Universidad del Valle de Guatemala

Facultad de Ciencias y Humanidades

Departamento de Ciencias de la Computación

CREACIÓN DE UN SISTEMA INMERSO PARA CONTROLAR LA MICROCOMPUTADORA DE LEGO® MINDSTORMS™

CARLOS LUIS RENDÓN M.

Trabajo de Graduación presentado para optar al grado académico de Maestría en Tecnología y Ciencias de la Computación

Guatemala, 2004

CREACIÓN DE UN SISTEMA INMERSO PARA CONTROLAR LA MICROCOMPUTADORA DE LEGO® MINDSTORMS™

Universidad del Valle de Guatemala

Facultad de Ciencias y Humanidades

Departamento de Ciencias de la Computación

CREACIÓN DE UN SISTEMA INMERSO PARA CONTROLAR LA MICROCOMPUTADORA DE LEGO® MINDSTORMS™

CARLOS LUIS RENDÓN M.

Trabajo de Graduación presentado para optar al grado académico de Maestría en Tecnología y Ciencias de la Computación

Guatemala, 2004

TABLA DE CONTENIDO

LISTADO DE CUADROS

Cuadro

LISTADO DE GRÁFICOS

Gráfico

RESUMEN EJECUTIVO

En la actualidad existe la necesidad de proveer un ambiente de pruebas a nivel universitario que facilite la enseñanza de cursos como Robótica, Sistemas Inmersos y Sistemas Operativos, entre otros. Muchas veces esto no es posible por la costos muy altos, especialmente en América Latina.

Desde 1998 existe una alternativa para estos casos: El conjunto robótico de Lego® Mindstorms™. A primera vista es un conjunto común y corriente de Lego®, sin embargo contiene una pieza que lo distingue de cualquier creación previa de la compañía: el RCX™ (por sus siglas en inglés Robot Command Explorer).

Hay dos formas de “darle vida” a un robot. La primera es programarlo para que sea independiente de la PC por medio de un programa que el RCX™ ejecuta solo. Y la segunda es controlarlo en modo “maestro / esclavo”, en el cual la PC o una entidad funciona como maestro y el RCX™ ejecuta las instrucciones de acuerdo a un protocolo establecido por el fabricante.

Este trabajo de graduación detalla cómo construir un sistema inmerso que enviará instrucciones a la microcomputadora de Lego®, el RCX™, en forma inalámbrica, a través de un enlace infrarrojo, sin depender de una computadora personal.

I. INTRODUCCIÓN

El proyecto consiste en desarrollar un sistema inmerso (ver figura I-1) para poder controlar la microcomputadora de Lego®, el RCX™. Esto resuelve uno de los principales problemas cuando se desarrolla un sistema inmerso hoy día: **la movilidad**. Una aplicación inmersa debe poder ser administrada remotamente y eliminar la necesidad del uso de una computadora, ya que esto incrementa los costos. Por ello la solución es utilizar otro sistema inmerso de bajo costo.

También se presenta una integración de varios tipos de tecnología, que puede servir para futuros prototipos, en los cuales ya el robot de Lego® se ha sustituido por un microcontrolador industrial (también llamado PLC, por sus siglas en inglés *Programmable Logic Controller*). En dicha integración intervienen tecnologías tanto análogas como digitales, así como protocolos de comunicación y se pretende hacer un diseño tan modular que permita intercambiar los diferentes elementos del sistema, e incluso controladores sin afectar a las otras partes.

Otro aspecto de beneficio que se encuentra, es la factibilidad de utilizar este producto de Lego®, en tópicos avanzados como los Sistemas Inmersos y la Robótica, en niveles superiores universitarios.

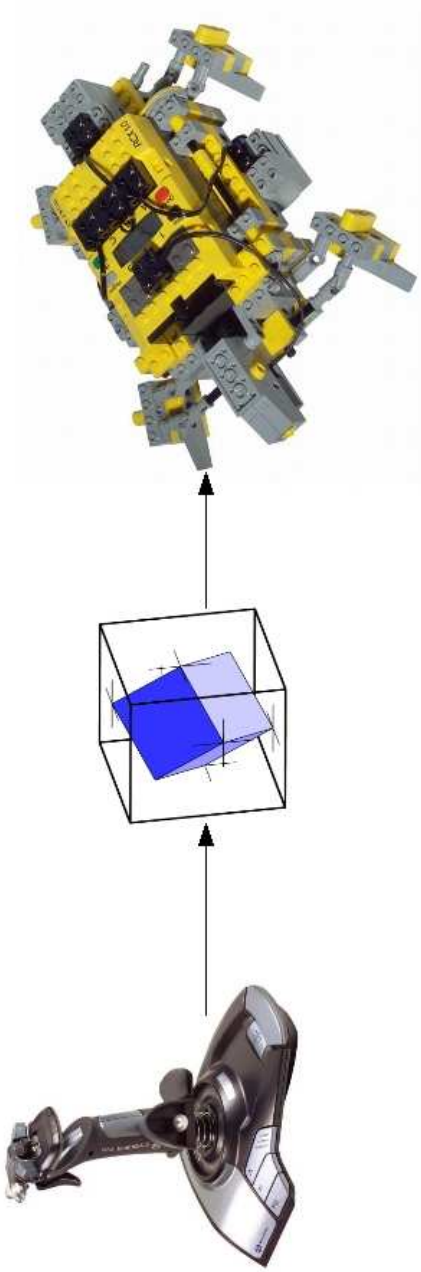

Figura I-1: La caja al centro representa el sistema inmerso a desarrollar en este proyecto

II. OBJETIVOS

A. General

Probar que el Lego® Mindstorms™ puede ser utilizado como un ambiente para desarrollo en el nivel universitario, de bajo costo, para cursos como sistemas inmersos, robótica así como sistemas operativos.

B. Específicos

1. Construir un sistema inmerso que envíe órdenes a la microcomputadora de Lego®, el RCX™, en forma inalámbrica, a través de una señal infrarroja.

2. Expandir el tipo de aplicaciones que actualmente se construyen con el RCX™, ya que la nueva ventaja de movilidad, sin depender de la computadora, puede dar lugar a robots más diversos que permitan estudiar fenómenos más específicos, por ejemplo, un prototipo de robot que localiza minas, el cual debe ser maniobrado en lugares donde una computadora no se puede llevar.

III. MARCO TEÓRICO

A. DEFINICIÓN DE UN SISTEMA INMERSO

Los sistemas inmersos se pueden definir de una forma general , como sistemas computacionales que están fuertemente acoplados al hardware y altamente integrados al software (también llamado firmware) que ejecutan para una tarea dedicada. La palabra inmerso denota que estos sistemas son una parte integral de un sistema mucho más grande, conocido como sistema envolvente (Li y Yao, 2003).

1. Ejemplos de sistemas inmersos. En el proceso de conceptualizar lo que es un sistema inmerso, los ejemplos son de gran utilidad. A continuación algunos:

Figura 1: Ejemplos de aplicaciones que utilizan sistemas inmersos en su interior

2. Características de un sistema inmerso. Hay algunas características que todo sistema inmerso debe cumplir y que lo define como tal. Entre ellas se pueden mencionar:

a. *Un sistema inmerso es dedicado a una única función**:* Ejecuta un programa único de manera repetida.

b. Por lo general, un sistema inmerso se orienta a lo pequeño (en diversas métricas): son de bajo costo (en relación a un sistema computacional más grande como una PC). Consumen poca energía, son de dimensiones físicas pequeñas.

c. Un sistema inmerso es reactivo al entorno en el cual se encuentra. Por medio de sensores, obtienen información que describe cierto fenómeno en la realidad, a partir de esto, de acuerdo a una programación interna, toma decisiones que se ejecutan por medio de efectores, sobre los cuales tiene control el sistema. Dependiendo del rendimiento que se desea que alcancen, pueden responder en tiempo real.

3. Métricas de diseño para un sistema inmerso. Una métrica, es una medida que permite evaluar el rendimiento de un sistema inmerso en varias áreas (no solamente en eficiencia). Un conjunto de dichas medidas pueden evaluar en forma cuantitativa el rendimiento de un sistema inmerso. Entre ellas podemos mencionar:

a. Costo por unidad: el costo monetario de reproducir una réplica del sistema original, sin tomar en cuenta el costo que tomó desarrollar el prototipo inicial (en la fase de desarrollo), NRI, a continuación expuesto.

b. Costo único de ingeniería (NRI por sus siglas en inglés Non Recurrent Engineering Cost): Este es el costo de desarrollar la primera unidad, es decir el primer prototipo; se incluye acá cualquier investigación que se haya llevado a cabo así como los materiales, consultores y otros.

c. Tamaño: el tamaño físico del sistema.

d. Rendimiento: el tiempo de ejecución del sistema teniendo en cuenta la cantidad de datos de entrada contra los de salida.

e. Consumo de energía: la cantidad de energía consumida por el sistema en cierto intervalo de tiempo.

f. Flexibilidad: la habilidad de realizar cambios, principalmente al hardware del sistema, sin incurrir en altos costos de NRI.

g. Tiempo de prototipo: el tiempo necesario para construir una versión funcional del sistema.

h. Tiempo de mercado: el tiempo que se necesita para tener el producto terminado y vendiéndose. Esto también toma en cuenta el tiempo de transporte desde el lugar de fabricación hasta que está colocado en el lugar donde se va a vender.

i. Factibilidad de mantenimiento: la habilidad de extender (principalmente en software) o mantener el sistema.

Dependiendo de lo que se desee alcanzar, el éxito de un sistema inmerso en particular, dependerá del correcto equilibrio del subconjunto de métricas que se quieran utilizar para evaluar el rendimiento final. Cuantas más métricas se elijan más complejo puede tornarse el diseño, ya que optimizar una, puede penalizar otra (o más); en especial las métricas de rendimiento y tiempo (de prototipo y mercado) penalizan a las de costos. Por ejemplo, tratar

de tener un prototipo funcional en muy corto tiempo, desencadenará en una métrica penalizada de costo NRI.

B. El Lego® Mindstorms™

El conjunto robótico Lego® Mindstorms™ es un producto originalmente diseñado para niños mayores de doce años y para uso en ambientes educativos. Sin embargo, tuvo un éxito inesperado en un mercado diferente, como lo es el de los adultos con pasatiempos de tecnología.

El objetivo principal del Mindstorms™ es proveer las herramientas básicas, de bajo costo, para construir robots con piezas y componentes de Lego® que además tengan “inteligencia” debido a una pieza especial, la cual contiene un sistema inmerso que desempeña el papel de “cerebro” del robot.

1. Breve historia. La primera versión comercial del Lego® Mindstorms™ fue lanzada en 1998, pero los orígenes de la parte central del conjunto (el sistema inmerso que provee la “inteligencia”) datan de 1986, cuando la compañía en alianza con los laboratorios de Medios del Instituto de Tecnología de Massachusetts (MIT) desarrollaron el entonces llamado “Ladrillo Programable”, que era una pequeña unidad que permitía que se le conectasen sensores y efectores con el fin de utilizarla en creaciones robóticas, teniendo en cuenta que la computadora tendría interacción con ella.

2. Componentes del conjunto. Los conjuntos de Lego® Mindstorms™ están formados por más de 700 piezas de lego tradicionales y del tipo Lego® Technic™ que son piezas orientadas a crear prototipos apegados a la realidad. Dentro de dichas piezas vienen además dos motores de alto empuje de corriente directa, dos

sensores de contacto y uno de luz. Hay varias versiones de Lego® Mindstorms™, las cuales son:

a. Robotics Invention System: Esta es la más poderosa de todas, ya que contiene al RCX™ (el segundo en la Figura 2b) y también es el que tiene más piezas.

b. Robotics Discovery Set: Este tiene una versión más sencilla de sistema inmerso, denominado SCOUT (primero en la Figura 2), la cual puede manejar menos motores y sensores

c. Droid Developer Kit: Este es el más sencillo de todos, tanto por el número de piezas como por el componente inmerso denominado microSCOUT (el último en la Figura 2b), que sólo maneja un motor y un sensor.

Figura 2a: Vista general del Robotics Invention System de Mindstorms™

Figura 2b: Los tres ladrillos que contienen sistemas inmersos para control de robots, según el conjunto de lego. De izquierda a derecha el SCOUT(b), el RCX™ (ael más poderoso y versátil) y el MicroSCOUT(c)

A partir de este punto en el documento, siempre que se mencione el término Lego® Mindstorms™ se entenderá el Robotics Invention System, el cual contiene al RCX™, y no se discutirá nada acerca del SCOUT™ y MICROSCOUT™.

3. Sensores. Los sensores actúan como dispositivos que convierten un fenómeno físico en uno que es capaz de ser procesado por un microprocesador. Dentro de los sensores para el Mindstorms™ están (ver Figura 3):

a. Sensor de tacto: éste tiene dos valores, presionado y no presionado. Es decir es binario.

b. Sensor de luz: envía una señal porcentual al RCX™ del nivel de luz en el ambiente.

c. Sensor de rotación: Estos envían una señal al RCX™ cada vez que se ha rotado una pieza, 22.5 grados sexagecimales.

d. Sensor de temperatura: Envía un voltaje proporcional a la temperatura del ambiente. Su rango de medición está entre los -20 ºC a los 50 ºC.

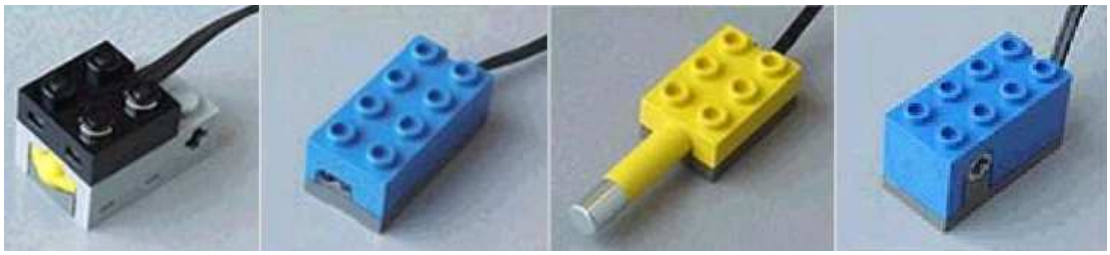

Figura 3: De izquierda a derecha los tipos de sensores disponibles comercialmente para el Mindstorms™: a) tacto b) luz, c) temperatura d) rotación

C. El RCX™

Lo que hace diferente el conjunto de Lego® Mindstorms™, de un set típico de Lego®, es que contiene una pieza denominada RCX™ (por sus siglas en inglés *Robot Command Explorer*). Ver Figura 4.

Figura 4: El RCX™

El RCX™ fue introducido al mercado como una “microcomputadora” capaz de darle inteligencia a las construcciones robóticas de Lego®. Sin embargo, no es una definición adecuada. El RCX™ es un sistema inmerso que utiliza un microcontrolador capaz de ejecutar programas del usuario que toman decisiones en base a las lecturas de sensores y ponen a funcionar motores de corriente continua que están enlazados mecánicamente a las construcciones robóticas de Lego®. Este microcontrolador también es capaz de operar en modo “esclavo”, obedeciendo órdenes de una entidad externa (generalmente la PC, o bien otro sistema inmerso como el caso del presente proyecto) que le envíe órdenes en su protocolo. Este hecho es el que permitirá desarrollar el sistema inmerso propuesto en este trabajo de graduación, como se explicará más adelante (ver Figura I-1).

1. Estructura física. En la parte interior del RCX™, podemos encontrar el circuito que almacena al microcontrolador y sus periféricos. Una vista interior se encuentra en la Figura 5.

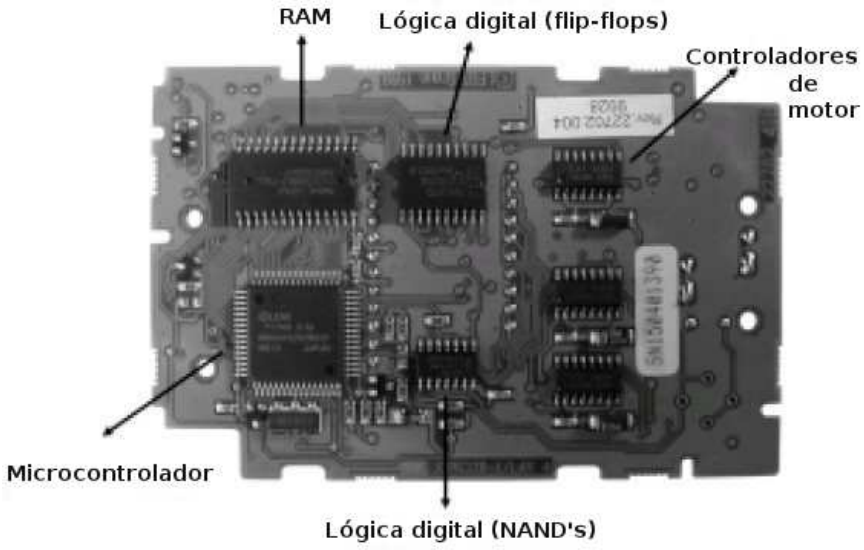

Figura 5: Circuito interno del RCX™, vista posterior

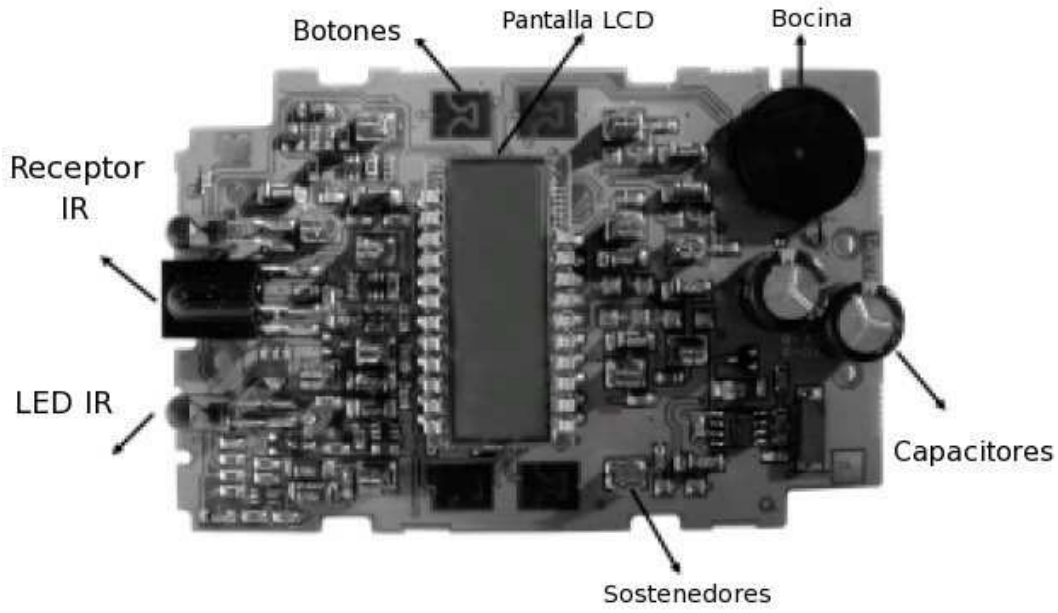

Figura 6: Circuito interno del RCX™, vista anterior

Procesador	**Hitachi H8/3292, 8 bits, 16 Mhz**
ROM	16 KBytes
SRAM, interna	512 KBytes
SRAM, external	32 KBytes
Salidas	3 puertos para motores
Entradas	3 puertos para sensores
Despliegue	1 pantalla LCD
Sonido	1 unidad de sonido
Cronómetros	4 Cronómetros (8 bits)
Baterías	6 x 1.5 V
Comunicaciones	Puerto IR (transmisor / receptor)

Tabla 1: Componentes detallados del RCX™ (Ferrari y Hilmer *et.al*, 2002)

2. Estructura lógica. Siguiendo el patrón más utilizado para sistemas inmersos, el RCX™ tiene una estructura de diseño por capas. A continuación, un diagrama del mismo:

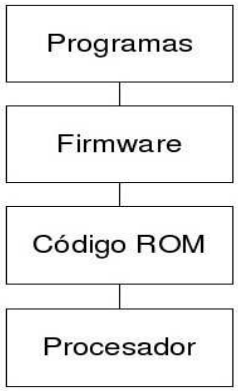

Figura 7: Estructura lógica del RCX™

El procesador es el que finalmente ejecuta las instrucciones en código máquina. El código ROM, es el conjunto de instrucciones provistas por el fabricante que proveen funcionalidades básicas, como el manejo de puertos, comunicaciones IR. Se podría comparar informalmente el ROM del RCX™ con el BIOS de una computadora. La tarea de éste, es iniciar los diferentes sub – sistemas del RCX™ para luego pasar el control al firmware, el cual funge como sistema operativo e intermediario entre el programa del usuario y el procesador.

a. Los programas en códigos de Bytes para el RCX™. Se entiende por código de bytes, un programa separado en una secuencia de bytes, el cual necesita un interpretador de bytes para ser ejecutado. Para programar el RCX™ , el usuario debe :

- Escribir un programa en lenguaje (normalmente el provisto por el fabricante).
- “Compilar” dicho programa en secuencias de código de bytes para ser interpretado por el firmware del RCX™.
- Descargar el programa utilizando una torre infrarroja conectada a la computadora, a la memoria RAM del RCX™.
- En el momento de ejecución el firmware transforma los códigos de bytes a instrucciones nativas del procesador utilizando rutinas del ROM.
- Por último, el procesador ejecuta las instrucciones nativas.

b. Reemplazando el ambiente de programación original de Lego®.

Debido al mercado objetivo del producto, el lenguaje de programación provisto por el fabricante es bastante gráfico y está orientado a ser “amigable” para programar, sacrificando así el nivel de complejidad que se puede alcanzar.

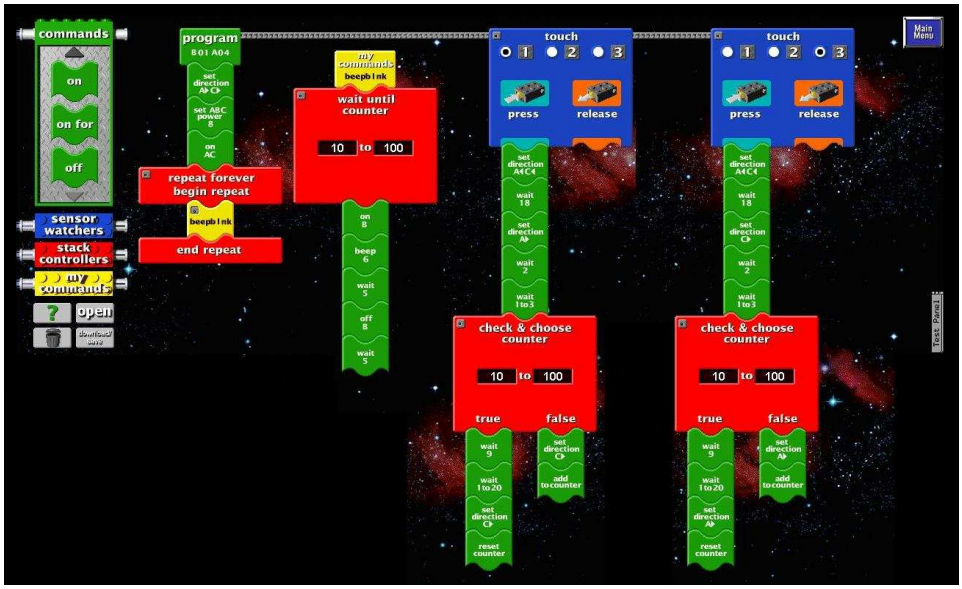

Figura 8: RCX™ Code, el ambiente de programación proporcionado por el fabricante

Debido al éxito que experimentó el producto en adultos con pasatiempos tecnológicos, se dio lugar a varios reemplazos del ambiente de programación:

1) Not Quite C: Un compilador que ayuda al usuario a utilizar un subconjunto de la sintaxis del popular lenguaje de programación “C” para ordenadores personales, para generar el archivo de códigos de bytes. Es la alternativa más sencilla al ambiente provisto por Lego®. Sin embargo no saca lo mejor del Hardware, ya que hace uso de las rutinas provistas por el fabricante, las cuales no son del todo eficientes. A continuación un programa ejemplo en NQC:

```
/* Ejemplo en NQC */
               int tiempo, giro;

               task main() {
                 SetSensor(SENSOR_1,SENSOR_TOUCH);
                 SetSensor(SENSOR_2,SENSOR_TOUCH);
                 start avanzar;

                 start evitar;
               }

               task avanzar(){
                 while(true){
                   OnFwd(OUT_A+OUT_C);
                 }
               }

               task evitar(){
                 while(true) {
                   if (SENSOR_1 ==1) {
                      stop avanzar;
                      OnRev(OUT_C); Wait(50);
                      start avanzar;
                   }
                   if (SENSOR_2 ==1) {
                      stop avanzar;
                      OnRev(OUT_A); Wait(50);
                      start avanzar;
                   }
                 }
               }
```

Listado 1: Un ejemplo de un programa escrito en NQC (Mateyan y González *et. al* 2001)

2) ROBOLAB: Conocido también como Mindstorms™ para escuelas, es un software optimizado para la enseñanza en ambientes educativos y es más poderoso que RCX™ Code. También es de índole visual.

3) Java: Permite generar código mucho más poderoso utilizando muchas características del paradigma de programación orientado a objetos. Este requiere que se sustituya el Firmware del RCX™ por una máquina virtual optimizada para correr en la plataforma del RCX™, llamada lejOS (por sus siglas en inglés ***lego java operating system***). A continuación, un ejemplo de Java en el RCX™:

```
import josx.platform.RCX.Motor;
import josx.platform.RCX.Sensor;
import josx.platform.RCX.Servo;
import josx.platform.RCX.Sound;

public class ServoDemo {

    public static void main (String[] args) throws
InterruptedException   {

     Servo s = new Servo (Sensor.S3, Motor.B, 0);

     boolean isthere = s.rotateTo (14);

        if (! isthere) {
           synchronized (s) {
             s.wait();
           }
        }

     Sound.buzz ();
     Thread.currentThread ().sleep (1000);

    }

}
```

Listado 2: Ejemplo de un programa en Java que controla un Servo en el RCX™ (un motor DC junto a un sensor de rotación) (Ferrari y Hilmer *et. al*, 2002).

3) C/C+: Este es el que genera código más eficiente y dinámico. Al igual que el anterior también requiere que se substituya el firmware, por uno nuevo llamado brickOS (por sus siglas en inglés *Brick Operating System*). A continuación, un programa ejemplo de BrickOS (el firmware que permite la programación en C/C++):

```
#include <conio.h>
#include <unistd.h>
#include <dsensor.h>
#include <dmotor.h>

wakeup_t colision(wakeup_t dato);

int main(int argc, char *argv[]) {
  int dir=0;

  while(1) {

    motor_a_speed(MAX_SPEED);
    motor_c_speed(MAX_SPEED);

    motor_a_dir(fwd);
    motor_c_dir(fwd);

    wait_event(&colision,0);
    if(SENSOR_1<0xf000)
      dir=0;
    else
      dir=1;

    motor_a_dir(rev);
    motor_c_dir(rev);
    msleep(500);
    motor_a_speed(MAX_SPEED);
    motor_c_speed(MAX_SPEED);

    if(dir==1) {
      motor_c_dir(fwd);
    } else {
      motor_a_dir(fwd);
    }

    msleep(500);
  }
}
wakeup_t colision(wakeup_t dato) {
        lcd_refresh();
        return SENSOR_1<0xf000 || SENSOR_2<0xf000;
}
```

Listado 3: Un ejemplo de un programa en C/C++ para BrickOS el firmware alternativo para el RCX™ (Mateyan y González *et. al* 2001).

c. Reemplazando el Firmware del RCX™. Lo que propició la popularidad del RCX™ entre la gente adulta fue la decisión de la compañía de almacenar el firmware en memoria RAM (ver Figura 7). Esto quiere decir que el sistema operativo del aparato no viene atado al sistema inmerso (de la misma manera que un ordenador personal puede utilizar otros sistemas operativos y no está atado a ninguno en particular).

La ventaja principal de reemplazar el firmware del fabricante es sobrepasar las limitantes que este tiene y en su lugar utilizar un firmware que realmente aproveche las características de hardware y sobre todo, si el firmware que se utiliza no es de la conveniencia del usuario, entonces siempre se puede regresar al firmware original. A continuación más detalles sobre la implementación de estos proyectos:

1) BrickOS: Este fue el primer intento de escribir un reemplazo al firmware provisto por el fabricante. Se inició en 1999 por Markus Noga y se convirtió en un proyecto de fuente abierta. Su objetivo principal es superar las limitantes que tiene el firmware original. Para mejorar la eficiencia, se ejecuta el código en forma nativa. Sobre todo utiliza un lenguaje muy fuerte como lo es C/C++ (Noga, 1999). Dentro de las características que brickOS provee tenemos:

a) Carga dinámica de programas y módulos.

b) Implementación de redes de RCXs sobre un medio infrarrojo.

c) Multi-tarea y calendarización anticipada.

d) Controladores para todos los sub-sistemas del RCX™:

- Motores
- Sensores (rotación, luz, contacto, temperatura)
- Interrupciones
- Pantalla de cristal líquido
- Manejo de Botones

- Control de batería

e) Modo nativo de trabajo de 16 Mega-Hertz.

f) Accesso a los 32 KB de memoria RAM.

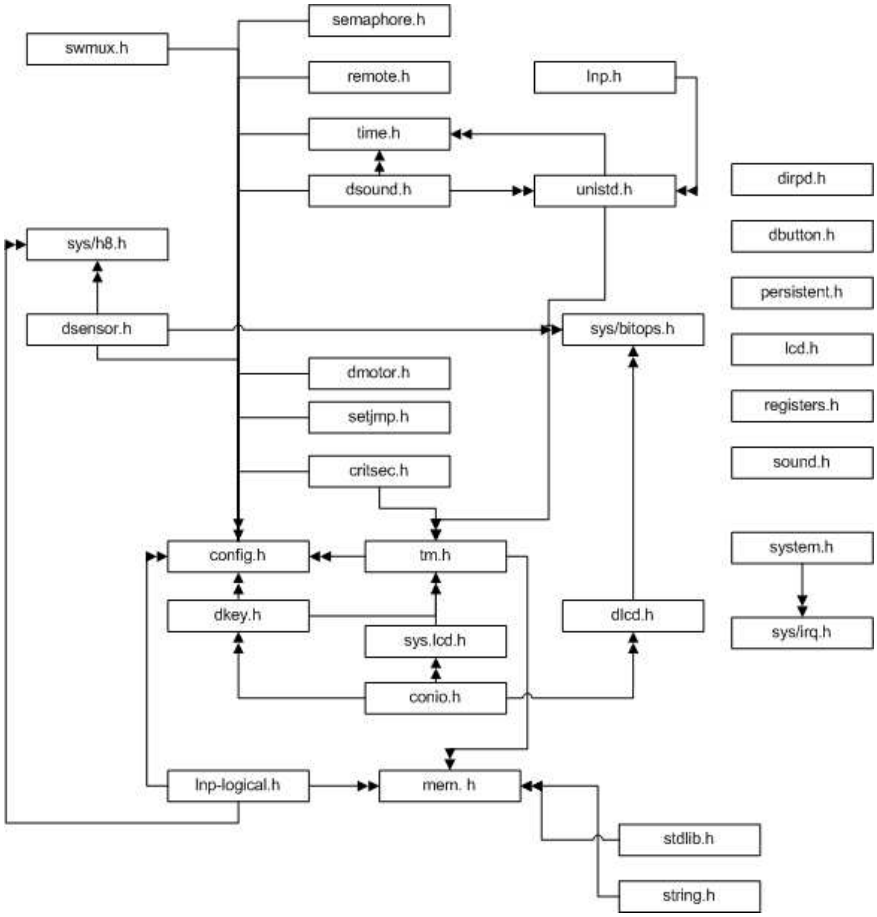

Figura 8a: Organización del sistema operativo BrickOS (http://brickos.sourceforce.net)

2) pbForth: Una variante del lenguaje Forth, concebido originalmente para robótica y sistemas inmersos, desarrollado en los 60's. Este, al igual que el anterior permite generar código en extremo eficiente, sin embargo utiliza notación reversa polaca, y su curva de aprendizaje es bastante alta en comparación con todos los anteriores.

3) Lego Java Operating System(lejOS): este reemplazo del firmware para el RCX™ provee muchas de las características típicas de Java, como lo son calendarización anticipada, arreglos múltiples dimensiones, operaciones de punto flotante, funciones de trigonometría por mencionar algunas.

Figura 8b: Niveles de abstracción de lejOS

D. Protocolo serial de comunicaciones del RCX™

Esta información forma parte del núcleo teórico que sustenta este trabajo de graduación. Las comunicaciones con el RCX™ tienen lugar en un modelo maestro/esclavo, donde el RCX™ funge como esclavo. Las transmisiones de datos son transportadas en paquetes, con encabezado, cuerpo y sumas de verificación.

1. Niveles de abstracción de la comunicación

a. Nivel de comando. En el nivel de comando, el RCX™ recibe órdenes o solicitudes de datos. Hay varios tipos de instrucciones que una entidad externa (en este caso un sistema inmerso) puede enviarle al RCX™:

-*Instrucciones inmediatas* (incluyendo solicitudes de datos)

-*Instrucciones de transferencia de datos*

-*Mensajes* (provenientes de otro RCX™)

-*Instrucciones de control remoto*

Del lado del RCX™ se pueden originar los siguientes mensajes:

1) Respuestas a las solicitudes de una entidad externa

2) Descargas de datos

3) Mensajes (con destino para otros RCXs)

Figura 9: Niveles de abstracción del modelo de comunicaciones del RCX™ (Mientki, 2001)

b. Nivel de paquete. En una comunicación normal, toda la información se transporta en un paquete de datos. El paquete consiste en un encabezado que comienza con:

```
55 FF 00
```

Y continua con:

$$D_1\ \sim D_1\ D_2\ \sim D_2\ \ldots\ D_n\ \sim D_n\ C\ \sim C$$

Donde D1 .. Dn son los bytes en el cuerpo del mensaje y ~D1.. ~Dn son los bytes complementados, por último $C = D_1 + D_2 + \ldots D_n$ y ~C es el complemento de C. El RCX™ nunca ejecuta el mismo mensaje dos veces. Esto es con el fin de evitar errores asociados a la superposición de la luz con la onda infrarroja. Es por eso que cualquier mensaje tiene dos versiones, una con el 3er bit en estado lógico de 1 y la otra no. Es decir, si se desea transmitir el mismo mensaje dos veces se debe ejecutar el cambio de dicho bit, de lo contrario el RCX™ ignorará la segunda transmisión (Baum, 1998).

c. Nivel RS-232. El estándar RS-232C es un método de comunicación serial asíncrono. El hecho que sea serial significa que los datos son enviados un bit a la vez y que sea asíncrono indica que dichos datos no se envían en segmentos de tiempo predefinidos. La transferencia de datos puede iniciar en cualquier momento, y es tarea del receptor detectar cuando un mensaje comienza o termina.

Los siguientes aspectos son parte básica de la especificación RS-232:

1) Torrente de datos: La información se segmenta en “palabras”, las cuales son un conjunto de bits de longitud variable (de común acuerdo entre receptor y emisor). La longitud de la palabra de datos es generalmente entre 5 y 8 bits. Luego de la palabra de datos, existen bits adicionales que tienen como propósito sincronizar y corregir errores. Es indispensable que ambos participantes de la comunicación tengan los mismos parámetros, de lo contrario la palabra de datos puede ser malinterpretada o bien no reconocida. El nivel de voltaje puede tener dos estados, el primero conocido como de marcaje (en inglés *marking*) y el segundo es el de espacio (*spacing*), teniendo un valor lógico de 1 y 0, respectivamente.

2) Bit de comienzo (Start – Bit): Como se mencionó anteriormente RS-232C define un tipo de comunicación asíncrona. Por tal razón cada palabra en la comunicación se inicia con un bit de “atención”. Dicho bit es de marcaje, ya que previo a la comunicación la línea estaba en estado de espacio, entonces dicho cambio notifica al recepto de la comunicación entrante.

3) Bit de paridad: Para detectar errores, es posible agregar un bit extra a la palabra de comunicación. Este bit debe de ser agregado automáticamente; el emisor calcula el valor del bit, dependiendo de la cantidad de bits de marcaje (en estado de 1) que la palabra tiene. Por su lado, el receptor también calcula el valor del bit, según la palabra que recibió y si no concordase, la palabra debe ser descartada.

4) Bits de parada: Este bit existe por si, en dado caso, el receptor no detectó el bit de comienzo (por ruido en la línea o cualquier razón). Entonces el bit de parada sirve para poder volver a sincronizar la palabra actual. Con la ayuda del reloj local, el receptor puede calcular y notar el el bit de comienzo no llegó correctamente. Por un lado la paridad “par” pone el bit final de paridad en estado de marcaje (es decir un 1) cuando la cantidad total de bits (de marcaje) en la palabra de datos es inpar. Y esto es inverso cuando es impar ya que el bit de paridad se pone en 1 cuando la cantidad de bits de marcaje en la palabra es par.

5) Propiedades físicas: El estándar RS-232 se utiliza en toda clase de ambientes y configuraciones, entre ellos ambientes industriales (que no necesariamente involucran computadoras personales). Esto tiene un impacto en los niveles de voltaje que cada pin puede tener. Sin embargo, las especificaciones del estándar datan de la década de los 70, por lo cual hoy día es posible llevar al límite este tipo de comunicaciones, especialmente en lo que es la distancia máxima permitida entre transmisión y recepción; así mismo la velocidad de transferencia también ha mejorado significativamente,

ya que al inicio únicamente soportaba 20kbps y actualmente se pueden alcanzar velocidades de hasta 1.5Mbps. A continuación los voltajes especificados:

Nivel	Capacidad del emisor (V)	Capacidad del receptor (V)
Estado lógico de 0 (espacio)	5 .. + 15	+3 .. +25
Estado lógico de 1 (marcaje)	-5 .. -15	-3 .. -25
No definido por el estándar	-	-3 .. +3

Tabla 2: Voltajes especificados por el estándar RS-232 (Bies, 2001)

Propiedad	Valor
Codificación de Bit	2400 bps (también 4800 bps)
Bits de inicio de palabra	1
Longitud de palabra	8
Bits de fin de palabra	1
Definición de '0'	417 micro segundos de pulso de 38kHz de IR
Definición de '1'	417 micro segundos de pulso de nada

Tabla 3: Características RS232 de la comunicación del RCX™ con una entidad externa

E. Joystick como dispositivo de entrada

1. Funcionamiento del joystick digital. El tipo más común de joystick por un tiempo fue el digital, también llamado joystick de atari (ya que venía originalmente con la consola de juegos de video Atari 2600). Luego estos

joysticks fueron adaptados por las primeras computadoras para el hogar (Commodore 64, Amiga y otras). El joystick en sí consistía en cuatro botones para posicionamiento y uno para disparo.

Figura 9a: un joystick digital tipo Atari

2. Funcionamiento del Joystick analógico para juegos. Un joystick para PC, es un dispositivo mecánico que transforma indicaciones del usuario en un plano XY a señales digitales que van hacia una tarjeta simple de 8 bits, de entrada y salida, la cual convierte en digital dicha información y está conectada al bus ISA de la computadora con la dirección 201h. El CPU puede leer y escribir a dicho puerto (ver figura

10). En esencia el joystick indica posicionamiento en el eje x y eje y, por medio de resistencias variables que varían según la posición del del dispositivo mecánico y es tarea de la computadora o entidad externa, convertir a un valor digital dicho dato analógico y tomar la decisión de acuerdo al valor.

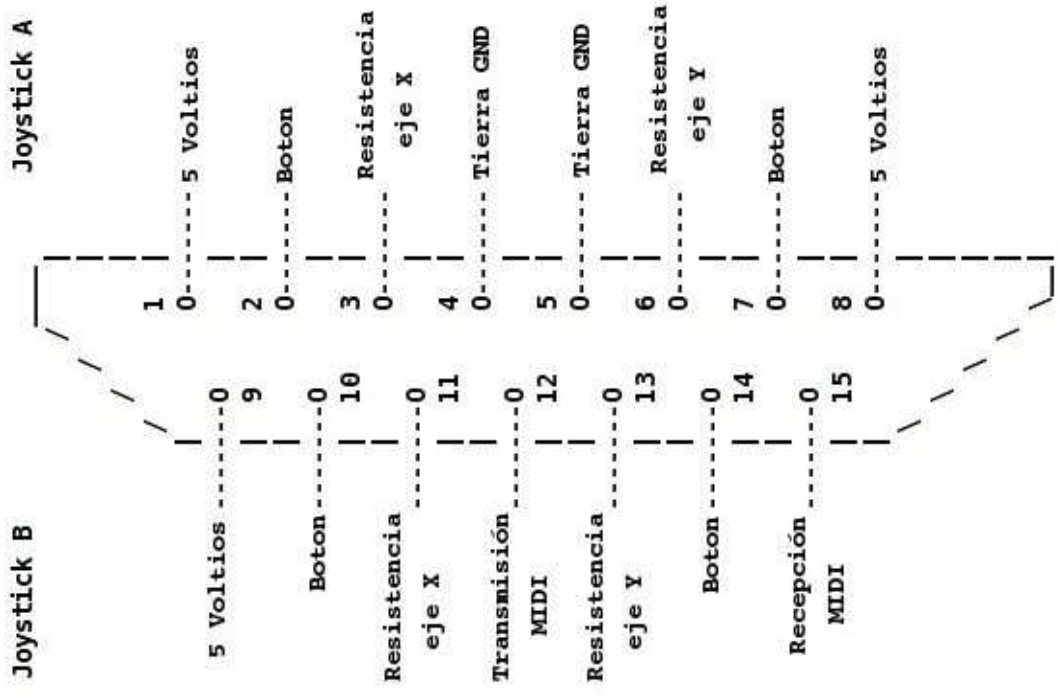

Figura 10: Diagrama de pines de un joystick típico para computadora personal

F. Tecnología de procesadores

La tecnología de procesadores se refiere a la arquitectura del núcleo computacional que se utiliza para implementar la funcionalidad de un sistema en particular (Vahid y Givargis, 2002).

A pesar que el término *procesador* es usualmente asociado con los procesadores programables en software, se puede referir también a otros

sistemas digitales, no programables, según la anterior definición. Se puede clasificar, de una manera general, a los procesadores, así:

1. Procesadores de propósito general. Este es un dispositivo que es adaptable a una buena variedad de aplicaciones. Una de las características de este tipo, es que tiene una memoria de programa (debido a que es de propósito general, el fabricante no sabe qué es exactamente lo que el dispositivo hará, sino es el usuario final quien lo programa). Otra característica es la ruta de datos (conocido también como bus de datos), que le permite realizar diversas tareas computacionales y también determina su rendimiento (tiempo que se tarda en realizar cierta operación, y la cantidad de datos que puede operar por unidad de tiempo). Utilizar un procesador de este tipo es en muchas maneras ventajoso para desarrollar un sistema inmerso, ya que muchas métricas pueden optimizarse notablemente, entre ellas las de *Tiempo de Mercado,* y el *costo NRI* , ya que el diseñador del sistema se concentra únicamente en la parte de software (es decir programar el procesador) y evita la parte de diseño digital, la cual puede penalizar severamente dicha métrica.

Entre ellas el costo por unidad puede dispararse para un alto volumen, porque si se espera que el dispositivo se venda en grandes cantidades, sería mejor diseñar un circuito a la medida que haga la tarea deseada (y que su costo sea bajo) y no comprar un dispositivo tan amplio y general (como lo es el procesador) y que no se utilice mucha de la funcionalidad que viene con él (y por la cual se paga también y por supuesto va afectar el costo por unidad). Otras desventajas son que para ciertas aplicaciones puede ser que no tengamos el rendimiento que se desea (ya que la misma generalidad de este tipo de procesador hace que sus prestaciones estén más centradas, es decir que no sean tan bajas, pero tampoco tan altas).

2. Procesadores de propósito único (basados en Hardware). Este tipo es el menos asociado al concepto actual de procesador. Es un circuito digital que ejecuta exactamente un programa, es decir el usuario no puede “programarlo”. Como ejemplos están: los co-procesadores matemáticos, los descomprimidores de imágenes o bien simplemente el controlador de un periférico, son un buen ejemplo de ello. La implementación de un sistema inmerso con este tipo de procesador implica la optimización de ciertas métricas, así como la penalización de otras. Se puede decir que tiene el comportamiento inverso al observado en los procesadores de propósito general, en cuanto a las métricas.

Por ejemplo, el rendimiento puede ser bastante bueno debido a que es un circuito diseñado específicamente para la tarea, al contrario de un procesador de propósito general el cual contiene un marco de trabajo mucho más general (es como decir que con un camión se puede hacer de todo, desde transportar carga pesada, hasta ir al cine, pero definitivamente la segunda actividad sería realizada mucho más cómodamente utilizando un auto compacto sin mayor potencia). Por otro lado, el costo también será bastante reducido ya que el procesador va a tener únicamente los componentes que se necesitan para llevar a cabo la tarea con eficiencia. Sin embargo, la penalización viene del lado del NRI, ya que la solución se está desarrollado desde cero, se debe construir el hardware, probarlo y certificarlo (cosas que no se deben de hacer con el anterior tipo de procesador). Otra métrica que se ve afectada es la de la flexibilidad, ya que este tipo de procesador hace lo que hace, nada más. Si en el futuro se desea agregar al sistema o expandirlo, se debe construir otro procesador o bien optar por migrar a otro tipo de procesador.

3. Procesadores de aplicación específica. Este tipo de procesador está en el punto medio de los dos anteriores. Se conoce como ASIP o

bien ASIC (por sus siglas en inglés Application Specific Proccesor o bien Aplication Specific Integrated Circuit). Un ASIP está optimizado para cierta actividad, y tiene características como periféricos incorporados desde su fabricación, tales como convertidores analógico/digital y dispositivos de procesamiento de señales. Los ASIPs tienen la ventaja de que eliminan los componentes de un procesador general que no son utiles para la aplicación a la que fueron diseñados y agregan los que si lo son. Ejemplos de ASIPs son los microcontroladores y procesadores de señales digitales. Un ASIP provee a un sistema inmerso el beneficio de la flexibilidad, manteniendo un buen rendimiento, consumo de energía y tamaño. Es de notar que el costo no es tan alto como los procesadores generales, pero tampoco tan bajo como los específicos.

4. Definición formal de un controlador. Un controlador es una interconexión de componentes formando una configuración de sistema, que va a provocar cierta respuesta deseada del sistema que controla (Dorf y Bishop, 1998).

5. Definición formal de un microcontrolador. Un microcontrolador es un chip altamente integrado que incluye en una sola pastilla (chip) todas o casi todas las partes para formar un controlador para un sistema dado. Este microcontrolador puede ser llamado “una solución de pastilla” . Normalmente incluye:

- CPU (Unidad central de procesamiento)
- RAM (Memoria de acceso aleatorio)
- EEPROM/PROM/ROM (memoria de solo lectura que se puede borrar y programar con un voltaje superior al nivel digital que es 5 voltios)
- E/S (Entrada y salida paralela y serial)

- Cronómetros
- Manejo de interrupciones

El hecho de incluir todas estas características, específicas a su tarea, hace que el costo sea relativamente bajo (contra tratar de comprar un procesador de propósito general y conectarle manualmente cada uno de estos periféricos). Un microcontrolador típico tiene instrucciones de manipulación de bit, acceso sencillo y directo a E/S (entrada y salida) y un rápido y eficiente control de interrupciones (Hersch, 1997).

6. Definición formal de un microprocesador. Un microprocesador, conocido también como Unidad Central de Proceso (CPU), es el eje central de un sistema computacional de propósito general (Brain, 2004).

7. Esquemas de de memoria para un microprocesador. Mientras que los registros internos de un microprocesador sirven para sus operaciones a corto plazo, la memoria sirve para propósitos de mediano y largo plazo. Prácticamente podemos decir que todo lo que el procesador almacena, va a ser o bien el programa que ejecuta o bien un dato, que es la información que está siendo ingresada y transformada por el programa, para ser enviada a un medio de salida.

Hay dos enfoques para organizar estos dos tipos de memoria : unidos o separados. Cuando los datos y el programa comparten el mismo espacio, se conoce como *Arquitectura de Princeton,* cuando dichos espacios están separados, se conoce como *Arquitectura de Harvard.*

G. Periféricos que se pueden conectar a un microprocesador

1. I^2C. Philips Semiconductors desarrolló el bus I^2C (por sus siglas en inglés *Inter-Integrated Circuit Bus*) a finales de la década de los 80s.

Es un bus serial de dos líneas. La transmisión de datos en su especificación original era de 100 kb/s y el direccionamiento de 7 bits (se pueden conectar hasta 128 direcciones, menos unas pocas que están reservadas para la especificación). Una reciente mejora a la especificación incrementó la velocidad a 3.4 Mb/s y el direccionamiento a 10 bits (pudiéndose conectar hasta 1024 direcciones).

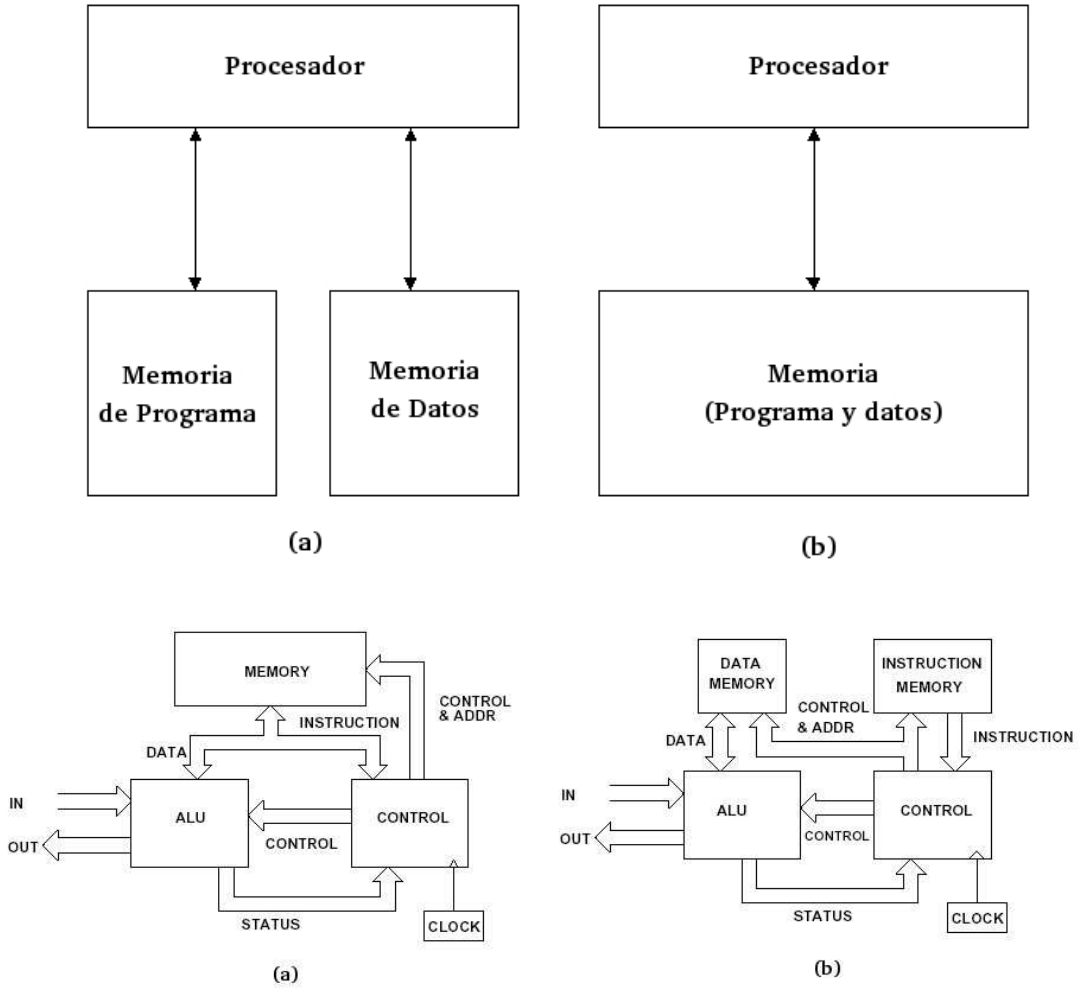

Figura 11: Arquitecturas detalladas a) Arquitectura de Harvard. b) Arquitectura de Princeton

Dentro de los dispositivos que pueden a ser accedidos a través de I^2C tenemos:

a. Memoria EEPROM

b. Memoria Flash

c. Memoria RAM

d. Relojes de tiempo real

e. *Watchdog timers*

f. *Microcontroladores*

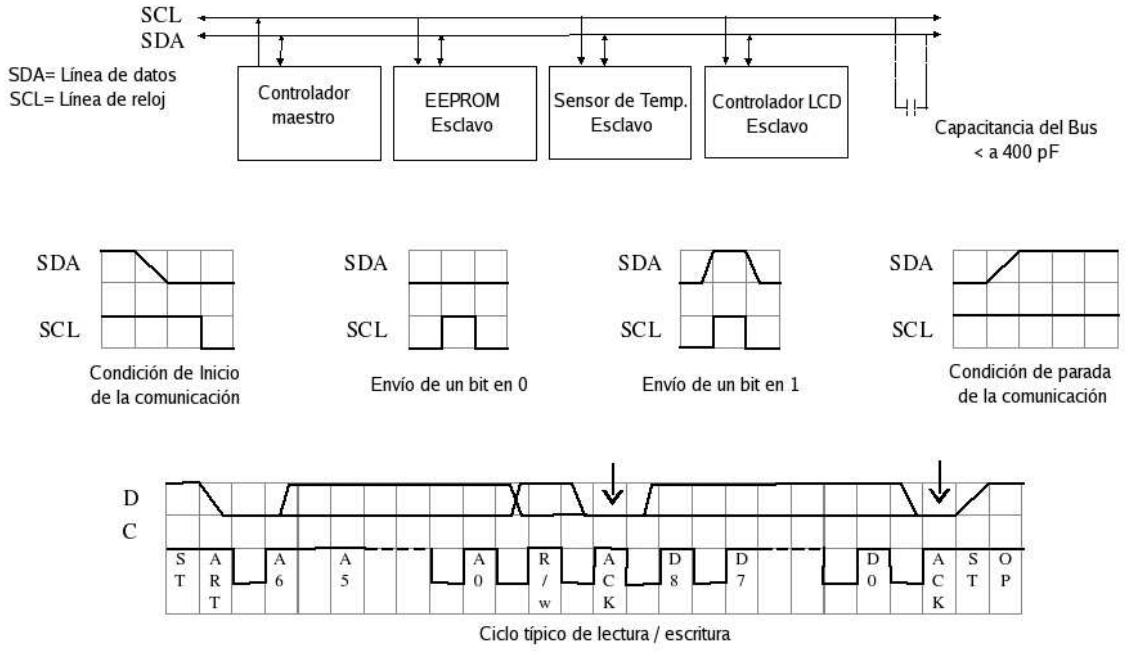

Figura 12: Esquema de funcionamiento del bus I^2C

IV. DESARROLLO DE LA PROPUESTA

A. Definición del problema

Hay dos formas de “darle vida” a un robot. La primera es hacerlo independiente de la computadora personal por medio de un programa (escrito en cualquiera de los ambientes / firmware expuestos en el Marco Teórico) que el RCX™ ejecuta solo. Y la segunda es controlarlo en modo “maestro / esclavo”, en el cual la computadora personal o un sistema inmerso (que es el propósito de este proyecto, ver Figura I-1) funciona como maestro y el RCX™ ejecuta las instrucciones de acuerdo a un protocolo establecido por el fabricante.

El problema es que si se desea controlar el robot en modo “maestro/esclavo” se necesita tener una computadora para poder “hablarle”, a través del protocolo del fabricante, lo cual limita la movilidad así como las aplicaciones que se pueden desarrollar.

Como solución a dicho problema, se propone desarrollar un sistema inmerso que desempeñe el papel de “maestro” y tener así la capacidad de obtener las direcciones del usuario y transformarlas al protocolo de comunicaciones del RCX™, todo esto sin depender de la computadora para alcanzarlo.

El propósito de las secciones siguientes es delimitar los requerimientos de dicho sistema, teniendo en cuenta futuras áreas para desarrollo futuro que se recomendarán al final del trabajo. En algunas secciones se plantean alternativas que a primera vista no son tan viables. El objetivo de dichas secciones es simplemente servir como camino para enumerar las posibilidades y con un análisis detallado proponer un diseño para finalmente implementar.

B. Planteamiento de requerimientos

Para desarrollar un sistema inmerso que se comunique con el RC es necesario que cumpla, o tenga, las siguientes condiciones:

1. Comunicación RS – 232. El sistema que se desarrolle debe tener la habilidad de comunicarse con el RCX™ por medio de un puerto RS – 232. Esto puede ser a nivel de hardware (que el microcontrolador ya venga con el puerto) o bien a nivel de software y que haya que implementar el estándar completamente desde cero.

2. Medio de captura versátil. El sistema debe ser capaz de guiar al RCX™, basado en las direcciones que el usuario de.

3. Versatilidad y escalabilidad. El diseño debe ser lo suficientemente versátil y escalable para permitir que futuros trabajos exploren áreas afines utilizando como marco de trabajo lo desarrollado.

4. Almacenamiento interno. Debido a que el RCX™ tiene su propio protocolo de comunicación, eso implica que hay que manejar ciertas reglas sobre la comunicación RS232 (en una capa superior). Por tal razón, el sistema debe poder almacenar dichas reglas para construir paquetes dinámicamente.

5. Sobre-dimensionamiento del rendimiento. Para poder construir proyectos futuros sobre el presente, el controlador del sistema inmerso debe poder soportar más carga de procesamiento de la que la actual implementación demande.

C. Organización del resto de esta sección

En la sección siguiente (D) se presentan los detalles del diseño propuesto para este proyecto. Al igual que en cualquier otro proyecto de ingeniería, se debe proponer alternativas, diseñar de acuerdo a ellas, para luego regresar y

corregir, y si es necesario volver a diseñar con el conocimiento adquirido. Por respeto al lector, esta sección presenta el diseño final (luego de haber pasado por el proceso mencionado, sin embargo se hace referencia a alternativas descartadas, cuyo detalle puede ser encontrado en la sección siguiente), y luego de ello, el resto es propuestas de diseño (sección G), cuyo fin es guiar al lector interesado en descubrir cómo se construyó el presente sistema.

D. Diseño del sistema inmerso

La sección D trata con el diseño final del sistema. Para una discusión detallada de todas las posibilidades ir a la sección G.

Lo importante es ver el sistema como un todo, tomando en cuenta métricas como las siguientes:

- *Eficiencia total*
- *Costo total*
- *Tiempo de desarrollo total*
- *Rendimiento total*

La metodología de llegar al diseño más conveniente, es analizando las métricas expuestas en el Marco Teórico, contra la información presentada en las secciones subsiguientes . Hay algunas métricas que no son indispensables, por lo cual se nota cuando se optimizan o bien penalizan pero la decisión que finalmente va a llevar a la propuesta de diseño está basada en las métricas a optimizar.

1. Enlace RS232. La selección de las propuestas presentadas en las secciones subsiguientes da inicio con el módulo RS-232.

a. Enlace RS232 implementado en forma manual. La primera opción a descartar debe ser la presentada en *G.1.a* que propone la implementación del módulo a mano, es decir que el programador controle el

manejo de interrupciones y la verificación de errores de bajo nivel (paridad, bits de inicio, bits de parada) del protocolo RS232. Primeramente se mencionan las métricas que este enfoque (ya descartado) optimiza para después mencionar las que se ven penalizadas y hacen que se descarte. Se analizan a continuación algunas de las métricas utilizadas para descartar la presente alternativa:

• ***Tamaño general del firmware:*** esta métrica se penaliza ya que el programador tiene que controlar las ráfagas de datos en el firmware, es decir son líneas de código para atender cada bit que ocurre,lo cual hara que el tamaño del programa (y por ende el tamaño de la memoria para almacenarlo) crezca.

• ***Costo por unidad:*** debido a que el controlador (o microprocesador) no tiene módulo RS232 en hardware, eso va a reducir el costo del mismo, haciendo que producir una replica del dispositivo sea más barata.

Sin embargo son métricas que por el momento (en el ámbito temporal de este trabajo) no es prioritario optimizar. A continuación las métricas que penaliza:

• ***Tiempo de prototipo:*** *por esta métrica se entiende que el programador va invertir más tiempo en crear el firmware.* Debido a que el mismo procesador se encargaría de manejar las comunicaciones RS232, el manejo de interrupciones depende del cristal (o reloj) que tenga en ese momento el controlador o microprocesador. Esto quiere decir que las rutinas deben, de alguna manera, ser configuradas según la frecuencia. Esto va a penalizar el tiempo en el que el prototipo se tendrá listo, ya que, es más complicado que solamente atender una interrupción en la cual el dato ya está colocado en cierta posición de memoria (o en un registro en el caso de un microcontrolador).

• ***Rendimiento:*** estar atendiendo esta comunicación serial penalizará el rendimiento general del sistema ya que el tiempo valioso de procesador, que se debiese de estar utilizando en cosas “más importantes” como análisis de datos (o bien simplemente ahorrar consumo de energía), se tendrá que utilizar para transmitir y recibir las tramas de datos RS232.

• ***Flexibilidad:*** la programación se volverá más complicada ya que cada vez que se desee realizar un cambio al software, tendrá que estarse considerando cómo afectará esto a la tarea de atender la transmisión RS232.

• ***Factibilidad de mantenimiento:*** por otro lado, si en un futuro se desea agregar más carga al controlador puede que se llegue a un tope más rápidamente debido a esto. Esto quiere decir que el procesador está invirtiendo tiempo en “atender una tarea muy básica” como lo es la comunicación RS-232 en vez de poder invertir dicho tiempo en la aplicación (firmware).

b. Un microcontrolador con el módulo RS232 de fábrica: esta es la opción que más conviene al proyecto G.1.b. Esto debido a que en general es el que menos complicaciones provee, tanto en la etapa de desarrollo como en la de implementación y ejecución. Las métricas que optimiza son las siguientes:

• ***Rendimiento:*** este tendrá un punteo más alto debido a que el controlador podrá concentrarse en la lógica del sistema y no se penalizará por atender la comunicación RS232.

• ***Consumo de potencia:*** la cantidad de energía consumida por el sistema será menor, porque el controlador tendrá menos trabajo que realizar. Cuando no tenga nada qué hacer el programador puede ponerlo a “dormir” y esperar que una interrupción lo “despierte”. Esto es menos energía consumida.

• ***Flexibilidad:*** cada vez que se realicen cambios al software no tendrá que pensar qué pasará con el módulo RS232 ya que dicho módulo está en Hardware y el controlador lo atiende en interrupciones y el software no está ligado a él. Sin embargo también hay que hacer mención que al igual que el caso de implementar el módulo a mano, la dependencia del reloj continúa ya que, como el módulo se encuentra interno en el controlador, este se ve en la necesidad de utilizar el mismo reloj del controlador. Pero aquí el cambio se reduce a re-configurar el módulo, no a cambiar gran parte del código, si fuese el enfoque manual. Cuando se dice re-configurar el módulo, básicamente es escribirle a algún registro un valor proporcional a la frecuencia para que este ajuste su generador de tiempos para el puerto serial.

• ***Tiempo de prototipo:*** el tiempo necesario para construir una versión funcional del sistema se reducirá ya que la parte de software se reduce a configurar el módulo al inicio y atender la interrupción correspondiente y ya se tienen los datos.

• ***Factibilidad de mantenimiento:*** el sistema debe poder extenderse fácilmente debido a que parte de los requerimientos es elegir un microcontrolador / microprocesador con mayor capacidad para futuros desarrollos basados en este proyecto y si a eso se agrega el hecho que el módulo RS232 no es atendido en software por el controlador, aumentar de carga no será un problema.

No todo es ideal con este enfoque. Algunas métricas como, por ejemplo, el Costo por Unidad se pueden ver penalizadas por tener que escoger un controlador que sea más costoso a cambio de los beneficios anteriormente mencionados. Sin embargo, el propósito de este trabajo no es optimizar un diseño para producción masiva, por lo cual dichas métricas no tienen alta prioridad.

c. Un microprocesador con un puerto RS232 conectado externamente. A pesar que éste no es el enfoque elegido (ver sección G.1.b), tiene una ventaja digna de mencionar antes de analizar sus métricas una a una. Este es el único enfoque que elimina la dependencia del reloj, en este caso del procesador, ya que dispositivos en el mercado como el 16C550 o el Intel 8250 tienen entradas para reloj independiente del procesador (aunque se puede hacer que utilicen el mismo, para ahorrar costos). Eso garantiza que se puede elegir independiente el reloj del microprocesador sin tomar en cuenta el módulo RS232. De todos modos se presenta un análisis de sus métricas ya que se desea dejar documentado las razones que llevaron al sistema, con el diseño final del presente trabajo de graduación. A continuación las métricas que penaliza:

• ***Costo por unidad:*** éste necesariamente tiene que incrementarse, como se observa en la figura 12b, ya que el módulo externo tiene un costo extra al microprocesador además hay que considerar que el cristal del módulo, si se trabaja en modo independiente (lo cual optimiza otras métricas) también tiene un costo asociado, que penalizaría la producción masiva del prototipo.

• ***Tamaño:*** el tamaño físico del sistema será superior frente a los otros enfoques. No sólo se requiere un circuito integrado extra, sino se tiene que conectar al microprocesador, esto implica más circuitos.

• ***Consumo de potencia:*** debido a que se estará alimentando a otro circuito integrado, el consumo de potencia general carecerá.

A pesar de las métricas anteriores, haciendo a un lado las cuestiones de costo, espacio y tiempo de desarrollo, al final un microprocesador es más escalable que microcontrolador . A continuación las métricas optimizadas por este enfoque:

• ***Flexibilidad:*** este enfoque no sólo propone flexibilidad en el software, porque el microprocesador atiende solamente interrupciones, sino también en hardware, porque, el módulo cuenta con su propio reloj.

• ***Factibilidad de mantenimiento:*** igualmente extender el software para ponerle más carga al sistema no será problema, gracias a la anterior métrica, porque prácticamente el diseñador solamente realiza el cambio con la certeza que el resto de módulos no se verán afectados

2. Medio de captura versátil. La discusión en este punto, se basa primordialmente en la información resumida que presenta la tabla 3 (G.2). Luego de observar detalladamente se puede concluir qué no importando qué elección se realice en este sentido (al menos en esta etapa) ya que para el usuario será completamente transparente. Con el fin de tener un diseño más eficiente, se discuten las métricas que se deben de optimizar:

a. Costo por unidad. Utilizar un joystick analógico encarecerá la solución, pero como se verá más adelante, esto también optimiza otras métricas. También hay que mencionar que la utilización de un joystick analógico conlleva necesariamente un módulo de conversión analógico digital, lo cual también incrementa los costos, ya sea que se utilice microcontrolador o microprocesador.

b. Flexibilidad. si se utiliza un joystick analógico, éste puede trabajarse como si fuese digital (esto porque pueden haber aplicaciones que funcionen mejor con dicha configuración). Es decir, es factible bajar de un analógico a un digital, pero no al revés. Este hecho va inclinando la balanza a favor de un joystick analógico. Un joystick analógico provee la ventaja de dividir el intervalo de voltaje en tantos niveles discretos como el proyecto lo requiera (y la resolución del convertidor analógico a digital provea); esto lleva a concluir que se puede incorporar el concepto de aceleración para controlar los motores del RCX. Sin embargo debido a los límites temporales (el manejo

requeriría significativamente más código) del presente proyecto, dicha opción no se incorpora como parte de la implementación del mismo.

c. Factibilidad de mantenimiento. mantener una configuración analógica será más sencillo que el contrario. Esto porque como ya se mencionó en el párrafo anterior, es posible en ciertos momentos trabajar el joystick analógico como digital y eventualmente utilizar el puerto de conversión analógico para otro tipo de sensor. Por último, una de las características que vale la pena mencionar en esta métrica, es que hay aplicaciones que se pueden ver beneficiadas de que el joystick analógico tiene tantas posiciones como resolución tenga el convertidor analógico digital, es decir para aplicaciones donde se necesita convertir la posición del joystick a aceleración, el joystick analógico es muy útil.

Después de revisar las métricas para el joystick y su consecuente efecto en el sistema como tal, el joystick analógico será el que se utilizará para este trabajo de graduación.

3. Versatilidad, escalabilidad, rendimiento y almacenamiento. A pesar que estas métricas se han venido mencionando en los párrafos anteriores, ha sido con un enfoque diferente. Ahora se retoman, junto a otras, para realizar la decisión más importante: escoger el procesador o microcontrolador que va a ser el centro del proyecto. Este punto es el que requiere un análisis más detallado, porque aún una mala decisión en los aspectos anteriores, puede ser sobrellevada, si el controlador fue elegido acorde a los requerimientos y a la realidad del sistema.

La elección reside tanto en aspectos técnicos como no técnicos. Por ejemplo, si el lado técnico fuese el único a tomar en cuenta, simplemente se escogerá el procesador con mejores prestaciones, pero hay que balancearlo con otras métricas que van más allá del campo técnico.

Una de las métricas más difíciles de optimizar es el rendimiento, porque muchas veces tiende a confundirse con velocidad del reloj que mueve al procesador, o bien en la cantidad de instrucciones por segundo, pero esto puede variar de fabricante a fabricante y no por eso el rendimiento tiene que verse afectado.

La primera decisión en esta categoría es escoger entre un microcontrolador y un microprocesador. Recordando la definición de microcontrolador expuesta en el marco teórico, es un chip altamente integrado que incluye en una sola pastilla (chip) todas o casi todas las partes para formar un controlador para un sistema dado. No hace falta analizarlo mucho: incluye todo lo que se necesita (de periféricos), lo cual optimiza en gran manera la métrica del costo. De hecho algunos microcontroladores vienen con los periféricos expuestos anteriormente:

- Módulo RS232
- Convertidor Analógico a Digital

Ahora bien, en la práctica para decidir si el controlador principal de cierto sistema será un microcontrolador o bien un microprocesador, es ver cómo será la interacción con el usuario: Si el sistema necesita un teclado matricial con muchas teclas, y un medio de presentación mayor a unas pocas líneas en LCD (es decir una pantalla formal que pueda desplegar gráficamente), entonces el controlador de nuestra aplicación es lo suficientemente general para que sea un microprocesador.

Sin embargo, si el sistema necesita poca interacción con el usuario y primordialmente va a tomar sus decisiones en base a su entorno (es decir en base a sensores que capturen de alguna manera la realidad) entonces lo mas probable sea que el mejor candidato sea un microcontrolador. Ahora bien, eso no quiere decir que el sistema no vaya a tener interacción con un

usuario o que no pueda tener teclado o pantalla (más bien estos son muy básicos y pequeños y son utilizados con poca frecuencia).

Por ende se concluye que lo más adecuado para el sistema que se está desarrollando, es un microcontrolador, ya que la mayor parte de decisiones se van a tomar de acuerdo a la lectura del joystick y en futuras implementaciones de sensores conectados al controlador, es por ello que tomar un microcontrolador que tenga incorporados muchos de los periféricos desde fabricación va optimizar las métricas de tiempo de prototipo y tiempo de mercado.

El único aspecto que hace falta determinar es, si ha de utilizarse un controlador con bastante memoria, tanto de programa como de datos. Debido a los límites temporales del proyecto, se utiliza un enfoque heurístico, asumiendo que, con un controlador con más de 4Kb de memoria de programa y, por lo menos 300 bytes de RAM, no será necesario utilizar memoria externa en esta fase del proyecto. Esto debido a que luego de observar los encabezados y cuerpos de mensajes que se necesitan enviar al RCX™, las anteriores especificaciones las satisfacen sin ningún problema.

Incluso para generar tablas de conversión analógica digital, basta con unos pocos bytes de memoria. Si en futuras implementaciones es necesario utilizar almacenamiento externo se puede agregar sin interferir con la propuesta actual. Es decir si se agrega un módulo de software más este puede almacenar sus datos en esa sección extra.

E. SELECCIÓN DEL MICROCONTROLADOR COMERCIAL

El fabricante escogido es **Microchip, Inc.** el cual fabrica controladores de 8 y hasta 16 bits. El tipo de controlador que Microchip fabrica, es RISC (Reduced Instruction set Computer). La razón primordial observada es el bajo precio de los controladores contra la cantidad de millones de instrucciones por segundo (por sus siglas en ingés MIPS), contra otros fabricantes tales

como Zilog (con su gama de controladores Z80), Rabbit, Hittachi y otros los cuales si bien ofrecen buen rendimiento y otras prestaciones de hardware, el precio se ve penalizado.

Se desea sobre-dimensionar en cierta medida las prestaciones del controlador contra los requerimientos de software, esto para permitir que en futuro se pueda reutilizar la infraestructura actual para crear un control más complejo. Esta consideración se tomará en cuenta para el momento de elegir el controlador.

Por lo anterior, se propone el siguiente controlador, como parte central del sistema inmerso a construir: Microchip PIC16F877. Dicho controlador cuenta con las siguientes características

-Arquitectura RISC

-Sólo 35 instrucciones de núcleo

-Instrucciones de un ciclo, con excepción de los saltos

-Velocidad de operación hasta de 20 Mhz (ciclos de 200 ns)

-Memoria de programa de 8K por 14 bits (palabra de programa) tipo FLASH

-Memoria RAM de 368 bytes

-256 bytes de memoria estática EEPROM

-14 fuentes distintas de interrupciones (entre ellas RS232, entradas digitales)

-Pila de 8 posiciones

-Modos directos, indirectos y relativos de direccionamiento

-*Cronómetro de monitoreo*

-Resguardo de energía con capacidad de “dormir” el procesador

-Convertidor Analógico Digital, de 8 canales.

-Módulo PWM

-Puerto I2C en modo maestro esclavo.

Hay otras opciones dentro de la gama que ofrece el mismo fabricante, tales como el PIC18F452, PIC18F2525, los cuales ambos superan en rendimiento (y prestaciones de hardware) al controlador seleccionado, sin embargo, como primera opción heurística, se optimiza el precio del controlador, eligiendo así el PIC16F877.

F. Implementación del prototipo

Una vez decidido el controlador y los periféricos a utilizarse, se presenta el siguiente circuito:

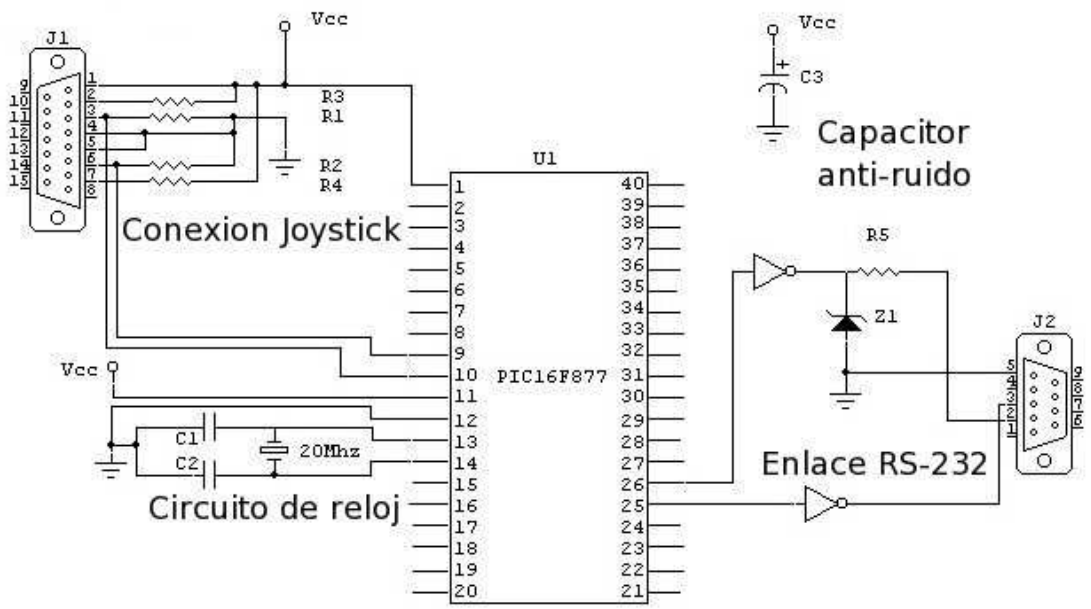

Figura 13: Circuito propuesto para el sistema inmerso de este proyecto

El prototipo fue implementado utilizando el lenguaje ensamblador de Microchip, llamado MPASM. Se utilizó codificación absoluta (es decir todo el firmware estaba en un único archivo).

Se utilizó un enfoque orientado a interrupciones. Esto con el fin de optimizar la métrica de consumo de Potencia para una futura implementación. Al inicio el controlador ejecuta las siguientes rutinas de configuración:

- Limpieza de los bancos de memoria RAM
- Configuración general de los registros de opciones y enmascaramiento de interrupciones
- Configuración del módulo RS232 (iniciar el generador de generador de tiempos, configuración del bit de paridad).
- Configuración del módulo PWM (aunque en la presente implementación no se utilizó para nada en particular, dicho módulo queda listo para futuros usos).
- Configuración del RCX™ para trabajar en modo de largo alcance (para que envíe con mayor potencia).
- Inicio del convertidor analógico / digital

Luego de esto, la rutina principal únicamente se mantendrá convirtiendo los valores de voltaje de los canales digitales 7 y 6 que corresponden precisamente a los ejes X y Y del Joystick respectivamente.

El concepto de interrupción es básicamente eso, una interrupción. El procesador tiene un flujo de trabajo continuo (ver Figura 16). La interrupción es reactiva; eso quiere decir que únicamente va a ocurrir cuando un evento de lugar a ello. Ejemplos de eventos son la transmisión serial, el vencimiento de un cronómetro del procesador, un evento digital y otros. Cuando dicho evento ocurre, el procesador va a dejar de hacer lo que sea que esté haciendo e irá a atender la interrupción (ejecutar el código para dicha

interrupción). Para el caso específico de este proyecto, en el manejo de las interrupciones

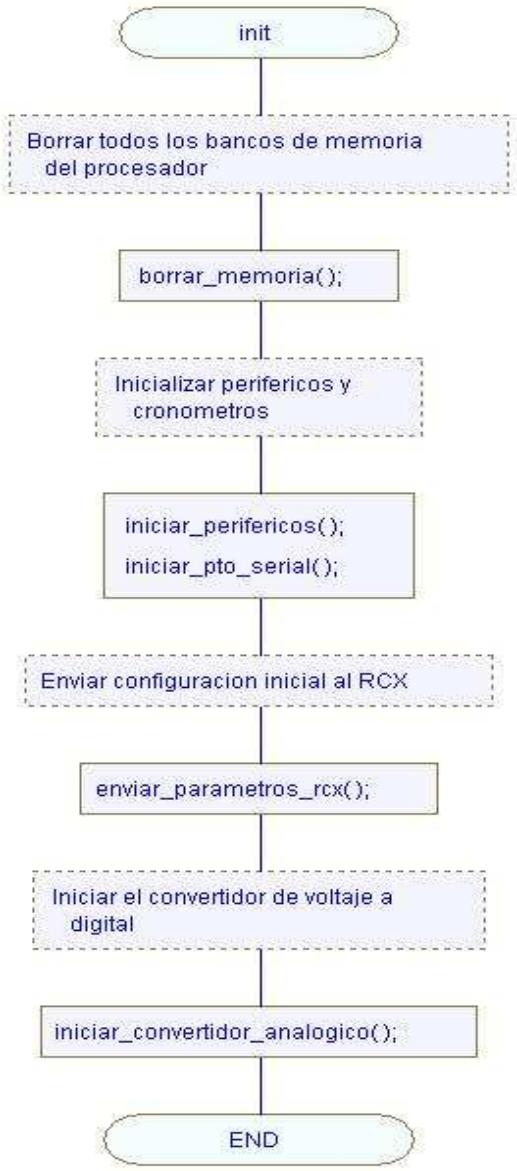

Figura 14: Algoritmo de inicio del programa para el sistema inmerso.

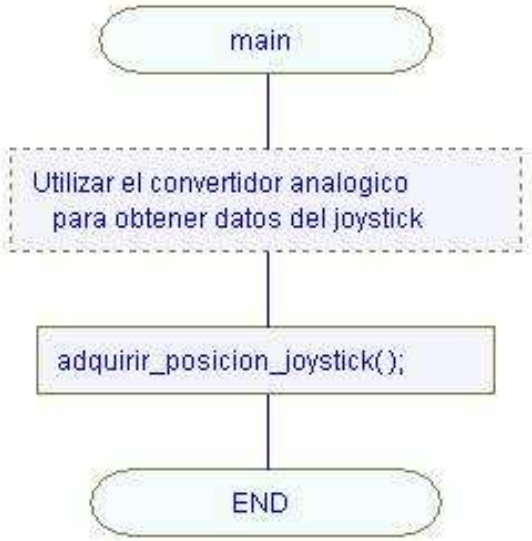

Figura 15: Algoritmo de ciclo principal para el sistema

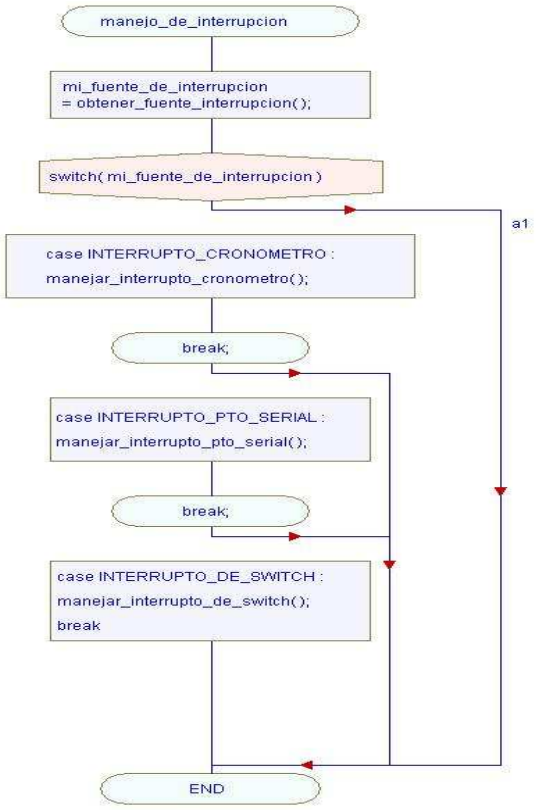

Figura 16: Diagrama de flujo de una interrupción para la presente aplicación.

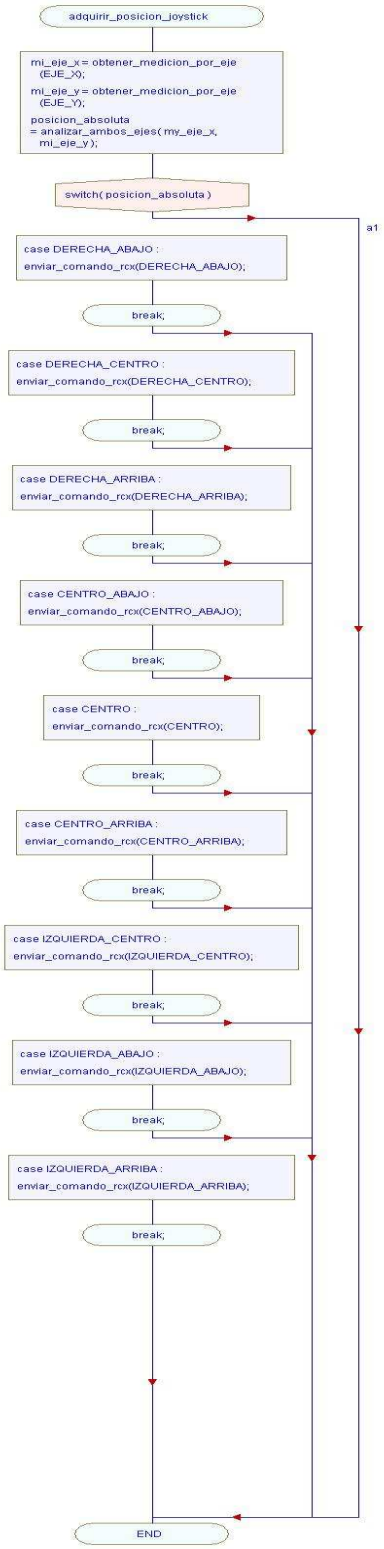

Figura 17: Diagrama de flujo para análisis combinado de las medidas del Joystick

1. Construcción del mensaje a ser enviado al RCX™. Estos se almacenan a partir de la posición 0x500 de la palabra de programa y cada byte ocupa una posición de 14 bits. A continuación un ejemplo de el mensaje para establecer largo alcance en el RCX™:

```
LONG_RANGEA DW 0X31,0XCE,0x01,0xFE,0x32,0xCD,'$'
```

El último símbolo le sirve a la rutina de transmisión serial para saber cuando un comando ha terminado.

Luego que se han digitalizando ambos canales y se tienen sus valores nominales, se procede a unificar la posición. Esto quiere decir que si el joystick está arriba / izquierda, esto va a tener una única posición, que se va a construir así:

XXXX YYYY

En donde XXXX son los 4 bits más significativos de la conversión analógica digital del canal de posición X y YYYY son sus opuestos del canal de posición Y.

2. Algunos problemas enfrentados durante el desarrollo. Un factor que frena la velocidad de desarrollo para una plataforma inmersa, es la metodología que se debe emplear cuando no se cuenta con el equipo necesario (emuladores de plataforma e interconexión con el ordenador) para buscar errores en la programación. Con el fin de ayudar a entender este concepto, considérese cuando se desarrolla un programa que esta destinado a ejecutarse en un ordenador: en comparación con un sistema inmerso típico, el ordenador cuenta con recursos de procesamiento y despliegue infinitos, de tal manera que el programador puede detener la ejecución en cualquier punto del programa y observar todo lo relacionado con su programa (el estado de las variables, los valores de los registros del CPU e incluso

tener acceso a información que ocurrió antes de que el programa fuese detenido.

La desventaja anterior se acentúa cuando se está desarrollando en lenguaje ensamblador, el cual no permite al programador elevar el nivel de abstracción cuando se construye el programa; también es un hecho que en el lenguaje ensamblador la posibilidad de cometer errores es mucho mayor que cuando se utiliza un lenguaje de tercera generación como lo es el lenguaje C.

La carencia de un osciloscopio para observar señales de voltaje también impuso una penalización en tiempo de desarrollo. Esto es porque el propósito del programa que se ejecuta en el sistema inmerso, es trabajar de una u otra manera con pines de entrada o salida (en modo digital o analógico) del microcontrolador.

G. Análisis de requerimientos sin perspectiva global ni de integración

Esta sección presenta diversas discusiones acerca de posibles enfoques de diseño. La mayoría nunca se logró incorporar al proyecto, sino que más bien sirvió de herramienta para llegar al diseño presentado en las secciones anteriores. Es decisión del lector continuar o no (no agregan valor para entender el concepto del proyecto, más bien fueron un puente para el diseñador). En este punto se plantearán diversos enfoques para poder satisfacer cada uno de los requerimientos, sin tomar en cuenta aspectos como la integración total, eficiencia total o costo. Se desea poder contar con varias opciones para la siguiente sección, la cual se centrará precisamente en aspectos de integración y eficiencia desde una perspectiva general.

Es indispensable tomar un patrón de diseño que vaya de lo específico a lo general. Esto porque la última decisión a tomar va a ser el tipo de microcontrolador / microprocesador a utilizarse y es en base a los módulos o características por separado.

1. Comunicación RS – 232.

a. Implementación manual. Lo que se propone aquí es tomar un microcontrolador o microprocesador e implementar las especificaciones de tiempo y chequeo de errores como parte de la programación. En otras palabras, es tarea del programador estar capturando las líneas e ir concatenando bit a bit las palabras que vienen. Si existiese paridad también debe ser calculado en software. Es de mencionar que se debe conocer la frecuencia de reloj que utiliza el microcontrolador / microprocesador ya que el tiempo que ejecuta una instrucción dada depende de ello y por ende el control de tiempo del módulo.

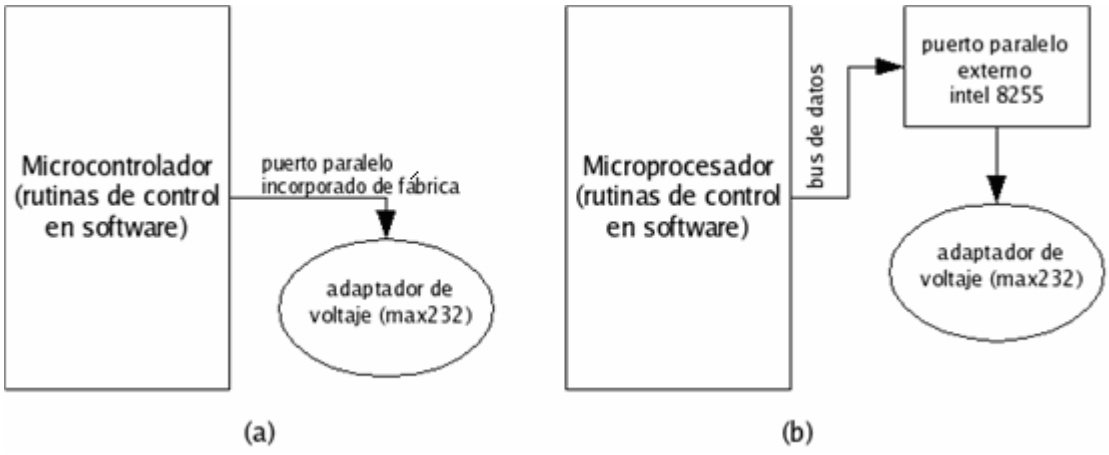

Figura 18: a) Implementación manual del puerto RS-232 en un microcontrolador b) Implementación manual del puerto RS-232 en un microprocesador

2. Un microcontrolador que tenga el módulo incorporado. En este enfoque, el módulo en hardware viene incorporado desde la fabricación del controlador. El programador no tiene que estar consciente de qué dirección tiene el puerto serial en el bus de datos. Y lo más probable sea que dicho módulo tenga también incorporada una interrupción y la tarea del programador se concentra en atender dicha interrupción.

3. Un microprocesador con un puerto RS232 conectado externamente. A diferencia de la propuesta anterior, aquí se tiene un módulo de RS232 (como lo sería el SIO Z80) y está conectado al bus de datos del procesador y este lo lee y escribe como que si fuera cualquier posición de memoria. Aquí hay que implementar manualmente la interrupción del módulo al procesador. En cuanto a programación no habría mayor diferencia respecto al anterior enfoque, sin embargo toda la configuración del módulo hay que hacerla según el fabricante, por medio de escribirle datos al módulo (generador de tiempos para el puerto serial, bits de parada, paridad, etc.)

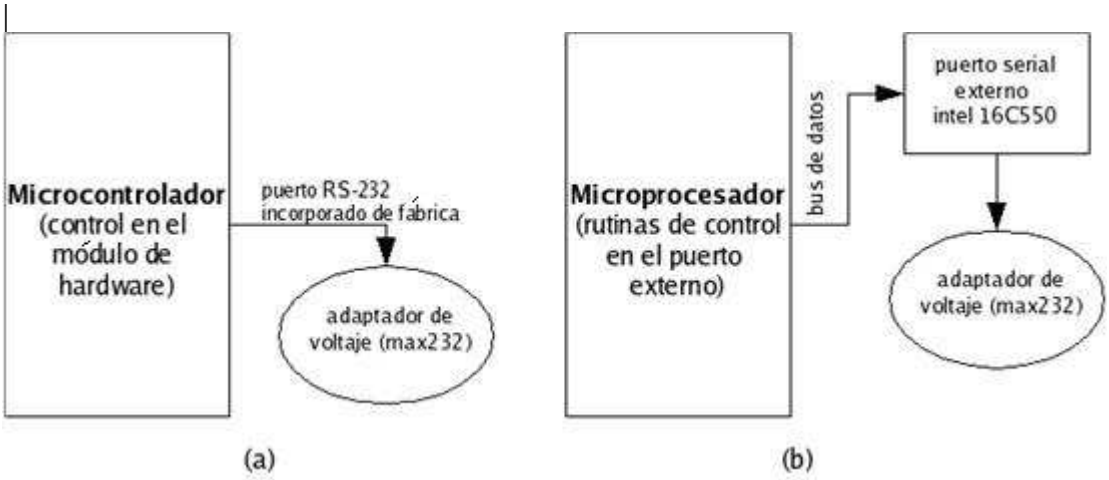

Figura 19: a) Un microcontrolador con un puerto RS-232 de fábrica b) Un microprocesador con un puerto RS-232 externo. En ambos casos la lógica y el control de tiempo residen en el módulo y el controlador / procesador no está consciente del protocolo RS-232

2. Medio de captura versátil. Por medio de captura versátil se entiende la forma en la que el usuario va a controlar el RCX™. De alguna manera el sistema inmerso debe poder interpretar lo que el usuario desea hacer, transformarlo en uno (o varios) paquete(s) construidos de una forma dinámica y enviarlo al RCX™. El primer dispositivo que sale a considerar es el joystick, ya que provee al usuario con una interfaz mecánica intuitiva, y sobre todo una manera fácil de interpretarlo por medio de una conversión analógica (con la excepción de los joystick digitales).

Figura 20: Un joystick analógico para la computadora

Como se mencionó en la sección de Marco Teórico es tarea de la computadora realizar la conversión análoga / digital. Existe una alternativa de Joystick, llamado Joystick analógico que solía venir con las primeras consolas de juegos de video (como el atari 2600) o los que traían las primeras consolas de computadora como la Commodore 64. La siguiente tabla compara ambos joysticks.

	Joystick tipo Atari (digital)	**Joystick para PC analógico**
Tecnología	5 interruptores	2 resistencias variables y 2 interruptores
Tipo de conector	D9 hembra, con alimentación de 5V para autofuego opcional	D15 macho. 5V necesariamente para alimentación
Cantidad de direcciones	8	Ilimitadas
Auto-colocación	sí	usualmente ajustable
Botones	1 (opcional 2 o 3)	2
Barra por controlador	1	1 ó 2 (requiere cable y)
Calibración	no	Sí
Duración del dispositivo	buena	promedio (dura menos que el digital)

	Joystick tipo Atari (digital)	**Joystick para PC analógico**
Limitaciones		
Otros usos para el conector (DB9/15)	Mouse, keypad, lightpen	Dispositivos para simulación (timón), pedales
Aspectos de programación		
Método de lectura	Un comando en programación	Una rutina especial
Tiempo de lectura / adquisición	< 1 us	1 .. 2 ms
Método de lectura de botón	Un comando de programación	Un comando de programación
Tiempo de lectura / adquisición del estado del botón	< 1 us	< 1 us
Adecuación para juegos		
Rendimiento en simulación de vuelo	mala	muy buena
Simulación de manejo	promedio	Buena
Boton que cumple con 2 funciones (toma y salta)	buena	Promedio
Boton que cumple con 2 funciones (dispara y salta)	buena	buena a mala dependiendo del juego

Tabla 4: Comparación entre joystick analógico y digital

a. Digitalizador externo conectado a un microprocesador. Esta opción asume que se elige el joystick analógico. A partir de esto, de alguna manera se debe de digitalizar dicha información. Es por ello que este enfoque propone utilizar un microprocesador de uso general con un puerto

de conversión analógica conectado a su bus de datos. Esto también presupone más de alguna carga del lado de programación ya que se debe de establecer la conexión así como el hardware auxiliar que utiliza el dispositivo externo.

b. Un microcontrolador que tenga el módulo incorporado. Aquí se utiliza un microcontrolador de aplicación específica que tenga internamente conectado un convertidor analógico / digital.

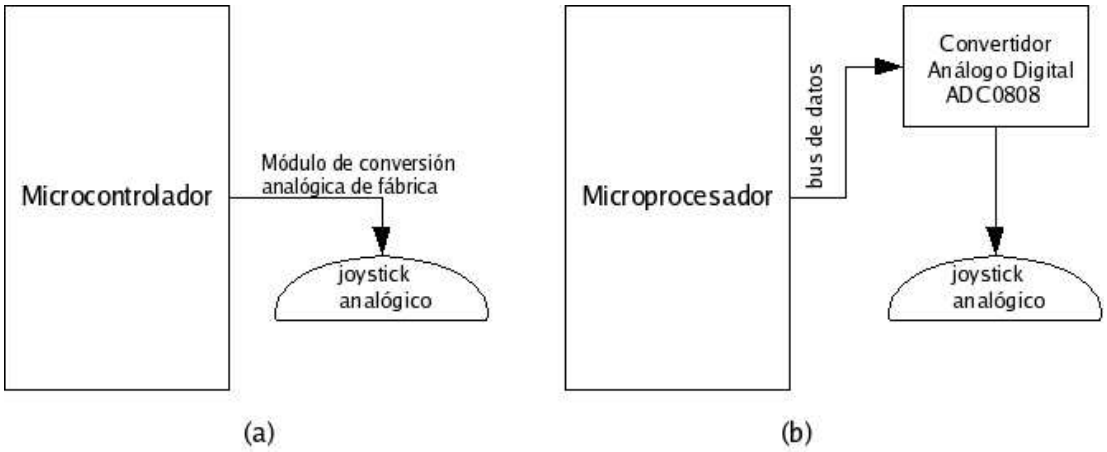

Figura 21: a) Un microcontrolador con un módulo ADC de fábrica b) Un microprocesador con un puerto convertidor ADC externo.

c. Un joystick digital que no requiera conversión analógica. Esta propuesta es leer en un puerto digital la posición que un joystick igualmente digital indica. Para esto, si es un microcontrolador, se debe utilizar un puerto digital incorporado desde fabricación y si es un microprocesador, se debe utilizar un chip de puerto paralelo externo como el PIO Z80, o el Intel 8051.

3. Versatilidad, escalabilidad, rendimiento y almacenamiento. Cuando se habla de escalabilidad, se quiere decir que en un futuro cuando se desee hacer que el sistema inmerso tenga más poder o realice operaciones más complicadas que las que la especificación de este trabajo de graduación dicta, no haya necesidad de re-diseñar la especificación de hardware, sino

únicamente trabajar en la capa de software, ya que eso es económico, al contrario del re-diseño físico. Otra forma de decirlo, es que se pretende que la especificación de hardware esté por encima de las demandas de software actual, con el fin de dar lugar a futuras implementaciones.

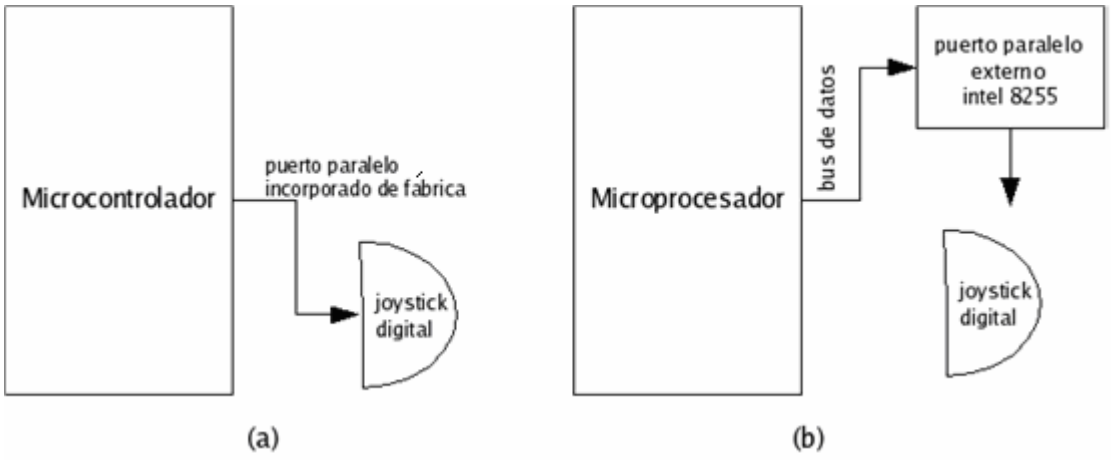

Figura 22: a) Un microcontrolador con un joystick digital conectado a un puerto digital desde fabricación b) Un microprocesador con un puerto digital, el cual a su vez tiene conectado el joystick

a. Un microprocesador con memoria externa. Este es el esquema de funcionamiento más sencillo y actualmente el más utilizado. Básicamente se conecta un rango de memoria al microprocesador y se accede posición por posición en un rango dado. Dependiendo de la arquitectura, si es compatible con Hardvard o Princeton así será el direccionamiento. Supongamos por simplicidad que vamos a utilizar la arquitectura de Hardvard.

V. CONCLUSIONES

A. El uso de la arquitectura propuesta para este prototipo permite agrandar los límites de control de un robot implementado con Lego® Mindstorms™ , ya que libera al robot de la necesidad de la computadora, cuando se trabaja en modo “maestro / esclavo”.

B. Debido a la división del Hardware y su integración con las rutinas de software, las distintas partes del prototipo pueden ser fácilmente modificadas sin afectar a otras.

C. Esta arquitectura permite la exploración de nuevos prototipos de robots y aplicaciones.

D. El Lego® Mindstorms™ es un ambiente idóneo para trabajo a nivel universitario en áreas como: Robótica, Sistemas Inmersos, Sistemas Operativos, Mecatrónica y otras.

E. La división de módulos de este sistema (escalabilidad) le da una ventaja al presente prototipo, esto con el fin de adaptarlo para diferentes aplicaciones.

VI. RECOMENDACIONES

A. El protocolo del RCX™ puede ser ampliado. Es decir no es lo suficientemente confiable, ya que por diseño no es orientado a conexión, pero puede mejorarse si se combinan ambos modos de funcionamiento en el RCX™: el modo independiente y el “maestro / esclavo”. Se propone como trabajo futuro, utilizar lenguajes de alto nivel (que tanto el sistema inmerso diseñado, como el RCX soportan) para crear una capa de conexión, que garantize la distribución de los paquetes. Para esto se puede sacar ventaja de los lenguajes de programación alternos como lo son Java o C++.

B. Uno de los posibles usos de los módulos del PIC16F877 es el de I2C. Esto da lugar a crear una red de controladores que esté diseminada por una área geográfica delimitada; entonces, esta red de controladores puede mantener informado al RCX™ acerca de condiciones ambientales y este puede tomar decisiones en base a esto, o bien el controlador maestro de la red puede hacerlo y enviar el comando resultante al RCX™. Un ejemplo idóneo de esto es una simulación de un vehículo tipo Pathfinder el cual puede recibir información de entidades externas para tomar decisiones. Se recomienda, para una continuación de este proyecto conectar más sensores (al sistema inmerso desarrollado por este trabajo de graduación) a través del protocolo I2C.

C. El RCX posee tres puertos para sensores únicamente, lo cual limita la información que se puede obtener de la realidad para tomar decisiones: esto es conocido como alimentación retroactiva. Un área para futuros proyectos

consiste en investigar como expandir el nivel de alimentación retroactiva que el RCX puede proveer, cuando trabaja en modo de maestro/esclavo, siendo una computadora la que toma las decisiones. Esto da lugar a dos posibilidades: la primera es simplemente colocar sensores trabajando independientemente el uno del otro, en cierta área geográfica. La segunda opción es estudiar cómo hacer para compartir los puertos que el RCX posee a lo largo de un período de tiempo determinado (técnica conocida en inglés como "multiplexing").

D. El presente sistema inmerso tiene la infraestructura para poder tomar decisiones el solo, es decir, no depender del usuario para que maneje el joystick. Con base a la recomendación B, se propone, que una vez el sistema inmerso tenga suficientes sensores en un área geográfica, agregar “inteligencia” al sistema inmerso para que tome decisiones el solo, sin depender del usuario, que es como fue planteado en el presente trabajo.

E. La información entre el ordenador y el RCX se trasmite por medio de ondas infrarrojas, lo cual presenta algunas desventajas; la comunicación puede ser fácilmente interrumpida cuando un obstáculo físico está en el medio de ambos sistemas. Se recomienda experimentar con tecnologías que tengan más tolerancia a dicho problema, tal como "Bluetooth" y "WI-FI" (por sus designaciones en inglés).

VII. REFERENCIAS BIBLIOGRÁFICAS

Baum, Dave; 1998. «**Lego Mindstorms™ Protocol [RE: Lego Mindstorms™ Page**» [Comunicación Electronica], http://graphics.stanford.edu/~kekoa/RCX™/protocol.html

Bies, Lammert; 2001. «**RS232 Specifications**» [WWW document], URL http://www.lammertbies.nl/comm/info/RS-232_specs.html

Brain, Marshall; 2004. «**How Microprocessors Work**» [WWW document], URL http://computer.howstuffworks.com/microprocessor.htm/printable

Dorf, Richard; Bishop, Robert; 1998. «***Modern Control Systems***» *8a. Edición, Addison Wesley Longman.* 855P.

Engdal, Tomi; 1998. «**Joysticks and Other Games Controllers**» [WWW document], URL, http://www.epanorama.net/documents/joystick/index.html

Ferrari, Giulio, Hilmer, Soren, et al. 2002. «***Programming Lego® Mindstorms™ with Java***» *Rockland Massachusetts.* 441p.

Gawthrop, Peter; McGookin, Euan; 2004. «***A LEGO-Based Control Experiment***» I*EEE Control Systems Magazine.* V24(I 5). P 43 – 56.

Hayes, Patrick. 2002. «***RoboCup Starter Kit***» *Proceedings of the RoboCup tournament*. P 1-40

Heck, Bonnie; Clements, Scott; Ferri, Aldo; 2004. «***A LEGO Experiment for Embedded Control System Design***» I*EEE Control Systems Magazine.* V24 (I 5). P 61 – 64.

Hersch, Russ; 1997. «**Embedded Processor and Microcontroller primer and FAQ**» [Comunicación Electronica], http://www.faqs.org/faqs/microcontroller-faq/primer/

Jucknath, Susanne. 2001. «***Experiences with using LEGO Mindstorms™ for Teaching***» *Informatica Feminale in Bremen.* P 3-4

Li, Qing; Yao, Caroline; 2003. «***Real-Time Concepts for Embedded Systems***» *1era Edición, CMP Books.* 294P.

Mateyan, Vicente; González Jesús et. al 2000. «***Programación de LEGO Mindstorms™ bajo GNU/Linux***» [WWW Document] *III Congreso HispaLinux virtual.*

URL http://es.tldp.org/Presentaciones/200002hispalinux/conf-16/16-html/programacion_LEGO_Mindstorm.html

Mientki, Stef; 2001. «**Mindstorms™ IR-communication**» [WWW document], URL http://oase.uci.kun.nl/~mientki/Lego_Knex/Lego_electronica/IR_tower/IR_tower.htm

Nielsson, Stig. 2000. «***Introduction to the legOS kernel***» *Main document for the legOS development team.* p 1-14

Nikander, Peka; 2000. «***An Operating System in Java for the Lego Mindstorms™ RCX™ Microcontroller***» *Proceedings of freenix track: Usenix Annual Technical Conference.* P 1-15.

Noga, Markus; 1999. «**about legOS**» [WWW document], URL http://www.noga.de/legOS/

Proudfoot, Kekoa. *1998.* «***RCX™ Internals***» *Main reverse engineering document.* P 1-40

Rieber, Jochen; Wehlan, Herbert; Allgöwer, Frank*; 2004. «**The ROBORACE Contest**» IEEE Control Systems Magazine. V24 (I 5). P 57 – 60.*

Russel, Nelson. *2004.* «***Mindstorms™ Internals***» *Main reverse engineering document.* P 1-48

Vahid, Frank; Givargis, Tony; 2002. «***Embedded System Design: A Unified Hardware / Software Introduction***» *1era Edición, John Wiley and Sons. 352*P.

Wallich, Paul; 2001. «***Mindstorms™, Not Just a Kid's toy***» I*EEE Spectrum Magazine.* V38(I 9). P 52 – 57.

Referencias web adicionales:

http://Mindstorms.lego.com

http://brickos.sourceforge.net

http://lejos.sourceforge.net

http://lugnet.com

APÉNDICE I GLOSARIO DE TÉRMINOS

La mayor parte de términos están cubiertos como parte del desarrollo, sin embargo hay algunos que son auxiliares, mas no se justifica incluir una definición como parte del contenido. Dichos términos se encuentran acá.

Bit: Unidad mínima de información utilización en lógica digital, dicha unidad es binaria en el sentido de que solo puede tener el valor de uno o cero.

Byte: Ocho bits, también llamado octeto.

EEPROM: Por sus siglas en inglés *Electrically Ereaseable Programable Read Only Memory*. Al igual que la memoria ROM la mayor parte del tiempo es de lectura únicamente, sin embargo, cuando se necesita borrar, es posible a través de un voltaje superior al voltaje de operación.

Firmware: un micro-programa diseñado para ser ejecutado en un sistema inmerso, generalmente almacenado en memoria ROM o EEPROM, más puede ser ejecutado desde RAM en casos especiales..

ISA: Por sus siglas en inglés *Industry Standard Architecture. Es una especificación de bus para* conectar diversos periféricos a una computadora.

Osciloscopio: Un dispositivo que produce imágenes de voltaje o corriente, en un intervalo definido.

ROM: Por sus siglas en inglés *Read Only Memory*, memoria de lectura únicamente.

APÉNDICE II CÓDIGO FUENTE DEL SISTEMA INMERSO