiene incorporado un joystick, que se usará para el manejo del Pololu 3pi. Además gracias a su flexibilidad y buena visibilidad se ha convertido en el estándar de visualización más utilizado con los microcontroladores. Como se muestra en la Figura 2-3.

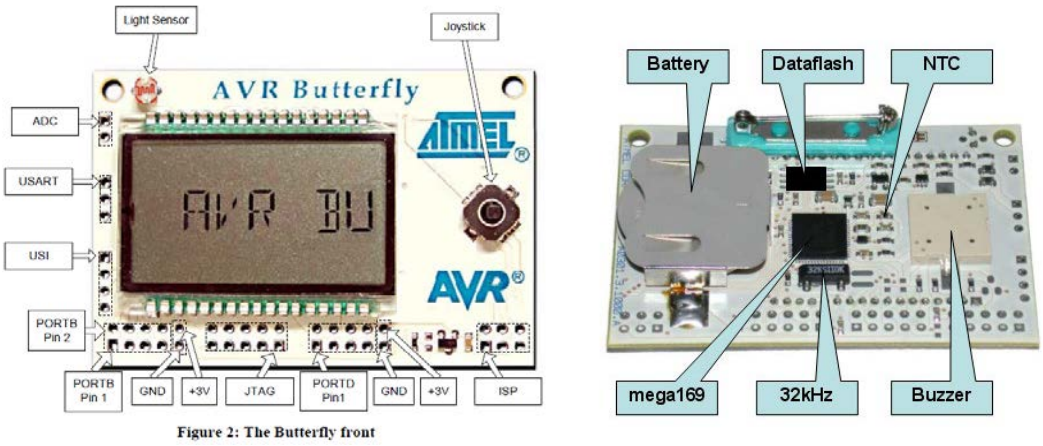


Figure 2: The Butterfly front

FIGURA 2-3: Pantalla LCD Gráfica (4x25) Butterfly

2.2.1.1. Características de la Tarjeta Butterfly:

La tarjeta Butterfly con microcontrolador Atmega169 integra los siguientes parámetros:

Memoria Flash de 16KB, Memoria EEPROM 512 Bytes, Memoria SRAM 1KB, pantalla LCD de 4x25 segmentos, dos timer/Counters de 8 bits , un timer/Couter de 16 bits, 4 Canales PWM, Watchdog Timer, comparador analógico.

2.2.1.2. Microcontrolador ATMEGA169

A continuación se muestra el microcontrolador ATMega169 Figura 2-4

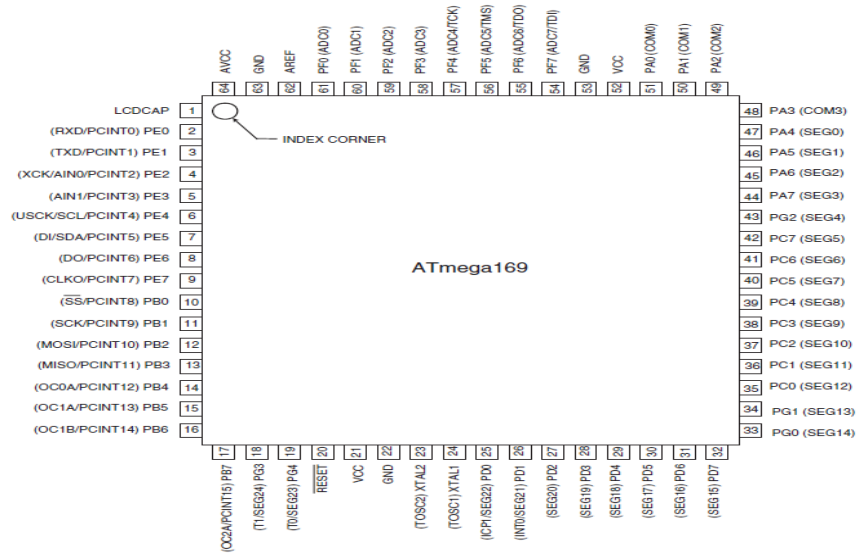


FIGURA 2-4: Visualización del microcontrolador ATMega169 de la tarjeta Butterfly.

2.2.1.3. Diagrama de bloques interno de la tarjeta Butterfly: Figura 2-5

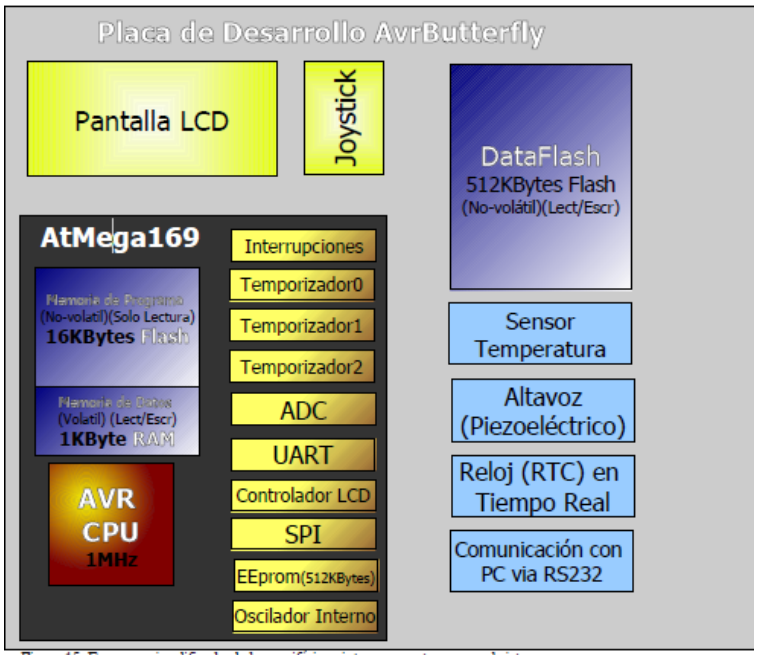


FIGURA 2-5: Diagrama de distribución interna de la tarjeta Butterfly Atmega169

2.2.2 Robot Pololu 3Pi

El robot está totalmente ensamblado con dos micro motores de metal para las ruedas, cinco sensores de reflexión, una pantalla LCD de 8x2 caracteres, un buzzer, tres pulsadores y más, todo ello conectado a un microcontroladores programable AVR. El 3pi mide aproximadamente 9,5 cm (3,7”) de diámetro y pesa alrededor de 83 gr. (2,9 oz.) sin baterías.

El 3pi se basa en un microcontroladores Atmel ATmega168 a 20 MHz con 16Kb de memoria flash y 1Kb de memoria de datos. El uso del ATmega168 lo hace compatible con la plataforma de desarrollo Arduino. Las herramientas gratuitas de desarrollo en C y C++ están disponibles así como un extenso paquete de librerías que

pueden trabajar con el hardware que lleva integrado. Están disponibles simples programas que muestran cómo trabajan los diferentes componentes 3pi. Como se muestra a continuación en la figura 2-6.

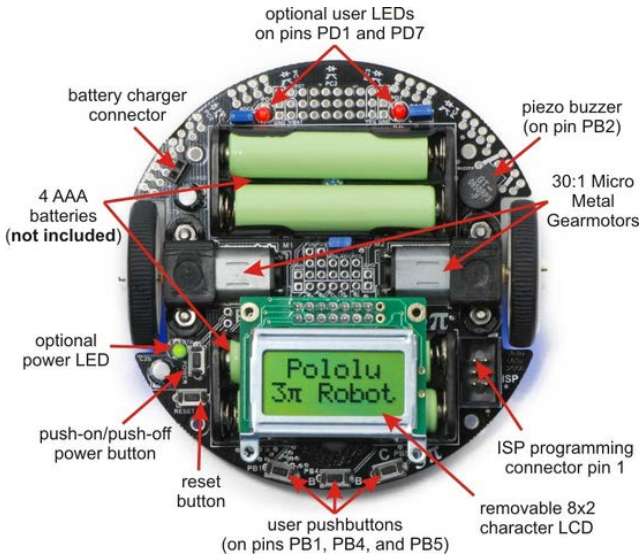
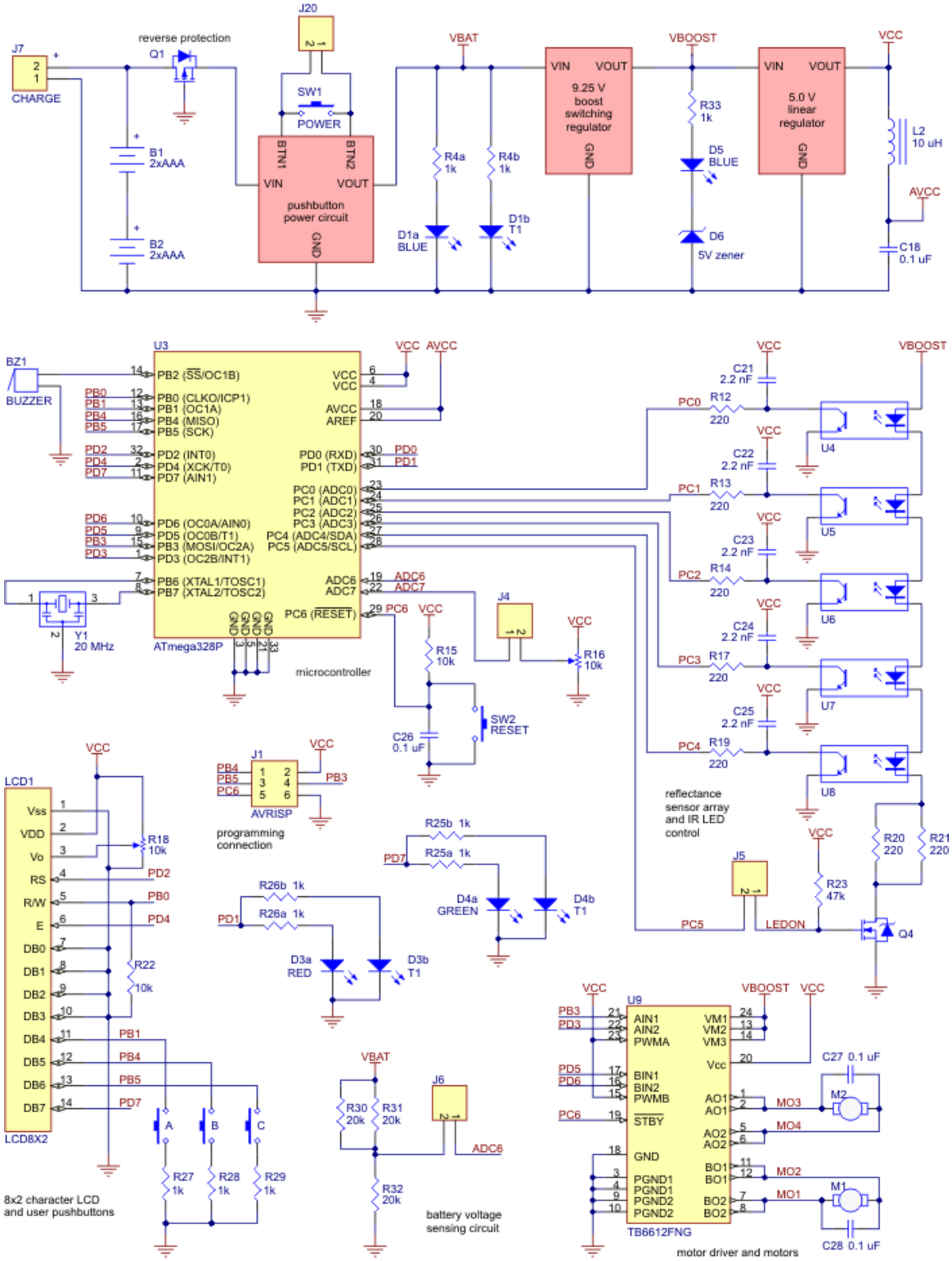


FIGURA 2-6: Robot Pololu 3Pi

2.2.2.1 Esquema del circuito simplificado



2.2.2.2 Tabla de asignación de PINES según función

función	ATmega169 Pin
I/O digitales (x3) (quita jumper PC5 para liberar el pin 19 digital)	PD0, PD1, PC5
Entradas analógicas (quita los jumpers, x3)	PC5, ADC6, ADC7
motor 1 (izquierdo) control (A y B)	PD5 y PD6
motor 2 (derecho) control (A y B)	PD3 y PB3
QTR-RC sensores de reflexión (izquierda a der, x5)	PC0 – PC4
rojo (izquierda) LED de usuario	PD1
verde (derecha) LED de usuario	PD7
Botones de usuario (left to right, x3)	PB1, PB4, y PB5
Buzzer	PB2
LCD control (RS, R/W, E)	PD2, PB0, y PD4
LCD data (4-bit: DB4 – DB7)	PB1, PB4, PB5, y PD7
sensor IR LED control drive low to turn IR LEDs off)	PC5
trimmer potenciometro de usuario	ADC7
2/3rds de voltaje de la batería	ADC6
ICSP lienas de programación (x3)	PB3, PB4, PB5
Botón de REST	PC6
UART (RX y TX)	PD0 y PD1
I2C/TWI	inaccesible al usuario
SPI	inaccesible al usuario

2.2.3 RADIOFRECUENCIA: DEFINICIÓN Y /O CONCEPTOS.

Al estudiar este tipo de medio de transmisión de datos es importante saber que cualquier transmisión de datos puede efectuarse sin ningún tipo de hilo, simplemente utilizando dispositivos que transportan la información mediante ondas. Por ejemplo la información recibida a través de radio y televisión.

La radiofrecuencia es en efecto un tipo de onda electromagnética que es muy semejante a la energía luminosa, y tiene la misma velocidad que la luz que es

300,000,000 metros por segundo. Las ondas de radio pueden generarse en una amplia gama de frecuencias, empezando aproximadamente de 10,000 hz y siguiendo a través de millones de hertzios hasta miles de millones. Se han incluido también ondas electromagnéticas, como luz visible.

Las características de propagación de las ondas de radio a través de la atmósfera varían en gran medida con la frecuencia y deben tenerse presentes a la hora de elegir una frecuencia para un servicio de radio en particular. Las ondas de radio se dividen en diferentes bandas de frecuencia de acuerdo con sus características de propagación.

Algunos de los servicios típicos asignados a las diferentes bandas de frecuencia son:

V.L.F. Radio difusión telegráfica a larga distancia.

L.F. Servicio punto a punto de larga distancia, ayudas a la navegación, difusión de sonido.

Sistemas de portadora por línea.

M.F. Difusión de sonido, servicios costeros para embarcaciones, sistemas de portadoras Líneas.

H.F. Servicios punto a punto para distancias media y larga, difusión de sonidos, sistemas De portadoras por líneas.

V.H.F. Comunicaciones a corta distancia, difusión de tv y sonido, radar.

U.H.F. Servicios aire-aire y tierra aire.

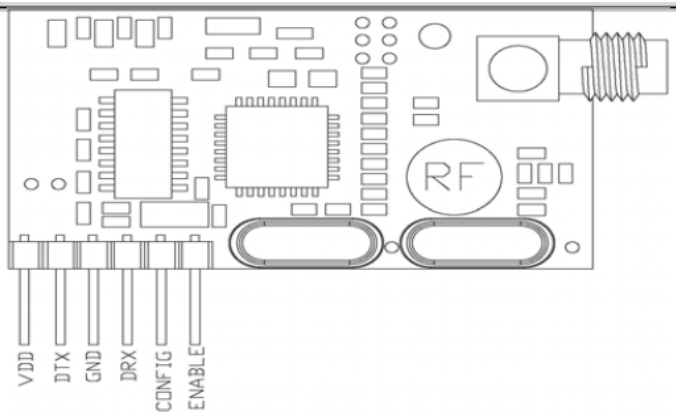
S.H.F. Sistemas de comunicación de microondas punto a punto, radar.

Este tipo de transmisión se realiza a través de:

- microondas
- Infrarrojos
- Láser
- Otros

2.2.3.1 Módulos de radiofrecuencia

Módulos HM-TR, uno conectado Pololu 3 Pi y otro conectado a la tarjeta AVR BUTTERFLY. Como se muestra en la Figura 2-7.



Pin	name	note
1	VDD	Power supply
2	DTX	Data output from module
3	GND	Ground
4	DRX	Data input to module
5	CONFIG	If this pin is high at power on, module will enter configure mode, while it communicates if set low
6	ENABLE	If this pin is low in normal mode, the module will enter sleep mode immediately. Assert high will awaken

FIGURA 2-7: Módulos de radiofrecuencia

CAPÍTULO 3

3. DESCRIPCIÓN E IMPLEMENTACIÓN DEL PROYECTO

3.1 Implementación del Proyecto.

Para la implementación física del proyecto se necesitó de los siguientes componentes detallados a continuación:

- Robot Pololu 3Pi seguidor de línea por medio de sensores infrarrojos controlado por radiofrecuencia, importado de USA.
- Cuatro pilas triple A recargables, que se utilizan como fuente de poder del Pololu 3Pi.
- Tarjeta Butterfly con joystick, en la cual ya viene incorporada una pila de 3 voltios como fuente de poder, importado de USA.
- Condensadores de 10 uF, en conjunto al integrado MAX 232, elevan los 3 voltios de salida del Butterfly a un nivel TTL (5 voltios).
- Un regulador 7805 con dos baterías de 9 voltios en serie conectados al pin de entrada del mismo.

- Componentes electrónicos emisor y receptor por radiofrecuencia, los cuales serán incorporados a nuestro proyecto final para enviar las señales de control desde la tarjeta Butterfly al Pololu 3Pi por medio de radiofrecuencia, la polarización del dispositivo receptor se tomó del mismo Pololu 3Pi 'Polarización de 5 voltios'.
- Programador ISP, se usó para la programación del robot Pololu 3Pi, importado de USA.

3.2 Descripción del Proyecto Final.

Al finalizar con la programación y la implementación del prototipo, podemos hacer una descripción del funcionamiento del proyecto:

I. Se enciende el dispositivo Pololu 3PI, conexión por baterías triple A recargables.
II. Se inicializa el robot presionado el botón B, el cual muestra un movimiento de motores.
III. Se presiona B nuevamente en ese momento se escucha un sonido, entonces el robot pololu 3Pi estará listo para recibir las ordenes de radiofrecuencia desde la tarjeta butterfly 'El robot debe estar sobre la pista'
IV. Al presionar el joystick de la butterfly el robot pololu 3Pi realiza los movimientos indicados en la LCD del módulo Butterfly.

V. La LCD muestra cada uno de los movimientos que realizará el Pololu 3pi dentro de la pista.

VI. En el diagrama de flujo del seguidor de línea se encuentra la respuesta a cada uno de los códigos recibidos, desde el PID 1 hasta el PID5, cada uno de estos indican diferentes movimientos controlados dentro de la misma pista.

PID1.- Es el modo normal en donde la velocidad no es tan rápida, se trata de mantener una velocidad constante durante todo el trayecto.

PID2.-Es el modo rápido, se mantiene una velocidad alta que mantenga el equilibrio del robot tratando de tener una velocidad constante.

PID3.- Se emplea en el modo de tramos rápidos y lentos, este modo desarrolla un incremento de velocidad hasta un tope y luego un decremento hasta casi llegar a una detención, luego se repite el proceso de manera cíclica.

PID4.- Es una detención total del robot en línea recta, si está en curva el robot intenta centrar su posición en la línea negra para detenerse.

PID5.-Es el modo de rectas rápidas y curvas lentas en donde una velocidad alta es alcanzada en línea recta y las curvas son tomadas con lentitud.

3.3 Diagrama de Bloques del proyecto

Figura 3-1 muestra el diagrama de bloques de nuestro proyecto.

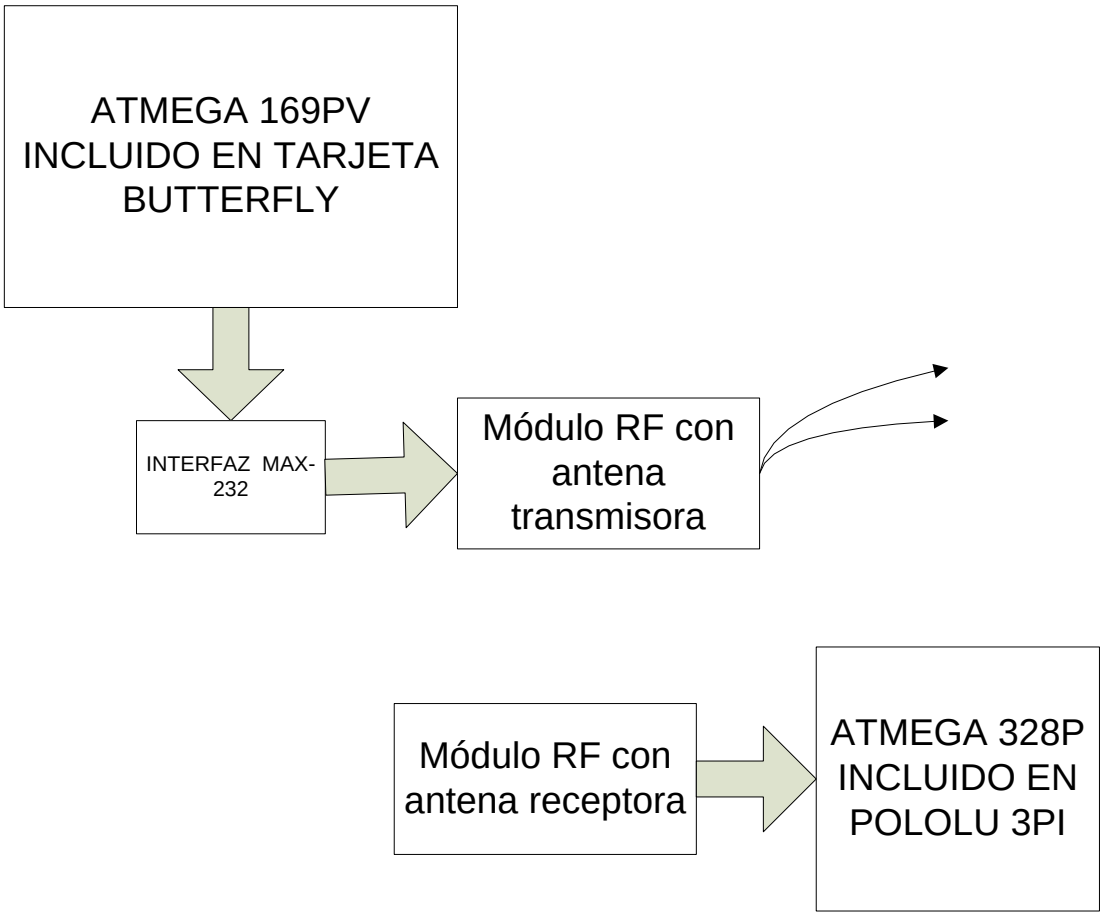
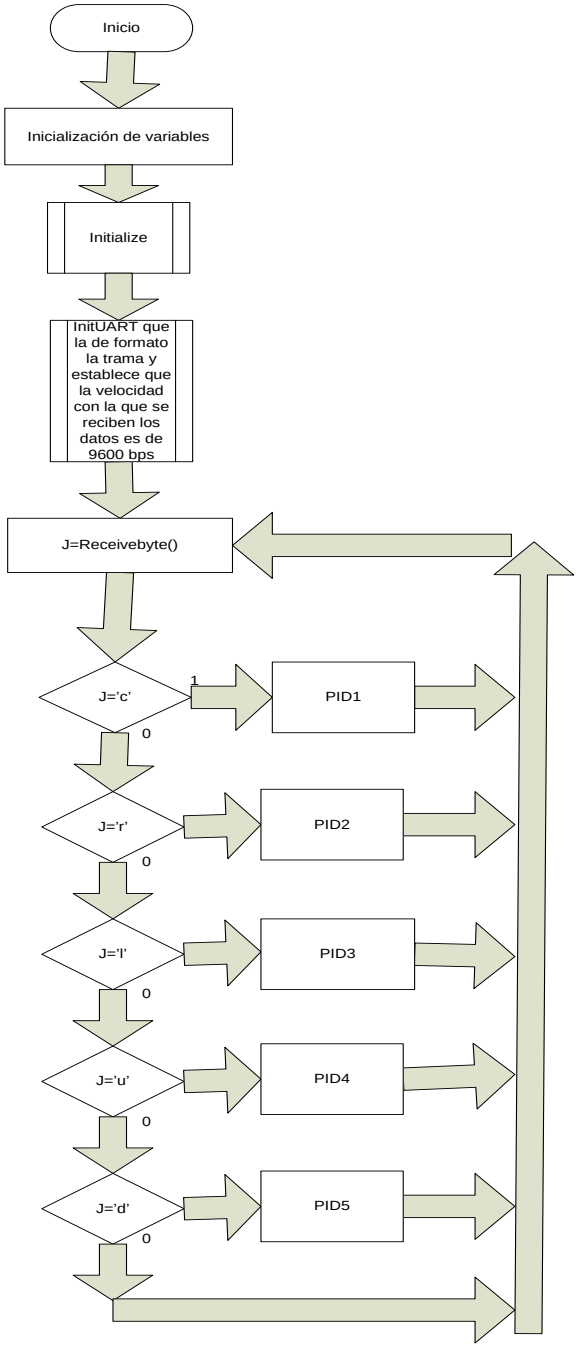
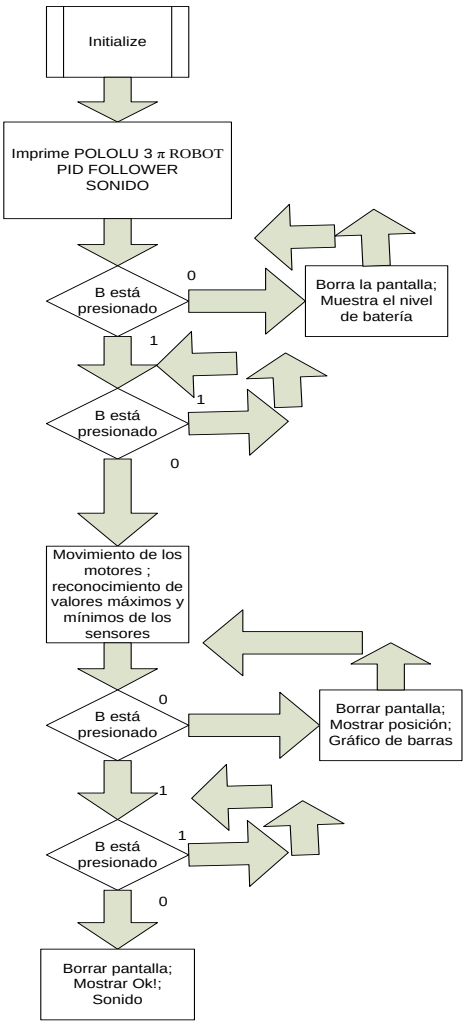
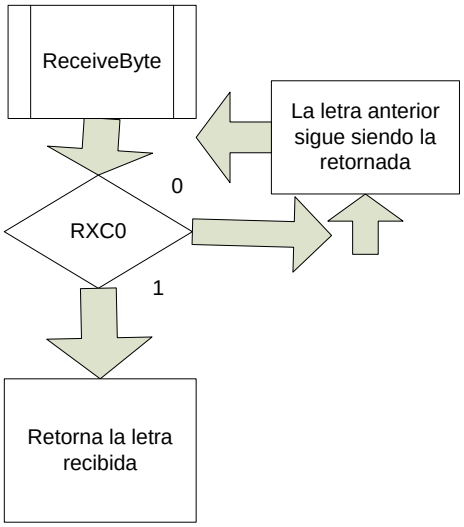


FIGURA 3-1: Diagrama de bloques del proyecto

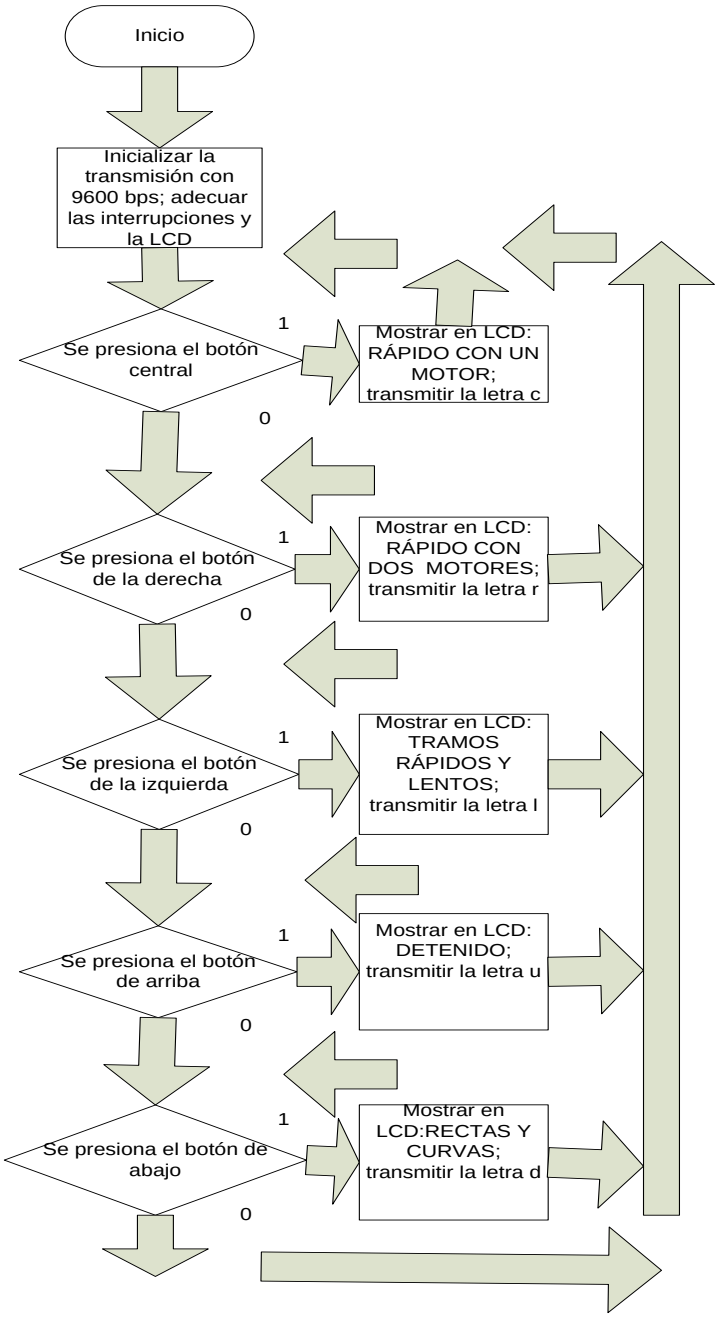
3.4 Diagrama de Flujo del Pololu 3Pi






ReceiveByte
La letra anterior sigue siendo la retornada
0
RXC0
1
Retorna la letra recibida

3.5 Diagrama de flujo del Butterfly



3.6 Programas principales.

3.6.1 PARA EL ATMEGA 169P DEL BUTTERFLY

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>

#include <inttypes.h>

#include "mydefs.h"
#include "LCD_functions.h"
#include "LCD_driver.h"
#include "button.h"
#include "usart.h"



int main(void){
   PGM_P statetext = PSTR("CONTROL RF");
  uint8_t input;



 // Disable Analog Comparator (power save)
   ACSR = (1<<ACD);

  // Disable Digital input on PF0-2 (power save)
   DIDR0 = (7<<ADC0D);

  // Enable pullups
  PORTB = (15<<PB0);
  PORTE = (15<<PE4);

   Button_Init();            // Initialize pin change interrupt on joystick

   LCD_Init();               // initialize the LCD

  CLKPR = (1<<CLKPCE);       // set Clock Prescaler Change Enable
   //  4Mhz
   CLKPR = (0<<CLKPS1) | (1<<CLKPS0);
```

```
USART_Init(25);    //  9600 BPS




 while (1)
 {
    if (statetext){

     LCD_puts_f(statetext, 1);
     LCD_Colon(0);
     statetext = NULL;
              }


  input = getkey();          // Read buttons

  switch (input) {
    case KEY_ENTER:




       statetext = PSTR("NORMAL");
       Usart_Tx('c');


       break;
    case KEY_NEXT:
       statetext = PSTR("RAPIDO");
       Usart_Tx('r');

       break;
    case KEY_PREV:
       statetext = PSTR("TRAMOS RAPIDOS Y LENTOS");
       Usart_Tx('l');

       break;
    case KEY_PLUS:

       statetext = PSTR("DETENIDO");

       Usart_Tx('u');
```

```
          break;
        case KEY_MINUS:
          statetext = PSTR("RECTAS Y CURVAS");
          Usart_Tx('d');

          break;
        default:
          break;

    }

    }

  return 0;
}
```

USART:

```
#include <avr/io.h>
#include "usart.h"

/* 25. May 2005 - adapted to avrlibc iom168.h version 1.17 */

void USART_Init( unsigned int ubrr) /* ALLOWS SETTING OF BPS */

{
     UBRRH = (unsigned char)(ubrr>>8);
     UBRRL = (unsigned char)ubrr;
    UCSRA = (0<<U2X);
     UCSRB = (1<<RXEN)|(1<<TXEN)|(0<<RXCIE)|(0<<UDRIE);
     UCSRC = (0<<UMSEL)|(0<<UPM0)|(0<<USBS)|(3<<UCSZ0)|(0<<UCPOL);
}


void Usart_Tx(char data) /* TRANSMISSION*/

{
   while (!(UCSRA & (1<<UDRE)));
   UDR = data;
}

char Usart_Rx(void) /* RECEPTION*/
{
   while (!(UCSRA & (1<<RXC)));
   return UDR;
```

}

3.6.2 PARA EL ATMEGA 328P DEL POLOLU 3Pi

```
#include <avr/io.h>
#include <pololu/3pi.h>

#include <avr/pgmspace.h>


/* 5 WAYS WITH PID FOLLOWER WHICH HAVE RADIO CONTROL */



/* Prototypes */
void InitUART (unsigned int ubrr);
unsigned char ReceiveByte (void);


// Introductory messages.  The "PROGMEM" identifier causes the data to
// go into program space.
const char welcome_line1[] PROGMEM = " Pololu";
const char welcome_line2[] PROGMEM = "3\xf7 Robot";
const char demo_name_line1[] PROGMEM = "PID Line";
const char demo_name_line2[] PROGMEM = "follower";

// A couple of simple tunes, stored in program space.
const char welcome[] PROGMEM = ">g32>>c32";
const char go[] PROGMEM = "L16 cdegreg4";

// Data for generating the characters used in load_custom_characters
// and display_readings.  By reading levels[] starting at various
// offsets, we can generate all of the 7 extra characters needed for a
// bargraph.  This is also stored in program space.
const char levels[] PROGMEM = {
    0b00000,
    0b00000,
    0b00000,
    0b00000,
```

```
    0b00000,
    0b00000,
    0b00000,
    0b11111,
    0b11111,
    0b11111,
    0b11111,
    0b11111,
    0b11111,
    0b11111
};

// This function loads custom characters into the LCD.  Up to 8
// characters can be loaded; we use them for 7 levels of a bar graph.
void load_custom_characters()
{
    lcd_load_custom_character(levels+0,0); // no offset, e.g. one bar
    lcd_load_custom_character(levels+1,1); // two bars
    lcd_load_custom_character(levels+2,2); // etc...
    lcd_load_custom_character(levels+3,3);
    lcd_load_custom_character(levels+4,4);
    lcd_load_custom_character(levels+5,5);
    lcd_load_custom_character(levels+6,6);
    clear(); // the LCD must be cleared for the characters to take effect
}

// This function displays the sensor readings using a bar graph.
void display_readings(const unsigned int *calibrated_values)
{
    unsigned char i;

    for(i=0;i<5;i++) {
            // Initialize the array of characters that we will use for the
            // graph.  Using the space, an extra copy of the one-bar
            // character, and character 255 (a full black box), we get 10
            // characters in the array.
            const char display_characters[10] = {' ',0,0,1,2,3,4,5,6,255};

            // The variable c will have values from 0 to 9, since
            // calibrated values are in the range of 0 to 1000, and
            // 1000/101 is 9 with integer math.
            char c = display_characters[calibrated_values[i]/101];

            // Display the bar graph character.
            print_character(c);
```

```
    }
}

// Initializes the 3pi, displays a welcome message, calibrates, and
// plays the initial music.
void initialize()
{
    unsigned int counter; // used as a simple timer
    unsigned int sensors[5]; // an array to hold sensor values


    pololu_3pi_init(2000);
    load_custom_characters(); // load the custom characters

    // Play welcome music and display a message
    print_from_program_space(welcome_line1);
    lcd_goto_xy(0,1);
    print_from_program_space(welcome_line2);
    play_from_program_space(welcome);
    delay_ms(1000);

    clear();
    print_from_program_space(demo_name_line1);
    lcd_goto_xy(0,1);
    print_from_program_space(demo_name_line2);
    delay_ms(1000);

    // Display battery voltage and wait for button press
    while(!button_is_pressed(BUTTON_B))
    {
            int bat = read_battery_millivolts();

            clear();
            print_long(bat);
            print("mV");
            lcd_goto_xy(0,1);
            print("Press B");

            delay_ms(100);
    }

    // Always wait for the button to be released so that 3pi doesn't
    // start moving until your hand is away from it.
    wait_for_button_release(BUTTON_B);
    delay_ms(1000);
```

```
// Auto-calibration: turn right and left while calibrating the
// sensors.
for(counter=0;counter<80;counter++)
{
        if(counter < 20 || counter >= 60)
                set_motors(40,-40);
        else
                set_motors(-40,40);

        // This function records a set of sensor readings and keeps
        // track of the minimum and maximum values encountered.  The
        // IR_EMITTERS_ON argument means that the IR LEDs will be
        // turned on during the reading, which is usually what you
        // want.
        calibrate_line_sensors(IR_EMITTERS_ON);

        // Since our counter runs to 80, the total delay will be
        // 80*20 = 1600 ms.
        delay_ms(20);
}
set_motors(0,0);

// Display calibrated values as a bar graph.
while(!button_is_pressed(BUTTON_B))
{
        // Read the sensor values and get the position measurement.
        unsigned int position = read_line(sensors,IR_EMITTERS_ON);

        // Display the position measurement, which will go from 0
        // (when the leftmost sensor is over the line) to 4000 (when
        // the rightmost sensor is over the line) on the 3pi, along
        // with a bar graph of the sensor readings.  This allows you
        // to make sure the robot is ready to go.
        clear();
        print_long(position);
        lcd_goto_xy(0,1);
        display_readings(sensors);

        delay_ms(100);
}


clear();
```

```
    print("OK!");

    // Play music and wait for it to finish before we start driving.
    play_from_program_space(go);
    while(is_playing());
}

// This is the main function, where the code starts.  All C programs
// must have a main() function defined somewhere.



void main ()

{
 unsigned int sensors[5]; // an array to hold sensor values
    unsigned int last_proportional=0;
    long integral=0;
  int contador1=0;
    int contador2=0;
  char j;
    char lastj='u';

initialize();


  InitUART(129);


            while(1){

            j = ReceiveByte();



                        if (j == 'c')
                {
            lastj='c';
                    // Get the position of the line.  Note that we *must* provide
            // the "sensors" argument to read_line() here, even though we
            // are not interested in the individual sensor readings.
            unsigned int position = read_line(sensors,IR_EMITTERS_ON);

            // The "proportional" term should be 0 when we are on the line.
            int proportional = ((int)position) - 2000;
```

```
// Compute the derivative (change) and integral (sum) of the
// position.
int derivative = proportional - last_proportional;
integral += proportional;

// Remember the last position.
last_proportional = proportional;

// Compute the difference between the two motor power settings,
// m1 - m2.  If this is a positive number the robot will turn
// to the right.  If it is a negative number, the robot will
// turn to the left, and the magnitude of the number determines
// the sharpness of the turn.
int power_difference = proportional/5 + integral/7000+ derivative*7;

// Compute the actual motor settings.  We never set either motor
// to a negative value.
const int max = 100;
if(power_difference > max)
        power_difference = max;
if(power_difference < -max)
        power_difference = -max;

if(power_difference < 0)
        set_motors(max+power_difference, max);
else
        set_motors(max, max-power_difference);


        }




        else if (j=='r')

        {
                lastj='r';
                // Get the position of the line.  Note that we *must* provide
// the "sensors" argument to read_line() here, even though we
// are not interested in the individual sensor readings.
unsigned int position = read_line(sensors,IR_EMITTERS_ON);

// The "proportional" term should be 0 when we are on the line.
```

```
        int proportional = ((int)position) - 2000;

        // Compute the derivative (change) and integral (sum) of the
        // position.
        int derivative = proportional - last_proportional;
        integral += proportional;

        // Remember the last position.
        last_proportional = proportional;

        // Compute the difference between the two motor power settings,
        // m1 - m2.  If this is a positive number the robot will turn
        // to the right.  If it is a negative number, the robot will
        // turn to the left, and the magnitude of the number determines
        // the sharpness of the turn.
        int power_difference = proportional/5+ integral/7000 + derivative*7;

        // Compute the actual motor settings.  We never set either motor
        // to a negative value.
        const int max = 150;
        if(power_difference > max)
                power_difference = max;
        if(power_difference < -max)
                power_difference = -max;

        if(power_difference < 0)
                set_motors(max+power_difference+20,max);
        else
                set_motors(max, max-power_difference+20);









                }

                else if (j=='l')
        {
        lastj='l';
                contador1=0;
// This is the "main loop" - it will run forever.
for (contador1=0;contador1<100;contador1++)
```

```
    {
delay_ms(8);

            // Get the position of the line.  Note that we *must* provide
            // the "sensors" argument to read_line() here, even though we
            // are not interested in the individual sensor readings.
            unsigned int position = read_line(sensors,IR_EMITTERS_ON);


            // The "proportional" term should be 0 when we are on the line.
            int proportional = ((int)position) - 2000;

            // Compute the derivative (change) and integral (sum) of the
            // position.
            int derivative = proportional - last_proportional;
            integral += proportional;

            // Remember the last position.
            last_proportional = proportional;

            // Compute the difference between the two motor power settings,
            // m1 - m2.  If this is a positive number the robot will turn
            // to the right.  If it is a negative number, the robot will
            // turn to the left, and the magnitude of the number determines
            // the sharpness of the turn.
            int power_difference =proportional/10+integral/8000+derivative*3/2;

            // Compute the actual motor settings.  We never set either motor
            // to a negative value.

    const int max =150;
    if(power_difference > max)
                    power_difference = max;
            if(power_difference < -max)
                    power_difference = -max;

            if(power_difference < 0)
                    set_motors(max+power_difference+5,max-contador1);
            else
                    set_motors(max-contador1, max-power_difference+5);


 }

 contador2=100;
```

```
  for (contador2==100;contador2>0;contador2=contador2-1)
     {
delay_ms(8);

            // Get the position of the line.  Note that we *must* provide
            // the "sensors" argument to read_line() here, even though we
            // are not interested in the individual sensor readings.
            unsigned int position = read_line(sensors,IR_EMITTERS_ON);


            // The "proportional" term should be 0 when we are on the line.
            int proportional = ((int)position) - 2000;

            // Compute the derivative (change) and integral (sum) of the
            // position.
            int derivative = proportional - last_proportional;
            integral += proportional;

            // Remember the last position.
            last_proportional = proportional;

            // Compute the difference between the two motor power settings,
            // m1 - m2.  If this is a positive number the robot will turn
            // to the right.  If it is a negative number, the robot will
            // turn to the left, and the magnitude of the number determines
            // the sharpness of the turn.
            int power_difference =proportional/5+integral/8000+derivative*7;

            // Compute the actual motor settings.  We never set either motor
            // to a negative value.

     const int max =150;
     if(power_difference > max)
                  power_difference = max;
            if(power_difference < -max)
                  power_difference = -max;

            if(power_difference < 0)
                  set_motors(max+power_difference+5,max-contador1);
            else
                  set_motors(max-contador1, max-power_difference+5);


            }
```

```
                }

                else if (j=='u')
        {
                lastj='u';
 // Get the position of the line.  Note that we *must* provide
        // the "sensors" argument to read_line() here, even though we
        // are not interested in the individual sensor readings.
        unsigned int position = read_line(sensors,IR_EMITTERS_ON);

        // The "proportional" term should be 0 when we are on the line.
        int proportional = ((int)position) - 2000;

        // Compute the derivative (change) and integral (sum) of the
        // position.
        int derivative = proportional - last_proportional;
        integral += proportional;

        // Remember the last position.
        last_proportional = proportional;

        // Compute the difference between the two motor power settings,
        // m1 - m2.  If this is a positive number the robot will turn
        // to the right.  If it is a negative number, the robot will
        // turn to the left, and the magnitude of the number determines
        // the sharpness of the turn.
        int power_difference =proportional/10+integral/8000+derivative*3/2;

        // Compute the actual motor settings.  We never set either motor
        // to a negative value.


    if((power_difference<=(50+integral/8000+derivative*7))&(power_difference>=(-
50+integral/8000+derivative*7)))
                set_motors(0,0);

     if(power_difference>(50+integral/8000+derivative*7))
                set_motors(35,0);
       if(power_difference<(-50+integral/8000+derivative*7))
                 set_motors(0,35);
```

```
            }
        else if (j=='d')
    {
                    lastj='d';
      // Get the position of the line.  Note that we *must* provide
    // the "sensors" argument to read_line() here, even though we
    // are not interested in the individual sensor readings.
    unsigned int position = read_line(sensors,IR_EMITTERS_ON);

    // The "proportional" term should be 0 when we are on the line.
    int proportional = ((int)position) - 2000;

    // Compute the derivative (change) and integral (sum) of the
    // position.
    int derivative = proportional - last_proportional;
    integral += proportional;

    // Remember the last position.
    last_proportional = proportional;

    // Compute the difference between the two motor power settings,
    // m1 - m2.  If this is a positive number the robot will turn
    // to the right.  If it is a negative number, the robot will
    // turn to the left, and the magnitude of the number determines
    // the sharpness of the turn.
    int power_difference =proportional/5+integral/8000+derivative*7;

    // Compute the actual motor settings.  We never set either motor
    // to a negative value.
    const int max =80;

  if((power_difference<=(100+integral/8000+derivative*7))&(power_difference>=(-
100+integral/8000+derivative*7)))
                set_motors(max,max);

   if(power_difference>(100+integral/8000+derivative*7))
                set_motors(35,0);
     if(power_difference<(-100+integral/8000+derivative*7))
```

```
            set_motors(0,35);
}


          else if (j=='w'){
    j=lastj;
    if (j=='c'){
            // Get the position of the line.  Note that we *must* provide
// the "sensors" argument to read_line() here, even though we
// are not interested in the individual sensor readings.
unsigned int position = read_line(sensors,IR_EMITTERS_ON);

// The "proportional" term should be 0 when we are on the line.
int proportional = ((int)position) - 2000;

// Compute the derivative (change) and integral (sum) of the
// position.
int derivative = proportional - last_proportional;
integral += proportional;

// Remember the last position.
last_proportional = proportional;

// Compute the difference between the two motor power settings,
// m1 - m2.  If this is a positive number the robot will turn
// to the right.  If it is a negative number, the robot will
// turn to the left, and the magnitude of the number determines
// the sharpness of the turn.
int power_difference = proportional/5 + integral/7000 + derivative*7;

// Compute the actual motor settings.  We never set either motor
// to a negative value.
const int max = 100;
if(power_difference > max)
          power_difference = max;
if(power_difference < -max)
          power_difference = -max;

if(power_difference < 0)
          set_motors(max+power_difference, max);
else
          set_motors(max, max-power_difference);
  }
   if (j=='r'){
   // Get the position of the line.  Note that we *must* provide
```

```
        // the "sensors" argument to read_line() here, even though we
        // are not interested in the individual sensor readings.
        unsigned int position = read_line(sensors,IR_EMITTERS_ON);

        // The "proportional" term should be 0 when we are on the line.
        int proportional = ((int)position) - 2000;

        // Compute the derivative (change) and integral (sum) of the
        // position.
        int derivative = proportional - last_proportional;
        integral += proportional;

        // Remember the last position.
        last_proportional = proportional;

        // Compute the difference between the two motor power settings,
        // m1 - m2.  If this is a positive number the robot will turn
        // to the right.  If it is a negative number, the robot will
        // turn to the left, and the magnitude of the number determines
        // the sharpness of the turn.
        int power_difference = proportional/5+ integral/7000 + derivative*7;

        // Compute the actual motor settings.  We never set either motor
        // to a negative value.
        const int max = 150;
        if(power_difference > max)
                power_difference = max;
        if(power_difference < -max)
                power_difference = -max;

        if(power_difference < 0)
                set_motors(max+power_difference+20,max);
        else
                set_motors(max, max-power_difference+20);




                }

                if (j=='l')         {
                contador1=0;
// This is the "main loop" - it will run forever.
for (contador1=0;contador1<100;contador1++)
```

```
    {
delay_ms(8);

            // Get the position of the line.  Note that we *must* provide
            // the "sensors" argument to read_line() here, even though we
            // are not interested in the individual sensor readings.
            unsigned int position = read_line(sensors,IR_EMITTERS_ON);


            // The "proportional" term should be 0 when we are on the line.
            int proportional = ((int)position) - 2000;

            // Compute the derivative (change) and integral (sum) of the
            // position.
            int derivative = proportional - last_proportional;
            integral += proportional;

            // Remember the last position.
            last_proportional = proportional;

            // Compute the difference between the two motor power settings,
            // m1 - m2.  If this is a positive number the robot will turn
            // to the right.  If it is a negative number, the robot will
            // turn to the left, and the magnitude of the number determines
            // the sharpness of the turn.
            int power_difference =proportional/10+integral/8000+derivative*3/2;

            // Compute the actual motor settings.  We never set either motor
            // to a negative value.

    const int max =150;
    if(power_difference > max)
                    power_difference = max;
            if(power_difference < -max)
                    power_difference = -max;

            if(power_difference < 0)
                    set_motors(max+power_difference+5,max-contador1);
            else
                    set_motors(max-contador1, max-power_difference+5);


 }

 contador2=100;
```

```
  for (contador2==100;contador2>0;contador2=contador2-1)
     {
delay_ms(8);

            // Get the position of the line.  Note that we *must* provide
            // the "sensors" argument to read_line() here, even though we
            // are not interested in the individual sensor readings.
            unsigned int position = read_line(sensors,IR_EMITTERS_ON);


            // The "proportional" term should be 0 when we are on the line.
            int proportional = ((int)position) - 2000;

            // Compute the derivative (change) and integral (sum) of the
            // position.
            int derivative = proportional - last_proportional;
            integral += proportional;

            // Remember the last position.
            last_proportional = proportional;

            // Compute the difference between the two motor power settings,
            // m1 - m2.  If this is a positive number the robot will turn
            // to the right.  If it is a negative number, the robot will
            // turn to the left, and the magnitude of the number determines
            // the sharpness of the turn.
            int power_difference =proportional/10+integral/8000+derivative*3/2;

            // Compute the actual motor settings.  We never set either motor
            // to a negative value.

     const int max =150;
     if(power_difference > max)
                  power_difference = max;
            if(power_difference < -max)
                  power_difference = -max;

            if(power_difference < 0)
                  set_motors(max+power_difference+5,max-contador1);
            else
                  set_motors(max-contador1, max-power_difference+5);
                  }
                  }
            if (j=='u'){// Get the position of the line.  Note that we *must* provide
            // the "sensors" argument to read_line() here, even though we
```

```
            // are not interested in the individual sensor readings.
            unsigned int position = read_line(sensors,IR_EMITTERS_ON);

            // The "proportional" term should be 0 when we are on the line.
            int proportional = ((int)position) - 2000;

            // Compute the derivative (change) and integral (sum) of the
            // position.
            int derivative = proportional - last_proportional;
            integral += proportional;

            // Remember the last position.
            last_proportional = proportional;

            // Compute the difference between the two motor power settings,
            // m1 - m2.  If this is a positive number the robot will turn
            // to the right.  If it is a negative number, the robot will
            // turn to the left, and the magnitude of the number determines
            // the sharpness of the turn.
            int power_difference =proportional/5+integral/8000+derivative*7;

            // Compute the actual motor settings.  We never set either motor
            // to a negative value.


    if((power_difference<=(50+integral/8000+derivative*7))&(power_difference>=(-
50+integral/8000+derivative*7)))
                        set_motors(0,0);

     if(power_difference>(50+integral/8000+derivative*7))
                        set_motors(35,0);
       if(power_difference<(-50+integral/8000+derivative*7))
                         set_motors(0,35);




                              }
            if (j=='d'){ // Get the position of the line.  Note that we *must* provide
            // the "sensors" argument to read_line() here, even though we
            // are not interested in the individual sensor readings.
            unsigned int position = read_line(sensors,IR_EMITTERS_ON);

            // The "proportional" term should be 0 when we are on the line.
```

```
            int proportional = ((int)position) - 2000;

            // Compute the derivative (change) and integral (sum) of the
            // position.
            int derivative = proportional - last_proportional;
            integral += proportional;

            // Remember the last position.
            last_proportional = proportional;

            // Compute the difference between the two motor power settings,
            // m1 - m2.  If this is a positive number the robot will turn
            // to the right.  If it is a negative number, the robot will
            // turn to the left, and the magnitude of the number determines
            // the sharpness of the turn.
            int power_difference =proportional/5+integral/8000+derivative*7;

            // Compute the actual motor settings.  We never set either motor
            // to a negative value.
            const int max =80;

    if((power_difference<=(100+integral/8000+derivative*7))&(power_difference>=(-
100+integral/8000+derivative*7)))
                    set_motors(max,max);

     if(power_difference>(100+integral/8000+derivative*7))
                    set_motors(35,0);
       if(power_difference<(-100+integral/8000+derivative*7))
                     set_motors(0,35);
            }



                    }



}

}

/*****************************************************************/
```

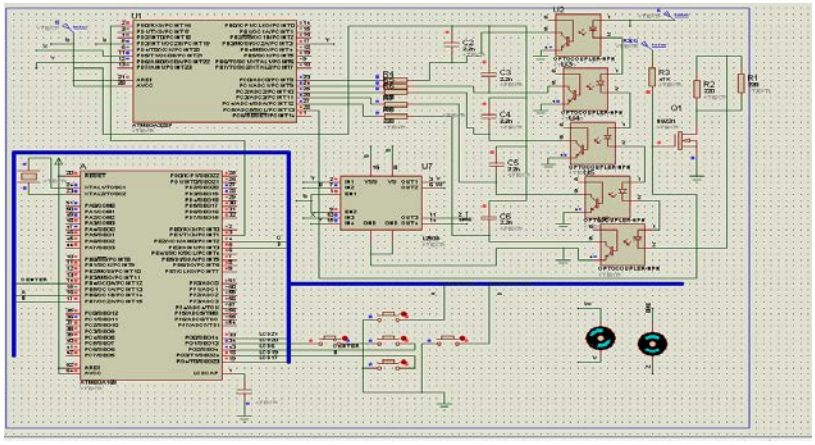
```
/* Initialize UART */
void InitUART (unsigned int ubrr)
{
     UBRR0H = (unsigned char)(ubrr>>8);
     UBRR0L = (unsigned char)ubrr;
       UCSR0A = (0<<U2X0);



  /* Enable UART receiver  */
  UCSR0B = (1 << RXEN0) ;

  /* set to 8 data bits, 1 stop bit */
  UCSR0C = (0<<USBS0) | (3 << UCSZ00);

}

/* Read and write functions */
unsigned char ReceiveByte (void)
{
  /* Wait for incomming data */
  while (!(UCSR0A & (1 << RXC0))){
  return 'w';}

  /* Return the data */
  return UDR0;
}



/*****************************************************************/
```

CAPÍTULO 4

4. SIMULACIÓN Y PRUEBAS

4.1 Simulación en Proteus

Gracias a la ayuda de la herramienta de simulación Proteus, se logró mostrar el funcionamiento básico de nuestro proyecto. En la primera figura 4-1 se muestra el esquemático simplificado del robot pololu con unos motores DC e intentando simular a los dos micromotores del robot, ambos van conectados a un driver que en este caso es el L293D que recibe las señales del microcontrolador. Dependiendo de las señales de los sensores se da el movimiento de los motores que mueven al seguidor de línea. Como se muestra en la Figura 4-1 que se puede observar a continuación.



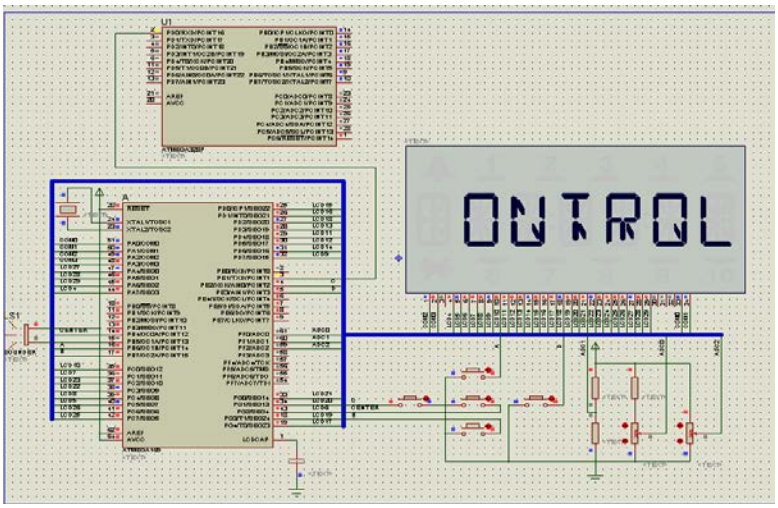


FIGURA 4-1: SIMULACIÓN DEL PROYECTO

La simulación de nuestro robot Pololu 3Pi lo hemos implementado con el uso de varios elementos como motores para representar el movimientos conectados a una fuente de 4 baterías triple A presionamos los push buttons para simular la señal que emite la tarjeta Butterfly.

4.2 Imágenes del proyecto final

En la Figura 4-2 se muestra la tarjeta Butterfly ensamblada en una tarjeta electrónica la cual tiene la batería de 9 voltios con un regulador de 5 voltios para alimentar al dispositivo de radiofrecuencia, para la tarjeta Butterfly se usa una pila de 3voltios independiente. Como se muestra en la figura 4-2.

FIGURA 4-2: Imagen del proyecto funcionando

El inicio del proyecto empieza mostrando en la pantalla LCD según se presione el joystick la orden que envía al Pololu 3Pi proyecto. Además nos pide presionar la tecla B del POLOLU para que los motores y se sensores puedan inicializarse, nuevamente se presiona B para que el Pololu 3Pi entre en modo de recepción de la señal de radiofrecuencia. Como se muestra en la Figura 4-3.

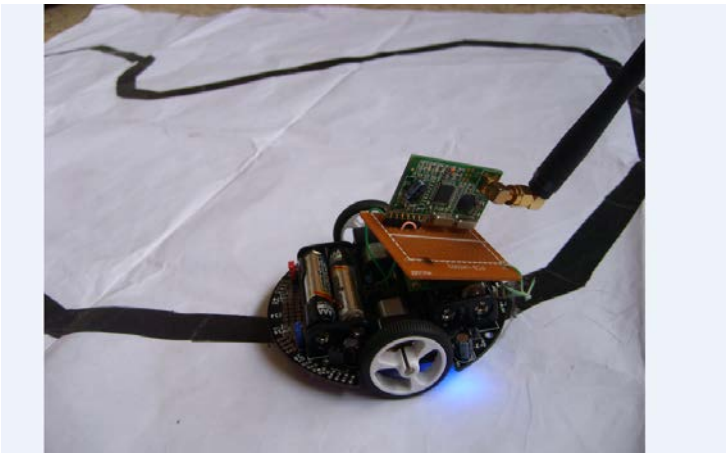


FIGURA 4-3: Pololu Seguidor de Línea.

Al momento del envió de la señal por parte de butterfly se podrá observar brillar unos leds que tienen los módulos de radiofrecuencia, estos leds nos indicarán que las señales están siendo enviadas y recibidas respectivamente, cuando el robot reciba las ordenes este ejecutara la programación asignada para sus respectivas ordenes con los diferentes lazos PID.

CONCLUSIONES Y RECOMENDACIONES

Las conclusiones son:

1. Nuestro proyecto realizado es un prototipo para mostrar uno de los muchos usos de los lazos de control PID, en nuestro caso mediante la detección de la señal de posición por sensores infrarrojos, del robot Pololu 3Pi seguidor de línea.

2. La optimización del lazo de control PID permite un incremento importante de velocidad y el control de la velocidad en diferentes tipos de tramos de la pista, tanto en las rectas como en los curvas, empleamos la debida operación para mantener una velocidad deseada en el tramo en el que se encuentre.

3. Las funciones que gobiernan el movimiento de los motores, sensores infrarrojos son de sencilla aplicación. Las librerías aplicadas pueden presentarnos un buen seguimiento de los valores que han sido obtenidos por los sensores de energía y de posición en la pista, necesarios para el buen desenvolvimiento del seguidor de línea.

4. El diseño y el prototipo se lo pueden tomar para comparar con otros modelos de robot seguidores de línea y así detectar diferencias, ventajas, desventajas para posteriores estudios e implementación de mejoras, reconociendo que el mundo está en constante cambio.

Las recomendaciones son:

1. Para el uso de los sensores infrarrojos se debe diseñar una pista donde claramente se distinga de color negro el camino a seguir y la plataforma de color blanco,debido a que los sensores son muy sensibles a cambios de colores y nos puede ocasionar una mala lectura lo cual hace salir de pista a nuestro robot.
2. Debemos tener en cuenta la inercia del robot con su antena, debido a que la velocidad en curvas muy pronunciadas, es peligrosa para el robot.
3. Es recomendable usar baterías del tipo recargables ya que nuestro robot demanda un consumo de energía moderado. Si usaramos las pilas normales, esto representaría un alto costo a largo plazo y mucho mayor la contaminación del medio ambiente debido a los químicos que estas contienen.
4. Debemos tener en cuenta que la tarjeta butterfly no puede proveer de mucha energía al módulo de radio frecuencia y fue necesario implementar un circuito para suministrarle energía a través de una fuente al dispositivo de transmision de datos.

ANEXOS

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1K Bytes EEPROM
  - 512/1K/1K/2K Bytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 4 MHz@1.8 - 5.5V, 0 - 10 MHz@2.7 - 5.5V, 0 - 20 MHz @ 4.5 - 5.5V
- Power Consumption at 1 MHz, 1.8V, 25°C
  - Active Mode: 0.2 mA
  - Power-down Mode: 0.1 µA
  - Power-save Mode: 0.75 µA (Including 32 kHz RTC)

8-bit AVR® Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash

ATmega48A
ATmega48PA
ATmega88A
ATmega88PA
ATmega168A
ATmega168PA
ATmega328
ATmega328P

Summary

Rev. 8271CS–AVR–08/10

1. Pin Configurations

Figure 1-1. Pinout ATmega48A/48PA/88A/88PA/168A/168PA/328/328P

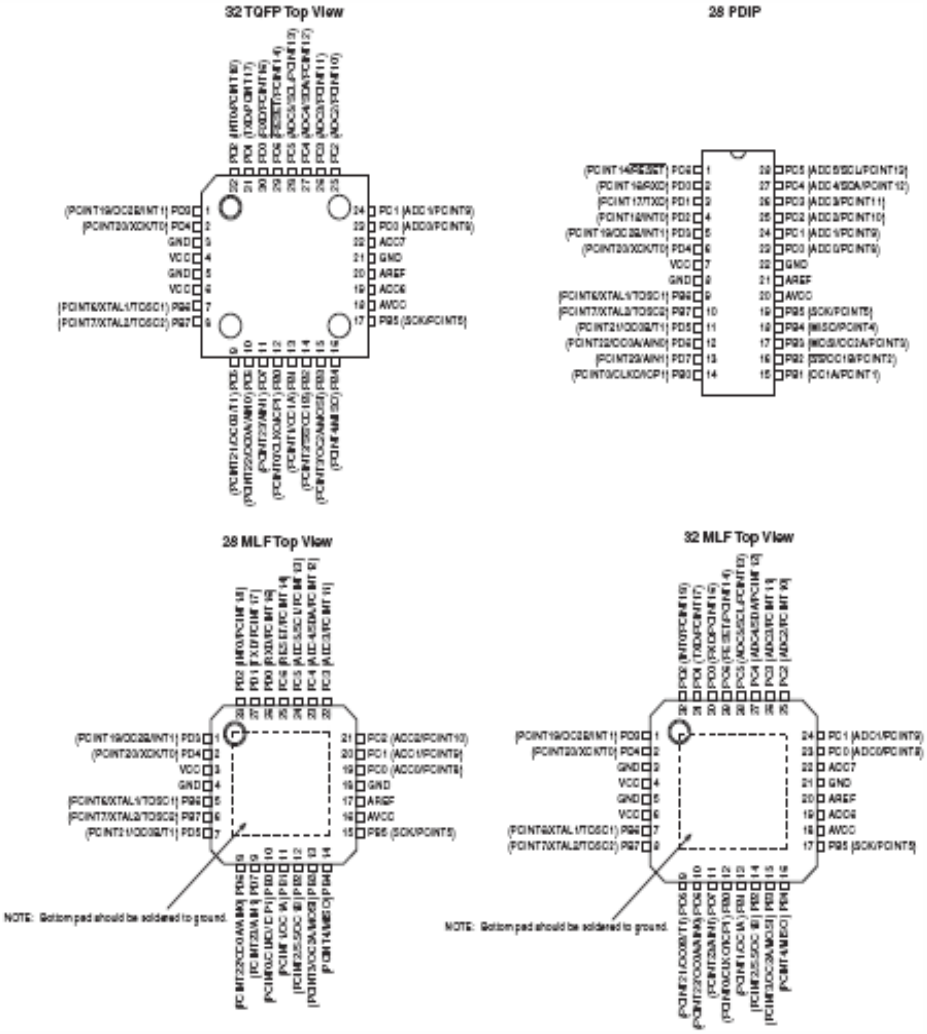


Table 1-1. 32UFBGA - Pinout ATmega48A/48PA/88A/88PA/168A/168PA

	1	2	3	4	5	6
A	PD2	PD1	PC6	PC4	PC2	PC1
B	PD3	PD4	PD0	PC5	PC3	PC0
C	GND	GND			ADC7	GND
D	VDD	VDD			AREF	ADC6
E	PB6	PD6	PB0	PB2	AVDD	PB5
F	PB7	PD5	PD7	PB1	PB3	PB4

1.1 Pin Descriptions

1.1.1 VCC

Digital supply voltage.

1.1.2 GND

Ground.

1.1.3 Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7...6 is used as TOSC2...1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

The various special features of Port B are elaborated in and "System Clock and Clock Options" on page 26.

1.1.4 Port C (PC5:0)

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5...0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

1.1.5 PC6/$\overline{\text{RESET}}$

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given in Table 28-12 on page 323. Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in "Alternate Functions of Port C" on page 86.

1.1.6 Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

The various special features of Port D are elaborated in "Alternate Functions of Port D" on page 89.

1.1.7 AV_{CC}

AV_{CC} is the supply voltage pin for the A/D Converter, PC3:0, and ADC7:6. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter. Note that PC6...4 use digital supply voltage, V_{CC}.

1.1.8 AREF

AREF is the analog reference pin for the A/D Converter.

1.1.9 ADC7:6 (TQFP and QFN/MLF Package Only)

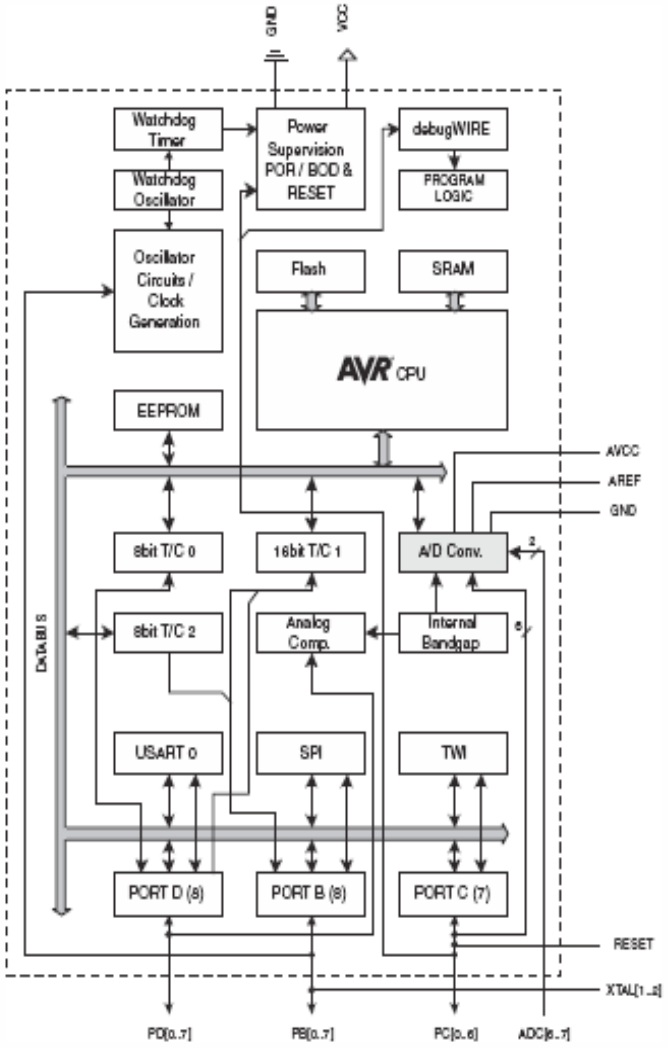
In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

2. Overview

The ATmega48A/48PA/88A/88PA/168A/168PA/328/328P is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega48A/48PA/88A/88PA/168A/168PA/328/328P achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

2.1 Block Diagram

Figure 2-1. Block Diagram



The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent

5. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF − Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 − Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd − 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP[1]	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL[1]	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	If (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd − Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd − Rr − C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd − K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	If (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	If (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	If (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	If (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	If (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	If (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	If (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	If (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	If (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	If (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	If (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	If (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	If (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	If (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	If (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	If (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	If (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	If (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	If (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	If (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	If (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	If (V = 0) then PC ← PC + k + 1	None	1/2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0...6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3...0)←Rd(7...4),Rd(7...4)←Rd(3...0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

Note: 1. These instructions are only available in ATmega168PA and ATmega328P.

Section 1

Introduction

The AVR Butterfly evaluation kit is designed to demonstrate the benefits and key features of the AVR microcontrollers. It is a stand alone microprocessor module that can be used in numerous applications:

- The AVR architecture in general and the ATmega169 in particular
- Low power design
- The MLF package type
- Peripherals
  - LCD controller
  - Memories
    - Flash, EEPROM, SRAM, external DataFlash
  - Communication interfaces
    - UART, SPI, USI
  - Programming methods
    - Selfprogramming/ Bootloader, SPI, Parallel, JTAG
  - Analog to Digital Converter (ADC)
  - Timers/Counters
    - Real Time Clock (RTC)
    - Pulse Width Modulation (PWM)

It also serve as a development kit for the ATmega169, and can be used as a module in other products.

Figure 1-1. AVR Butterfly

1.1 Resources Available on the AVR Butterfly Kit

The following resources are available on the Butterfly kit.

- ATmega169 (MLF-package)
- LCD-on-glass display with 120 segments, for demonstrating the ATmega169 LCD controller.
- Joystick, 4-directions with centre push, as user input
- Piezo element, to play sounds
- 32kHz Xtal for the RTC
- 4 Mbit DataFlash, for data storage
- RS-232 level-converter, for communicating with off-board units
- Negative Temperature Coefficient (NTC) thermistor, to measure temperature
- Light Dependent Resistor (LDR), to measure light intenisty
- 3V button cell battery (600mAh) to provide operating power
- JTAG emulation, for debugging
- USI-interface, for additional communication interface
- Supported by AVR Studio 4.
- Pre-programmed with a demonstration application, including bootloader
- No external hardware is required to reprogram the AVR Butterfly

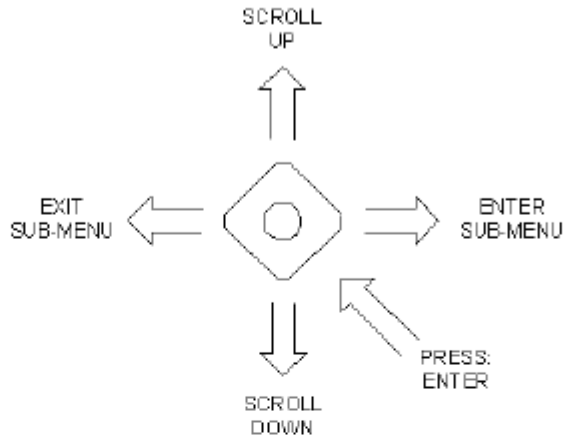
The ATmega169 in the kit controls the external peripherals, and can also be used to do voltage readings from 0 to 5 volts. The kit can be reprogrammed a number of different ways including serial programming through the JTAG port. Most users will prefer to use the preloaded bootloader with AVR Studio to download new code.

For more information about the ATmega169, see the datasheet at www.atmel.com.

2.2 Joystick Input

To operate the AVR Butterfly a joystick is used as user input. It operates in five directions, including center-push, see *Figure 2-1*.

Figure 2-1. Joystick Input

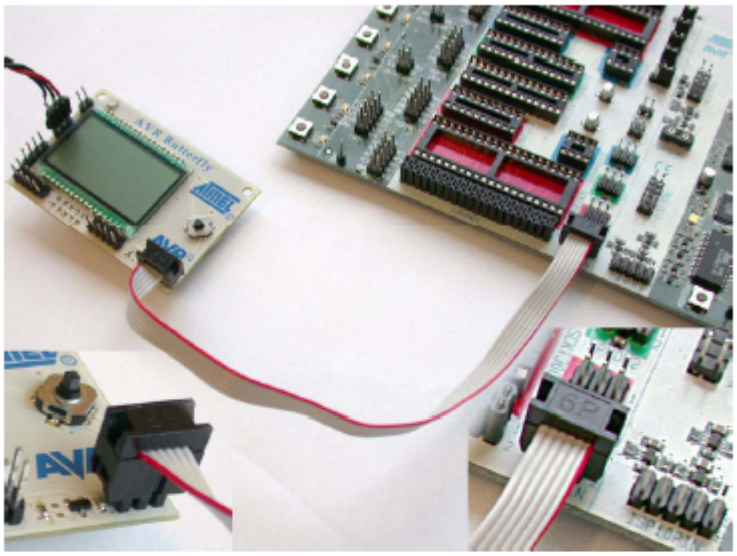


Using the joystick one can move around in the menu shown in *Figure 2-2*, and edit values, entering name, etc. Here are examples on how to enter your name.

2.2.1 Entering Your Name Using the Joystick:

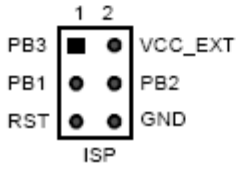
1. Press the joystick up ("SCROLL UP") to wake the AVR Butterfly. If "AVR BUTTERFLY" is notscrolling over the display, press the joystick to the left ("EXIT SUB-MENU") until it does.
2. Press the joystick down ("SCROLL DOWN") three times, so the string "NAME" is displayed.
3. Press the joystick to the right ("ENTER SUB-MENU"). If this is the first time a name is entered, the string "ENTER NAME" will be displayed, otherwise the name already entered will be displayed and you have to press the joystick to the right ("ENTER SUB-MENU") once more.
4. When "ENTER NAME" is displayed press center push ("ENTER"). If this is the first time you enter a name, the character "A" should be blinking in the right side in the display, otherwise the last character of the already entered name will blink.
5. Press the joystick up ("SCROLL UP") or down ("SCROLL DOWN") to get to the wanted character. Press the joystick to the right ("ENTER SUB-MENU") to add a new character or press the joystick to the left ("EXIT SUB-MENU") to remove a character.
6. When you have got all the characters, up to maximum 25, press center push ("ENTER") to *save* this name. The name will now be displayed in the display. If the name is more than 6 characters long it will scroll over the display, otherwise it will be displayed static.

Figure 3-2. In-System Programming



To program the ATmega169 using ISP Programming mode, connect a 6-wire cable between the ISP6PIN connector on the STK500 board and J403 the ISP connector on the AVR Butterfly as shown in *Figure 3-2*. This device can be programmed using the Serial Programming mode in the AVR Studio4 STK500 software. Instead of soldering in a ISP-header, one can make contact just by pressing the header to the footprint. Make sure that pin 1 on the STK500 match with pin 1 on the AVR Butterfly. See *Figure 3-3* for the pinout of the ISP Connector.

Figure 3-3. ISP Connector, J403



BIBLIOGRAFÍA

1. GONZALEZ OSCAR, COMO EMPEZAR CON EL ROBOT POLOLU, http://blog.bricogeek.com/noticias/robotica/como-empezar-con-el-sigue-lineas-3pi-de-pololu/, 06 de Enero del 2011.

2. 2001-2009 POLOLU CORPORATION, SAMPLE PROJECT POLOLU, http://www.sparkfun.com/datasheets/Robotics/3pi_wall_follower.pdf, 16 de febrero 2011.

3. HAJI REZA, VIDEOS DEL ROBOT POLOLU 3PI FUNCIONAMINETO, http://code.google.com/p/quickanddirty/wiki/Pololu3piRobot , 22 de febrero 2011.

4. HOSSEIN EGHBALI, DESCARGA DEL AVR STUDIO 4.0, http://www.atmel.com/microsite/avr_studio_5/default.asp?source=cms&icn=hmap1-AVR_STUDIO&ici=apr_2011, 23 de febrero 2011.

5. REEL SIZZLE, PRODUCTOS ATMEL, http://www.atmel.com/products/AT91/, 23 febrero 2011.

6. REES EVER, DESCRIPCIÓN DEL MICROCONTROLADOR ATMEL BUTTERFLY, http://store.atmel.com/PartDetail.aspx?q=p:10500051, 24 febrero 2011.