Taller de proyecto final
Robot recolector de residuos
Placa módulo genérico

20 de abril de 2010

Resumen

En el presente se establecen las especificaciones para la placa del módulo genérico. Se expone el circuito de la placa, explica funcionamiento y se muestran posibles usos.

Palabras Clave: *Robot, residuos, protocolo, serial, rs-232, daisy chain.*

1. Introducción

Durante el diseño de las placas controladoras surgió la necesidad de establecer un módulo común de comunicación y una base de prueba para los testeos con los distintos sensores y periféricos que se utilizarían en el robot. Crear una placa que resuelva dichas cuestiones fue de gran ayuda y aceleró en gran medida las pruebas y diseño de las placas que le sucedieron.

En las siguientes secciones se detalla cada uno de los aspectos tenidos en cuenta para crear una placa genérica para el desarrollo del robot.

2. Microcontrolador

El microcontrolador elegido para la placa es el 16F88 de Microchip. Cuenta con una memoria *FLASH* para 4096 instrucciones de programa, una memoria *RAM* de 368 bytes y una memoria *EEPROM* de 256 bytes. Cuenta con un reducido set de instrucciones básicas todas con el mismo tiempo de ejecución. En la subsección 2.1 se listan algunos de los principales periféricos incluidos en el microcontrolador. Se utiliza con un cristal externo de 20MHz como clock.

Para la carga y debug del firmware específico para cada placa se utiliza el programador *ICD2*, como se explica en la sección 9.

2.1. Periféricos

El microcontrolador 16F88 cuenta con 2 puertos de 8 entradas y salidas cada uno de tipo TTL y CMOS. Cada pin se encuentra multiplexado con uno o más periféricos internos.

2.1.1. Timers

Cuenta con 3 timers o contadores.

El *TMR0* es de 8 bits y contiene un *preescaler* de 8bits, es usado como WDT. También puede ser utilizado como contador externo por el pin *RA4*.

El *TMR1* es de 16 bits y contiene un *preescaler* de 2bits. Puede ser utilizado como contador externo por el pin *RB6* o con un cristal externo conectado a los pines *RB6* y *RB7*.

El *TMR2* es de 8 bits, contiene un *preescaler* de 2 bits y contiene un *postscaler* de 4 bits. Es de vital importancia para el módulo de PWM por hardware.

2.1.2. ADC

Cuenta con un conversor analógico digital de 8 o 10 bits multiplexado en 7 canales, 5 canales en el puerto A y 2 en el puerto B. Es posible definir voltajes de referencia mediante ciertos pines o usar valores internos de referencia como *Vcc* y *GND*.

2.1.3. PWM

Cuenta con un módulo de generación de un PWM por hardware de 10 bits de resolución con el ciclo y período configurable mediante el *TMR2*

2.1.4. AUSART

Cuenta con un módulo de UART para comunicaciíon sincrónica o asincrónica utilizado para la implementación del daisy chain por RS-232.

2.1.5. Otros

Para mayor información respecto a los periféricos o configuración del microcontrolador, se recomienda revisar las hojas de datos directo del fabricante.

3. Comunicación

El protocolo de comuncación está formado por paquetes que representan un pedido de información o comando que debe ser ejecutado en el destino. Ver documentación del protocolo para mayor información.

La comunicación está basada en el método *Daisy chain* (patente US20090316836A1). La cadena está formada por las placas controladoras, las cuales se comunican entre ellas retransmitiendo cada paquete hacia adenlante.

Como parte de la configuración de la placa, existe un switch que determina el tipo de eslabón de la placa (modo *LINK*), si es un nodo intermedio o la punta de la cadena (modo *LAST*).

En la figura 2 se muestran los conectores utilizados. En los cuadros 1 y 2 se especifica el conexionado entre placas y contra el controlador principal.

Se recomienda el uso de cable de par trenzado o mallado para disminuir la interferencia.

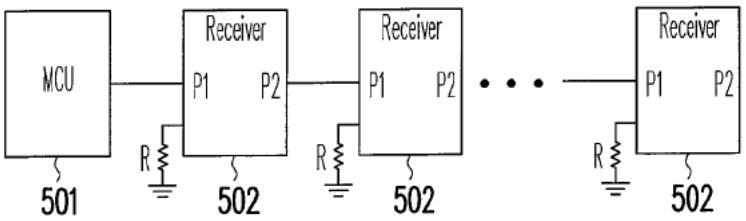

Figura 1: Diagrama general del método daisy chain

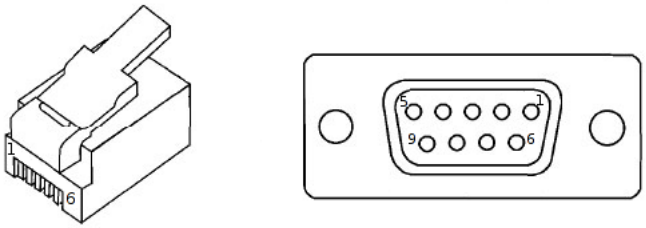

Figura 2: Conectores RJ11 y DB9 utilizados para la comuncaciíon

Función	Conector RJ11	Conector RJ11	Función
Serial RX	2	2	Serial TX
Serial TX	3	3	Serial RX
No conectado	4	4	No conectado
GND	5	5	GND

Cuadro 1: Conexionado entre placas en modo Link

Función	Conector RJ11	Conector DB9	Función
Serial RX	2	3	Serial TX
Serial TX	3	2	Serial RX
No conectado	4	4	Shield
GND	5	5	GND

Cuadro 2: Conexionado entre placa y la PC

Figura 3: Bornera de 3 pines para la alimentación de la placa

Pin	Voltaje
1	GND
2	5v
3	7v a 12v

Cuadro 3: Alimentación de la lógica

4. Alimentación

La alimentación principal de la placa es 7 a 20 voltios, con la posibilidad de alimentarla directamente con 5 voltios por uno de los pines del conector. En la figura 3 se muestra la bornera y en el cuadro 3 el pinout de la misma.

La regulación interna de voltaje realiza por medio de un regulador 7805 corriente máxima de 1A.

5. Configuración

El header de programación *P1* se utiliza para conectar la placa con el programador y debuguer de código *ICD2* como se explica en la sección 9.

La fila de pines *P2* exporta todos los pines con funciones dentro del microcontrolador, para realizar conexiones con periféricos de prueba. Los headers *P3* y *P4* son jumpers que vinculan los pines *RA1* y *RA4* del microcontrolador los leds 1 y 2 respectivamente.

El switch *S3* como se explica en la sección 3, se utiliza para determinar el papel de la placa dentro de la cadena (modo *LINK* o modo *LAST*). El switch *S2* se utiliza para asociar los pines del microcontrolador con los canales de clock y data del header de programación (modo *ICD2*) o con los pines del header *P2*.

6. Posibles usos

Desde el principio, la placa fue pensada como una placa es el testeo de la comunicación y conexionado de nuevos periféricos y sensores. Aunque también puede ser usada como placa entrenadora para proveer un punto de entrada a nuevos integrantes del proyecto en la programación de placas y nuevos sensores.

Ultilizada mayormente en la etapa de diseño, disminuye los errores y posibilita el concentrarse en las partes nuevas con las que se está trabajando.

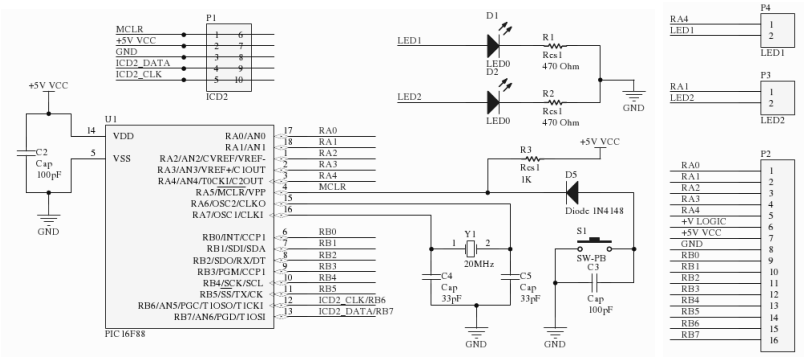

Figura 4: Microcontrolador y headers

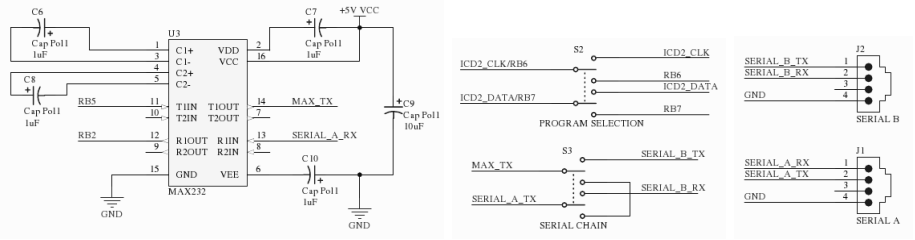

Figura 5: Comunicación, switch de modo y conectores de entrada y salida

7. Esquemático

En la figura 4 se detalla el esquemático del microcontrolador y el conexionado con los headers.

En la figura 5 se muestra el módulo de comunicación y el conexionado con los conectores para conformar la cadena *Daisy chain*.

En la figura 6 se muestra la fuente de alimentación y bornera.

8. Circuito

En la figura 7 se muestra la máscara de componentes de la placa. En la figura 8 se muestran ambas capas de la placa.

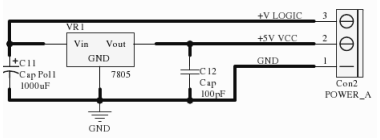

Figura 6: Fuente de alimentación

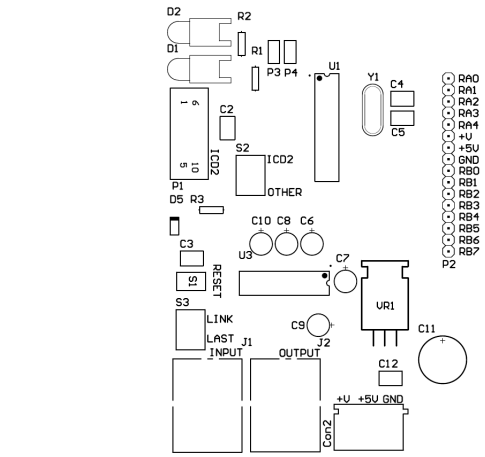

Figura 7: Máscara de componentes

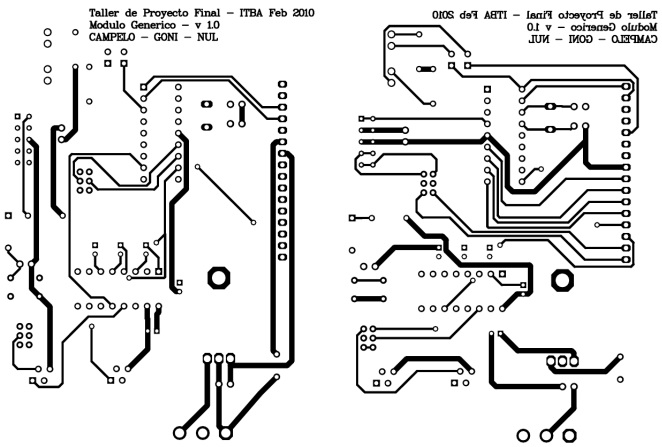

Figura 8: Capas superior e inferior de la placa

9. El programador

El programador utilizado es el modelo *ICD2* de la empresa *Microchip*. Provee una interfaz tanto para la carga y descarga de firmware al microcontrolador, sino que también permite debuguear dicho código.

La IDE de programación utilizada es *Microchip MPLAB* debido a su completa integración con el producto.

El lenguaje de programación utilizado es *C* y el compilador elegido es *CCS PCM V4.023*.

10. Código básico

Debido a que la placa fue pensada como punto de partida, se provee del código utilizado como base para la programación de las otras placas controladoras.

```
#define CARD_GROUP      MOTOR_DC        // Ver protocol.h
#define CARD_ID         0               // Valor entre 0 y E

// Descripcion de la placa
#define DESC            "PLACA GENERICA - 1.0" // Maximo DATA_SIZE bytes

/* Modulo Generico - main.c
 * PIC16F88 - MAX232 - GENERICO
 *
 *                               PIC16F88
 *                .-------------------------------------.
 *               -|RA2/AN2/CVREF/VREF           RA1/AN1|- LED
 *               -|RA3/AN3/VREF+/C1OUT          RA0/AN0|-
 *           LED -|RA4/AN4/T0CKI/C2OUT    RA7/OSC1/CLKI|- XT CLOCK pin1, 27pF to GND
 * RST/ICD2:MCLR -|RA5/MCLR/VPP           RA6/OSC2/CLKO|- XT CLOCK pin2, 27pF to GND
 *           GND -|VSS                         VDD|- +5v
 *               -|RB0/INT/CCP1       RB7/AN6/PGD/T1OSI|- ICD2:PGD/
 *               -|RB1/SDI/SDA  RB6/AN5/PGC/T1OSO/T1CKI|- ICD2:PGC/
 *  MAX232:R1OUT -|RB2/SDO/RX/DT           RB5/SS/TX/CK|- MAX232:T1IN
 *               -|RB3/PGM/CCP1             RB4/SCK/SCL|-
 *                '-------------------------------------'
 *
 */

#include <16F88.h>
#DEVICE ADC = 10
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay (clock=20000000)

#use rs232(BAUD=115200,PARITY=N,XMIT=PIN_B5,RCV=PIN_B2,BITS=8,ERRORS,TIMEOUT=1,STOP=1,UART1)
#use fast_io(A)
#use fast_io(B)

#byte porta=0x05
#byte portb=0x06

// Led
#bit led1=porta.1
#bit led2=porta.4

// MAX232
#bit tx=portb.5
#bit rx=portb.2

#include <../../protocolo/src/protocol.c>
/*
** Variables definidas en protocol.c
```

```
short reset; // Variable para hacer el reset
short crcOK; // Informa si el CRC del paquete parseado fue correcto
short sendResponse; // Informa que no debe mandarse la respuesta automatica

char buffer[MAX_BUFFER_SIZE];   // Buffer de recepcion de comandos
int buffer_write;               // Indice de escritura
int buffer_read;                // Indice de lectura
int data_length;                // Largo de los datos en el buffer

struct command_t command;       // Comando parseado
struct command_t response;      // Respuesta

** Implementar las siguientes funciones (usadas por el protocolo)

void init(); // Inicializa puertos y variables
void doCommand(struct command_t * cmd); // Examina y ejecula el comando

***/

void init()
{
        // Inicializa puertos
        set_tris_a(0b11100101);
        set_tris_b(0b11100110);

        // Variable para hacer el reset
        reset = false;

        return;
}

void main()
{
        // Placa Generica - Implementacion del protocolo
        init();

        // Init del protocol
        initProtocol();

        // FOREVER
        while(true)
        {
                // Hace sus funciones...

                // Protocolo
                runProtocol(&command);
        }

        return;
}

/* Verifica que el comando sea valido y lo ejecuta */
void doCommand(struct command_t * cmd)
{
        int crc, i, len;

        // Calculo del CRC
        crc = generate_8bit_crc((char *)cmd, cmd->len, CRC_PATTERN);

        // CRC ok?
        if (cmd->crc != crc)
        {
                // Creo respuesta de error
                response.len = MIN_LENGTH + cmd->len + 2 + 1;
                response.to = cmd->from;
                response.from = THIS_CARD;
                response.cmd = COMMON_ERROR;
                response.data[0] = 0x00;
                // Agrego el paquete que contiene el error de CRC
                response.data[1] = cmd->len;
                response.data[2] = cmd->to;
                response.data[3] = cmd->from;
                response.data[4] = cmd->cmd;
                // Campo data
```

```
                len = cmd->len - MIN_LENGTH;
                for (i = 0; i < len; i++)
                        response.data[5 + i] = (cmd->data)[i];
                // CRC erroneo
                response.data[5 + len] = cmd->crc;
                // CRC esperado
                response.data[5 + len + 1] = crc;
                // CRC de la respuesta
                response.crc = generate_8bit_crc((char *)(&response), response.len, CRC_PATTERN);

                crcOK = false;
                return;
        }

        crcOK = true;

        // Minimo todos setean esto
        response.len = MIN_LENGTH;
        response.to = cmd->from & 0x77;
        response.from = THIS_CARD;
        response.cmd = cmd->cmd | 0x80;

        switch (cmd->cmd)
        {
                // Comandos comunes
                case COMMON_INIT:
                        init();
                        // Enviar la descripcion de la placa en texto plano
                        strcpy(response.data, DESC);
                        response.len += strlen(response.data);
                break;
                case COMMON_RESET:
                        // Enviar la descripcion de la placa en texto plano
                        strcpy(response.data, DESC);
                        response.len += strlen(response.data);
                        // Reset!
                        reset = true;
                break;
                case COMMON_PING:
                        // No hace falta hacer mas nada
                break;
                case COMMON_ERROR:
                        // Por ahora se ignora el comando
                break;

                /* Comandos especificos */

                case 0x40:
                        /*
                        :DATO:
                        -
                        :RESP:
                        -
                        */
                break;

                default:
                        response.len++;
                        response.cmd = COMMON_ERROR;
                        response.data[0] = 0x01; // Comando desconocido
                break;
        }

        // CRC de la respuesta
        response.crc = generate_8bit_crc((char *)(&response), response.len, CRC_PATTERN);

        return;
}
```

El código incluye al archivo *protocolo.c* que se muestra a continuación.

```
#include <../../protocolo/src/protocol.h>

short reset;
```

```
short crcOK;
short sendResponse;

char buffer[MAX_BUFFER_SIZE];
int buffer_write;
int buffer_read;
int data_length;

// Comando parseado
struct command_t command;
// Respuesta
struct command_t response;

// Interrupcion del RS232
#INT_RDA
void RS232()
{
        disable_interrupts(INT_RDA);
        // Un nuevo dato...
        buffer[buffer_write++] = getc();
        data_length++;
        if (buffer_write == MAX_BUFFER_SIZE)
                buffer_write -= MAX_BUFFER_SIZE;
        enable_interrupts(INT_RDA);
        return;
}

/* Envia los datos por el pto serial */
void initProtocol()
{
        // Variables de comunicacion
        buffer_write = 0;
        buffer_read = 0;
        data_length = 0;
        crcOK = false;

        // Interrupcion Rcv
        enable_interrupts(INT_RDA);

        // Habilito las interrupciones
        enable_interrupts(GLOBAL);
}

/* Envia los datos por el pto serial */
void send(struct command_t * cmd)
{
        int i, len;

        len = cmd->len - 4;
        putc(cmd->len);
        putc(cmd->to);
        putc(cmd->from);
        putc(cmd->cmd);

        for (i = 0; i < len; i++)
        {
                putc((cmd->data)[i]);
        }

        // Enviar el CRC
        putc(cmd->crc);

        return;
}

void runProtocol(struct command_t * cmd)
{
        // Analiza el buffer
        if (buffer[buffer_read] < data_length)
        {
                data_length -= buffer[buffer_read] + 1;

                cmd->len = buffer[buffer_read++];
```

```
                if (buffer_read == MAX_BUFFER_SIZE)
                        buffer_read = 0;

                cmd->to = buffer[buffer_read++];

                if (buffer_read == MAX_BUFFER_SIZE)
                        buffer_read = 0;

                cmd->from = buffer[buffer_read++];

                if (buffer_read == MAX_BUFFER_SIZE)
                        buffer_read = 0;

                cmd->cmd = buffer[buffer_read++];

                if (buffer_read == MAX_BUFFER_SIZE)
                        buffer_read = 0;

                // Obtiene el campo DATA
                if ((buffer_read + cmd->len - MIN_LENGTH) > MAX_BUFFER_SIZE)
                {
                        // DATA esta partido en el buffer ciclico
                        memcpy(cmd->data, buffer + buffer_read, MAX_BUFFER_SIZE - buffer_read);
                        memcpy(cmd->data + MAX_BUFFER_SIZE - buffer_read, buffer,
                                cmd->len - MIN_LENGTH - MAX_BUFFER_SIZE + buffer_read);
                } else {
                        // DATA esta continuo
                        memcpy(cmd->data, buffer + buffer_read, cmd->len - MIN_LENGTH);
                }

                buffer_read += cmd->len - MIN_LENGTH;
                if (buffer_read >= MAX_BUFFER_SIZE)
                        buffer_read -= MAX_BUFFER_SIZE;

                cmd->crc = buffer[buffer_read++];

                if (buffer_read == MAX_BUFFER_SIZE)
                        buffer_read = 0;

                sendResponse = true;

                // Soy el destinatario?
                if (cmd->to == THIS_CARD)
                {
                        // Ejecuta el comando
                        doCommand(cmd);
                } else // Es broadcast?
                        if ((cmd->to & 0xF0) == 0xF0)
                {
                        // Ejecuta el comando
                        doCommand(cmd);

                        if (crcOK == true)
                        {
                                // Envia la respuesta
                                send(&response);
                                // Envia nuevamente el comando recibido
                                response = *cmd;
                        }
                } else // Es broadcast para mi grupo?
                        if (((cmd->to & 0x0F) == 0x0F) &&
                                ((cmd->to & 0xF0) == THIS_GROUP))
                {
                        // Ejecuta el comando
                        doCommand(cmd);
#ifdef RESEND_GROUP_BROADCAST
                        if (crcOK == true)
                        {
                                // Envia la respuesta
                                send(&response);
                                // Envia nuevamente el comando recibido
                                response = *cmd;
                        }
#endif
```

```
                } else {
                        response = *cmd;
                }

                // Envia la respuesta?
                if (sendResponse == true)
                {
                        send(&response);
                }

        }

        // Reset del micro
        if (reset == true)
        {
                reset_cpu();
        }

}

int generate_8bit_crc(char* data, int length, int pattern)
{
        // TODO: reemplazar por el crc?

        int crc_byte, i;

        crc_byte = data[0];

        for (i = 1; i < length; i++)
                crc_byte ^= data[i];

        return crc_byte;
}
```