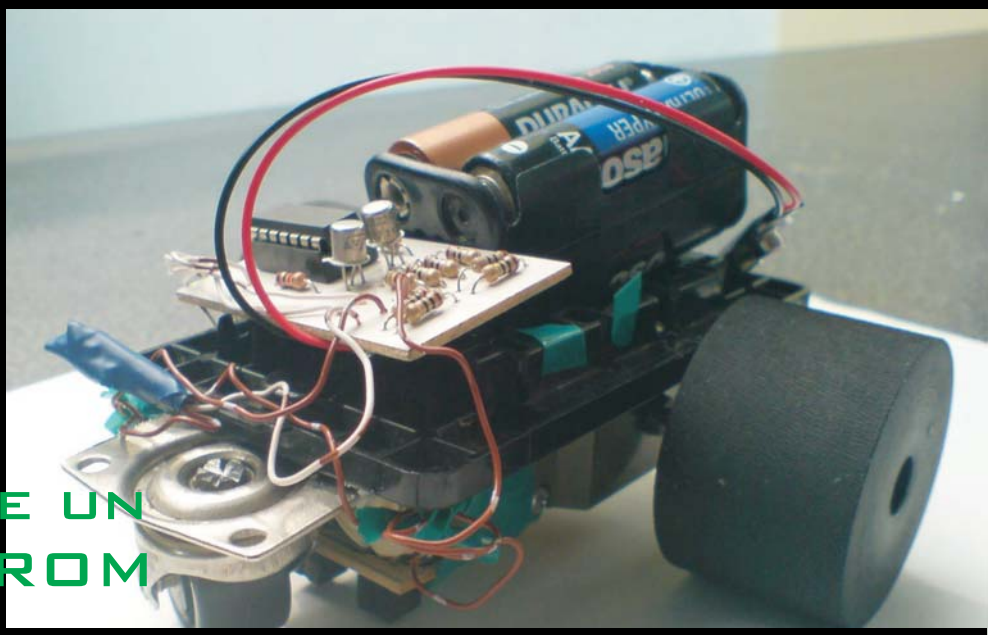
爆風ロボット

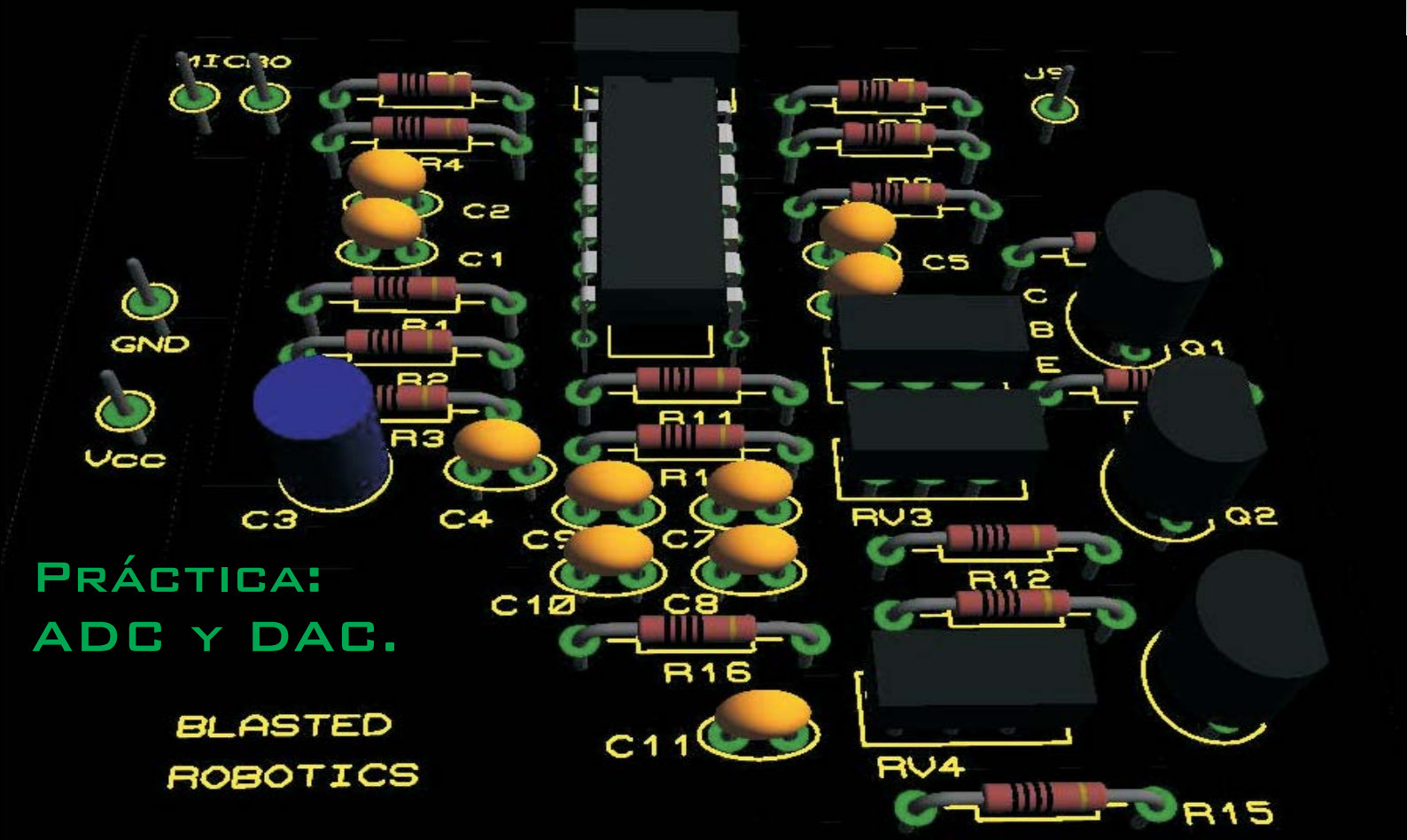
WIKI

Robot seguidor de líneas con NAND



Introducción a PICC de CCS

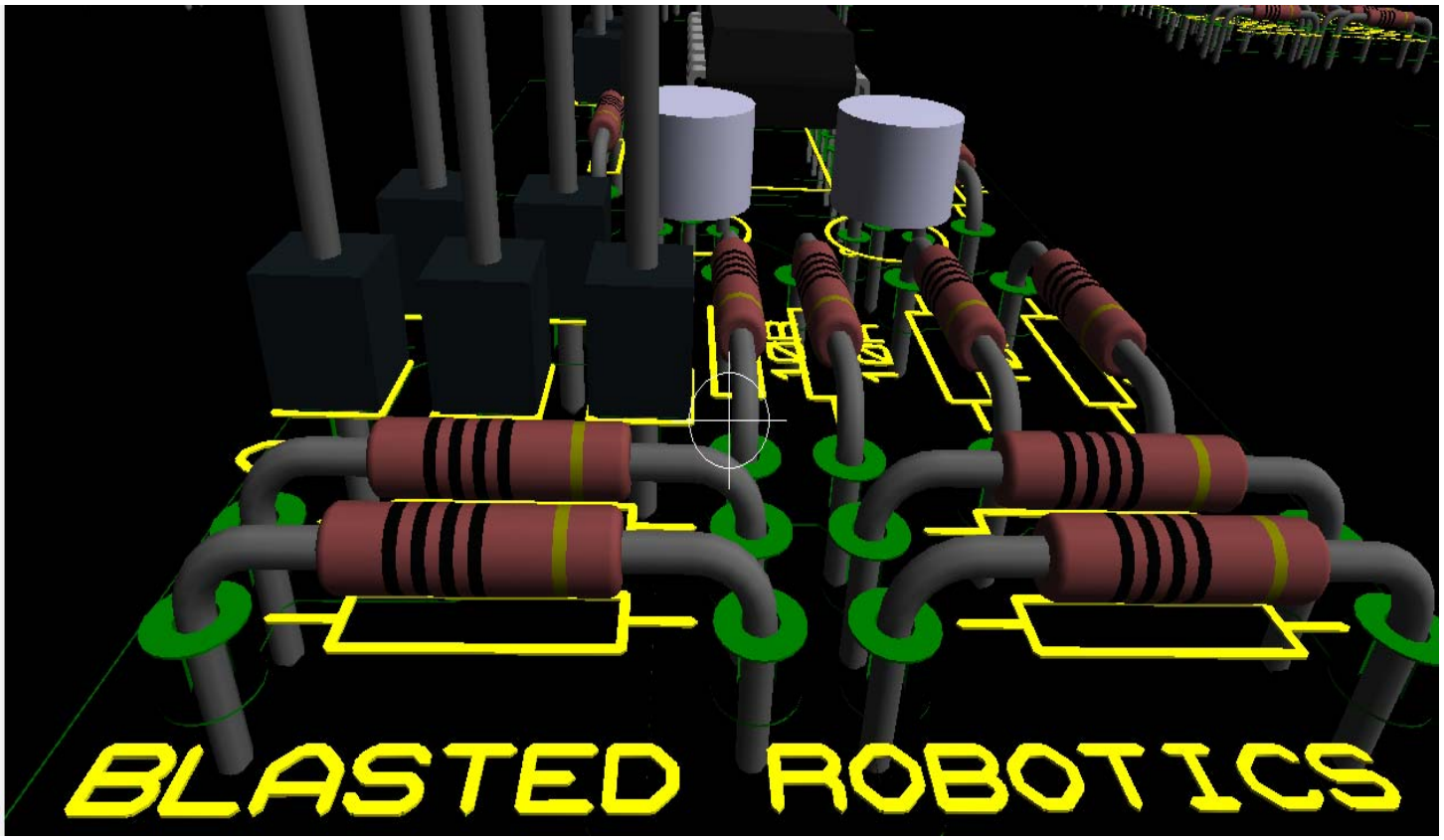
Programación de un dispositivo EEPROM



Práctica: ADC y DAC.

BLASTED ROBOTICS

Año 1 - Noviembre 2008 - Número 1


BLASTED ROBOTICS


爆風ロボット

爆風ロボット

Índice.

Publicación del grupo estudiantil

Blasted Robotics, de la FCE, BUAP, Puebla, Pue.

México.

Noviembre de 2008.

e-mail:

blastedrobotics@yahoo.com.mx

Edición:

García Chetla Raúl.

San Pablo Juárez Miguel Ángel.

Gonzales Vellano Pablo.

La revista tiene como finalidad publicar artículos hechos por la comunidad estudiantil de la Facultad de Ciencias de la Electrónica de la Benemérita Universidad Autónoma de Puebla, para apoyar a nuestros compañeros difundiendo un poco de la creatividad desarrollada y aplicada en el área de la electrónica, el lector podrá construir y probar los diferentes proyectos y prácticas descritas en esta publicación teniendo la certeza de que funcionarán al cien por ciento, todos los artículos han sido diseñados, innovados y probados en los laboratorios de nuestra universidad y otros varios en casa o lugares específicos de diseño en pasatiempo de algunos autores.

En este número 1 de nuestra revista nos aventuramos en el diseño y mejora de un robot seguidor de líneas construido con componentes baratos y lógica digital, después nos introducimos a algunos conceptos fundamentales para el diseño con el lenguaje C de los microcontroladores PIC, incluimos dos prácticas básicas, una de programación de una memoria y otra del armado de un circuito digital con compuertas, una práctica de laboratorio típica de convertidores analógico a digital y de digital a analógico, y finalizamos con un cuestionario que nos hace reflexionar sobre cuestiones de nuestro mundo físico.

Esperamos disfruten de los artículos y esperamos que les sea de utilidad, de antemano gracias.

El grupo estudiantil Blasted Robotics.

PROGRAMACIÓN DE UN DISPOSITIVO EEPROM (AT28HC64)

Aprender a programar un dispositivo EEPROM, con un respectivo código en base hexadecimal es sumamente fácil, en este caso una memoria EEPROM AT28C64, de 64k (8k x 8), un bus de 13 líneas de dirección y 8 de datos. Para esta práctica se toma como diseño un decodificador de BCD a 7 segmentos.

Material necesario:
1 EEPROM AT28C64 de 64k(8k x 8) de Atmel (ó equivalente)
1 Programador Chipmaster 7000
1 programador Chipmaster (Quemador)
1 Display de 7 segmentos cátodo común
1 Protoboard
8 Resistencias de 330ohm
Cables para el protoboard
Fuente de alimentación a 5 V
Caimanes

Tenemos el caso de hacer un decodificador con nuestra EEPROM de BCD a 7 segmentos como se muestra en la figura:

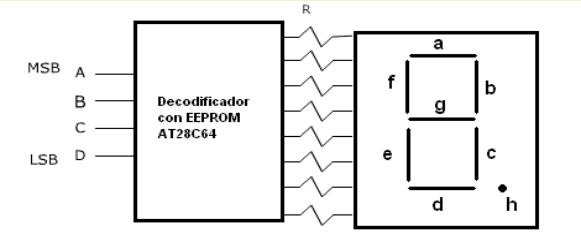
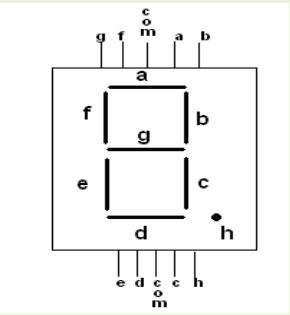


Figura 1. Decodificador de BCD a 7 Segmentos.

De acuerdo al diseño que se va a tomar en cuenta es necesario ver la relación que se necesita para la implementación del decodificador de 7 segmentos, es decir, ahora se presenta el diagrama de pines de un display cátodo común, para hacer la relación de leds que encenderán de acuerdo al código binario.

El diagrama del display es el siguiente:



Donde a hasta g representan una parte del número mientras que h representa el punto que trae el dislpay, el cual se mantendrá encendido todo el tiempo.

Los números a decodificar son del 0 al 15 decimal. Entonces hacemos una tabla de relación del código binario de estos números y luego de los leds que encenderán en nuestro display, por

ejemplo, para encender el 5dec= 0101bin debemos encender los segmentos a, c, d f y g y apagar los otros, entonces la relación sería: a=”1”, b=”0”, c=”1”, d=”1”, e=”0”, f=”1”, g=”1” h=”1”, donde los números entre comillas son niveles lógicos. Y así sucesivamente se llena la tabla para después transformar lo que nos queda a base hexadecimal, esto se hace agrupando de cuatro en cuatro bits y después transformar a su equivalente hexadecimal, donde el ejemplo anterior quedaría:

$5_{10} = 0101_2$ => $1011\ 0111_2 = B7_{16}$

Entonces nos queda la siguiente tabla.

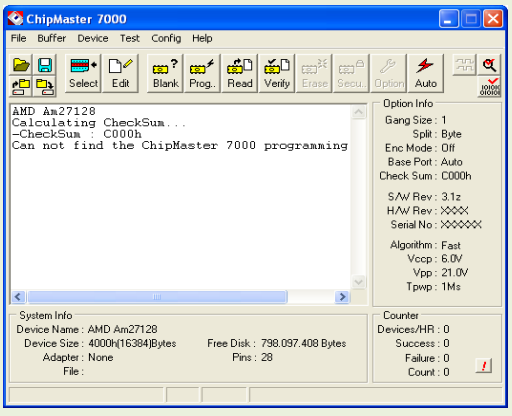
Dec	A	B	C	D	a	b	c	d	e	f	g	h	Hex
0	0	0	0	0	1	1	1	1	1	1	0	1	FD
1	0	0	0	1	0	1	1	0	0	0	0	1	61
2	0	0	1	0	1	1	0	1	1	0	1	1	DB
3	0	0	1	1	1	1	1	1	0	0	1	1	F3
4	0	1	0	0	0	1	1	0	0	1	1	1	67
5	0	1	0	1	1	0	1	1	0	1	1	1	B7
6	0	1	1	0	1	0	1	1	1	1	1	1	BF
7	0	1	1	1	1	1	1	0	0	0	0	1	E1
8	1	0	0	0	1	1	1	1	1	1	1	1	FF
9	1	0	0	1	1	1	1	1	0	1	1	1	F7
10	1	0	1	0	1	1	1	0	1	1	1	1	EF
11	1	0	1	1	0	0	1	1	1	1	1	1	3F
12	1	1	0	0	1	0	0	1	1	1	0	1	9D
13	1	1	0	1	0	1	1	1	1	0	1	1	7B
14	1	1	1	0	1	0	0	1	1	1	1	1	9F
15	1	1	1	1	1	0	0	0	1	1	1	1	8F

Para los números 16, 17, 18, … ,8191 no decodificaremos nada, en una memoria EPROM eso quiere decir que dejaremos sólo “1” lógicos programados. Esto también quiere decir que trabajaremos sólo con los bits A3, A2, A1, y A0 donde A12 al A4 deberán estar conectados a “0” ya que la tabla anterior funciona cuando estos bits son precisamente “0” lógicos ya que si no es así estaremos trabajando en otra dirección y nuestro decodificador nos arrojará “1” lógicos.

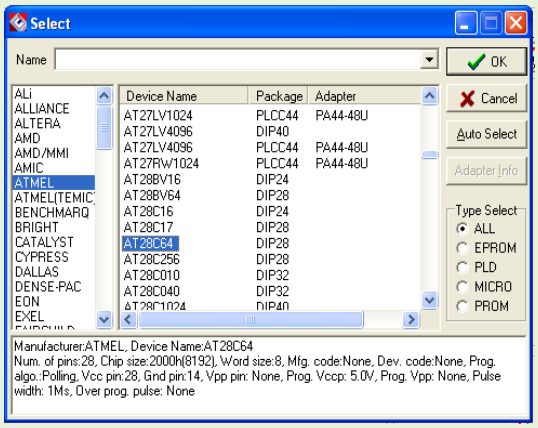
Ahora bien, solamente resta ingresar estos datos en hexadecimal que obtuvimos en nuestra tabla a nuestra EEPROM, para hacer esto usamos el programa de computadora llamado Chipmaster 7000, resaltando que esto se puede hacer en cualquier otro programador como Icprog o incluso desde C++.

Para el programado abrimos el chipmaster 7000 que debe estar previamente instalado en nuestra PC, en caso contrario lo instalamos primero ya que lo podemos descargar desde Internet.

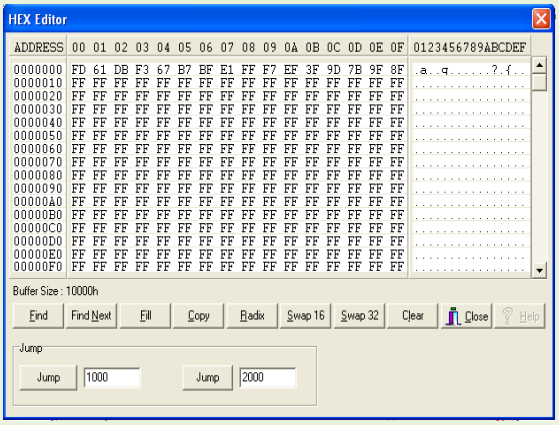
El ambiente del programa es el siguiente:



Hacemos clic en Select, y desde ahí seleccionamos en este caso AT28C64 y damos clic en ok.



Después clic en Edit e ingresamos el código hexadecimal que tenemos de izquierda a derecha en la tabla que aparece y damos clic en close.



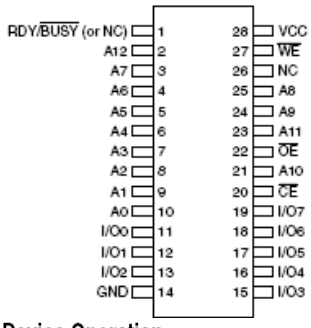
Finalmente clic en Prog y ya está. Cabe mencionar que el programador (quemador) chipmaster debe de estar ya conectado y encendido en la misma PC que utilizamos, con nuestra EEPROM ya colocada en él, el quemador es un poco caro pero podemos guardar nuestro archivo dando clic en file y luego save e ir a algún lugar donde lo tengan como una tienda de electrónica o una universidad como lo fue en este caso; abrir el archivo dando clic en load y programarlo.

Finalmente con nuestro dispositivo ya programado hay que probarlo, se arma el circuito que se muestra al inicio de este reporte y se alimenta con 5VDC que es lo que marca la hoja de especificación. De acuerdo a la hoja de especificación el diagrama de pines es el siguiente:

Pin Configurations

Pin Name	Function
A0 - A12	Addresses
$\overline{CE}$	Chip Enable
$\overline{OE}$	Output Enable
$\overline{WE}$	Write Enable
I/O0 - I/O7	Data Inputs/Outputs
RDY/$\overline{BUSY}$	Ready/$\overline{Busy}$ Output
NC	No Connect
DC	Don't Connect

PDIP, SOIC
Top View



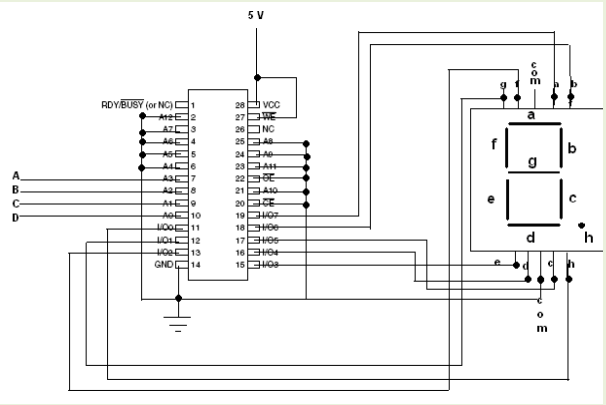
Device Operation

READ: The AT28C64 is accessed like a Static RAM. When $\overline{CE}$ and $\overline{OE}$ are low and $\overline{WE}$ is high, the data stored at the memory location determined by the address pins is asserted on the outputs. The outputs are put in a high impedance state whenever $\overline{CE}$ or $\overline{OE}$ is high. This dual line control gives designers increased flexibility in preventing bus contention.

Esto quiere decir que para leer la memoria los pines 20, 22 y 14 (GND) se conectan a tierra (0 VDC); el pin 1 y 26 no se conectan, el pin 28 y el 27 se conecta a 5 VDC. Y como se había pensado los pines 2, 3, 4, 5,

6, 25, 24, 23, 21 que son A12-A4 van conectados a GND. Luego los pines 7, 8, 9, 10 que son A3-A0 serán las direcciones que ocupemos para ingresar nuestro número en binario. Y los pines 11, 12, 13, 15, 16, 17, 18, 19 serán nuestras salidas h, g, f, e, d, c, b, a respectivamente ya que en la tabla podemos ver que el menos significativo es h y el más es a; estas salidas las conectamos a una resistencia de 330ohms cada una y el otro extremo al display que al inicio del reporte puede mostrar su diagrama de pines, donde com es el cátodo común y va conectado a GND. Todo esto es montado en un protoboard haciendo los puentes con cable telefónico.

El diagrama del circuito final es este



Observaciones

La programación de esta EEPROM sólo fue para aprender a programar este tipo de dispositivos, por ello sólo usamos pocas direcciones pero el potencial de un dispositivo como este es muy poderoso ya que podemos hacer decodificaciones más grandes, y no nada más con este modelo, podemos usar otros como los de la familia de AM27CXXX, o los de la familia AT28CXXX ó AT28HCXXX. Esto ya depende del uso que le queramos dar.

Una observación importante es que si no se conectan a cero las direcciones que no usamos los datos de salida serán siempre todos unos, es también importante apreciar que en algunos dispositivos el diagrama de pines es parecido pero no es igual, ya que en algunos traen ciertos habilitadores y en otros no, hay que ver bien la hoja de especificación en la parte que dice READ para ver cómo conectar el chip con sus respectivos habilitadores, ya que si es conectado mal el dispositivo no funcionará.

Programar un dispositivo de este tipo como lo es una EPROM ó EEPROM, es demasiado fácil, pero se torna difícil cuando no se tiene el conocimiento y la práctica de cómo hacerlo, claro que también se torna difícil cuando no tenemos la tecnología adecuada para hacerlo, pero de ahí en fuera el proceso es mucho muy fácil como lo anteriormente detallado, no es tan costoso y reduce una gran cantidad

de circuitería cuando hacemos lógica grande, la desventaja de este tipo de implementación es que debemos obtener nosotros todos los minitérminos que se puedan generar y después ingresarlos a mano en el programador, pero al final nos percatamos de que en realidad no es tanto trabajo, a diferencia de hacerlo con compuertas.

El dispositivo funcionó correctamente al probarlo en todas sus decodificaciones, por tanto se aprendió cómo programar el dispositivo EEPROM AT28C64.

::

::

Referencias:

Hoja de especificaciones de la AT28C64

Manual NTE (para el display de 7 segmentos)

Robot Seguidor de Líneas.

Usemos una metodología simple y ordenada, primero debemos definir las funciones que realizará nuestro móvil para establecer las condiciones del funcionamiento, así será mucho más sencillo llegar al circuito idóneo. Establezcamos esta condición como parte del comportamiento del móvil, cuando salgan de la línea **totalmente**, los sensores que viren hasta que la encuentre de nuevo. Es decir, si los sensores salen hacia la derecha o izquierda, que tan sólo detenga el movimiento de la rueda respectiva y así retornará al camino a seguir de manera normal, esta función es muy útil cuando las curvas son muy cerradas o que alguno de los motores sea un poco rápido y se salga del camino, como comúnmente pasa en un móvil de este tipo por su sencillez.

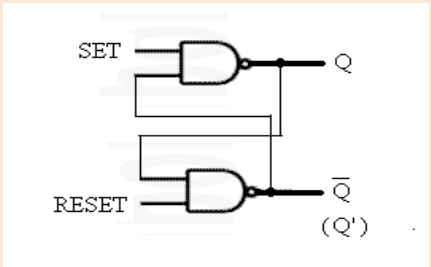
Lo anterior nos indica que si salen hacia la derecha o hacia la izquierda, el móvil intentará buscar la línea hacia el lado contrario de donde se salió, es decir que si sale a la derecha totalmente que vire hacia la izquierda, si lo hace a la izquierda que vire a la derecha, esto dependerá del orden en los cuales los sensores, en este caso dos, salgan de la línea. Como condición inicial es que inicie el recorrido con los sensores sobre la línea, de esa manera nuestro simple circuito estará en condiciones de operar, de lo anterior deducimos que debe tener "memoria" para recordar cuál de los dos sensores salió de la línea primero, de esa manera si el móvil sale de la línea y los dos sensores no lo detectan, "recuerde" hacia dónde debe ir. Como es simple y con dos entradas, podemos implementar nuestro circuito con un Flip-Flop R-S, y para que nuestro circuito sea aún más sencillo, lo armaremos con dos compuertas NAND, de acuerdo a la tabla de verdad que implementamos. El estado de memoria se dará sólo cuando las entradas sean un "1" lógico, si ambas entradas son "0" lógico las dos salidas del Flip-Flop serán uno, (a este estado se le puede llamar estado prohibido) si SET lo llevamos primero a "1" Q será "0" y Q' será "1", si a continuación RESET se lleva a "1"la salida se conservará y por lo tanto, el estado de memoria se dará.

Si realizamos lo anterior pero empezando por RESET, y luego pasamos por SET, los estados de salida serán invertidos, por lo tanto el estado de memoria dependerá de cual sensor salga primero de la línea. Un Flip-Flop implementado con dos NAND tiene la peculiaridad de que la activación se da por los niveles bajos, razón por la cual el estado inicial permitido a las entradas deba ser "1"lógico, esto para no causar confusión en el análisis de la tabla de la figura 1.

Figura 1:

Set	Reset	Q	Q'
0	0	1	1
1	0	0	1
1	1	0	1
0	0	1	1
0	1	1	0
1	1	1	0

Tabla de verdad para el Flip-Flop Implementado con NAND



Flip-Flop

Por lo tanto, para que se dé la condición para que avance hacia delante, es que las entradas R y S deben ser “0” que es cuando los sensores están detectando la línea, que en este caso será blanca sobre fondo negro, y usando el circuito de la figura 2 tendremos que en el resistor de 10k Ω (R2 y R5) que está como salida del fotodiodo infrarrojo una caída de voltaje que interpretaremos como “1” lógico, y tenemos que invertir el estado colocando un simple inversor.

Ahora las salidas del Flip-Flop serán un “1” lógico con lo cual podemos llevar a saturación a un transistor y así poner en marcha cada motor.

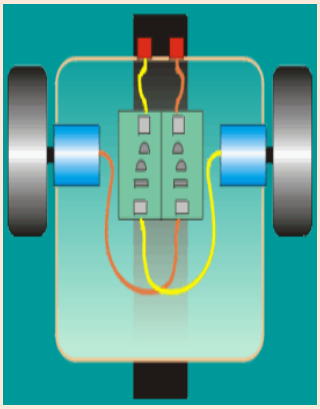


Figura 3. Colocación de sensores respecto a los motores.

Si alguno de los sensores sale de la línea, el resistor de 10k Ω entregará un “0” lógico, que pasando por el inversor será un “1” lógico y el funcionamiento será como el descrito para el Flip-Flop.

De esta manera se detendrá el motor respectivo para que nuestro móvil pueda regresar a la línea, “recordando” hacia dónde podrá virar.

La colocación de los sensores con respecto a los motores en nuestro pequeño móvil, debe ser que el sensor 1 esté del lado opuesto del motor 1 y el sensor 2 del lado contrario al motor 2.

Los emisores deben colocarse al centro y los receptores a los lados, los sensores deben ser colocados de tal manera que no sobresalgan del ancho de una cinta de aislar plástica.

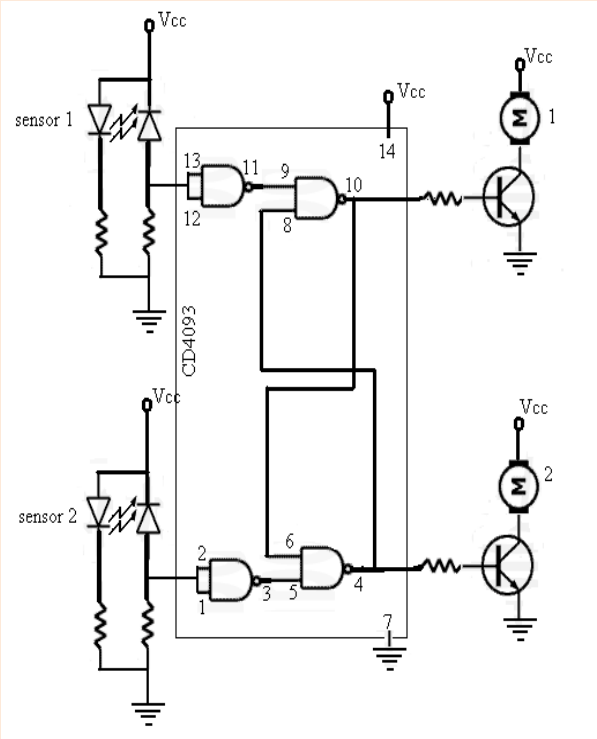
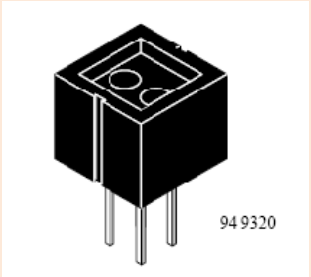


Diagrama de conexiones.

Sensores.

Para este diseño utilizamos sensores de tipo CNY70 que son dos sensores infrarrojos ópticos reflectivos con salida a transistor de la marca Vishay Telefunken, cuyo empaquetado es cuadrado con la superficie empotrada y es mostrado a continuación.



Forma del CNY70.

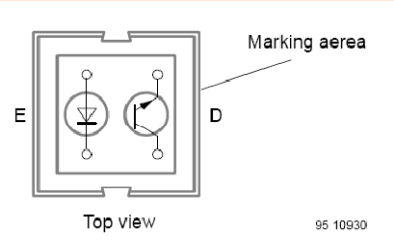


Diagrama de pines.

Estos sensores constan de un emisor infrarrojo y un foto receptor, que son sólo un fotodiodo y un fototransistor separados por una pared, esta pared impide que haya censados no deseados; cuando el emisor esté funcionando, el rayo infrarrojo puede rebotar en la superficie a censar y luego el fototransistor se polarizará en su región de corte o de saturación dependiendo del estado lógico del rebote.

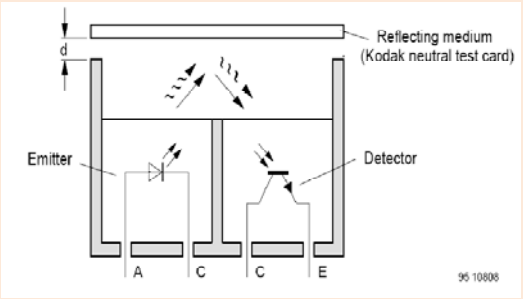
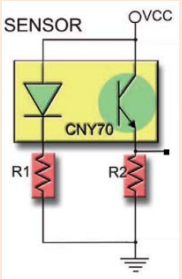


Ilustración del procedimiento de emisión y recepción para el sensor.

Para este diseño se utiliza una polarización clásica de este tipo de sensores que es llevar al fotodiodo a emitir conectándolo a una resistencia de 1KΩ ya sea en el ánodo o el cátodo y el fototransistor se polariza en colector común con una resistencia de 100kΩ al emisor, tomando la salida en el emisor mismo.



Configuración básica del CNY70.

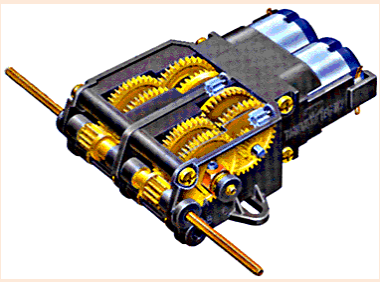
Así, cuando leamos una línea blanca o negra la salida nos entregará un “1” lógico con el transistor en saturación ó un “0” lógico con el transistor en corte.

Motores.

En cuanto a la salida de la compuerta, tenemos que es conectado a un transistor 2N2222 ó uno de mejor calidad para conectar ahora los motores en el colector y VCC, esto es suficiente si utilizamos motores pequeños de 3 volts, pero si usamos otros más potentes como los de 5 volts o un Twin como en este caso, debemos agregar una resistencia de colector para atenuar la corriente ya que este tipo de motores demandan un poco más de corriente, usaremos una resistencia de 10Ω a 2W ó en su caso un arreglo en serie de dos pares de configuraciones de resistencias en paralelo todas de 10Ω a 1/4W, ya que funcionará igualmente y será más económico el precio.

Con esto el robot estará listo para probar y funcionar, el armado final depende de los materiales que usemos.

Para este diseño se usan motores Twin con su adicional arreglo de resistencias en el colector.


Motores Twin Gearbox.

El diagrama final armado y probado es el siguiente:

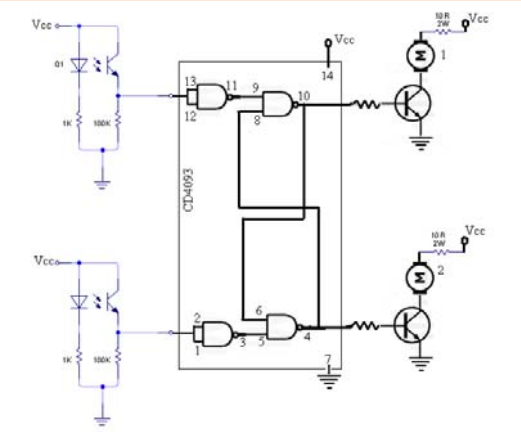
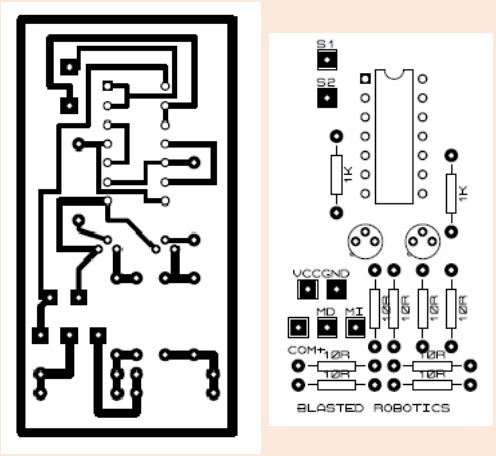
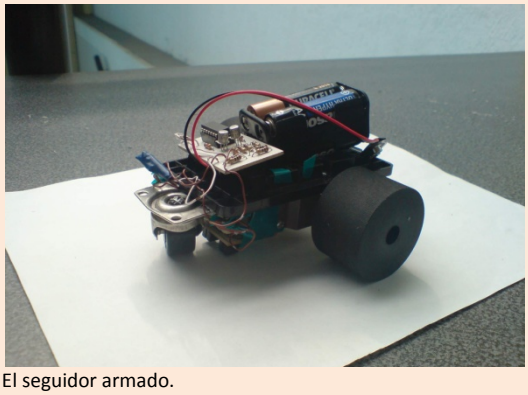


Diagrama final.

El diseño puede hacerse simplemente en una base DIP o un PCB como el que diseñamos:



PCB de el bloque de lógica.


El seguidor armado.

¿Por qué en vez de un Flip Flop integrado se utilizan dos compuertas NAND?

Primero por que algunos Flip Flop establecen algunos estados prohibidos al aplicar el mismo estado lógico a las entradas, y el implementarlo con dos NAND nos facilita el establecer la lógica de funcionamiento de acuerdo a nuestras necesidades, y porque son necesarias también dos inversores y pueden implementarse fácilmente al cortocircuitar las dos entradas de una NAND.

Además, de esa manera **uso un solo integrado**.

¿Puedo usar lógica TTL para este circuito?

No, el diseño es para implementarlo con CMOS ya que así tendremos la libertad de usar voltajes de alimentación de 3 a 6 Volts con el mismo circuito, así no tendremos sorpresas al disminuir un poco el nivel de baterías y podemos utilizar cualquiera entre este rango, además que los niveles lógicos en esta tecnología se adecuan a los niveles de voltaje de alimentación, y podemos activar con relativa facilidad la etapa de transistores.

¿Por qué aquí no se utilizan amplificadores operacionales para el ajuste de umbral?

Para acoplar la salida de los sensores con el Flip Flop se usó un inversor implementado con NAND. Al usar el C.I. 4093 tiene la característica de ser4 con entrada Schmitt Trigger, lo cual internamente establece umbrales de cambio , y a su vez minimiza errores de funcionamiento por ruido además de que al tener a la entrada un resistor de 10k Ω aterrizado lo interpreta como un “0”lógico al no haber prácticamente corriente circulante a través de él.

¿Es posible cambiar la etapa de potencia del motor por algo de mayor capacidad?

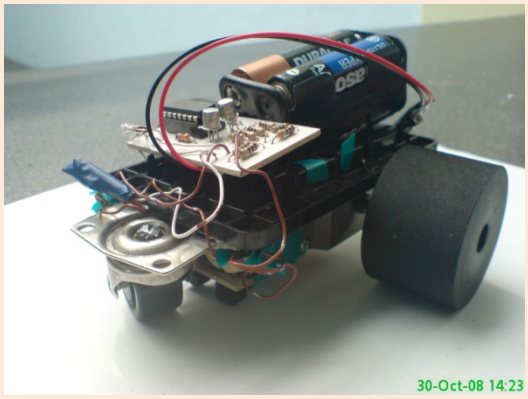
Si usamos transistores 2N2222, es por que los mecanismos que se utilizaron tienen motores muy eficientes que no consumen gran corriente y son pequeñitos incluso el diseño soporta motores Twin. Si se desean utilizar motores de mayor capacidad tendrán que utilizar una etapa Darlington.

¿Los sensores deben ser CNY70?

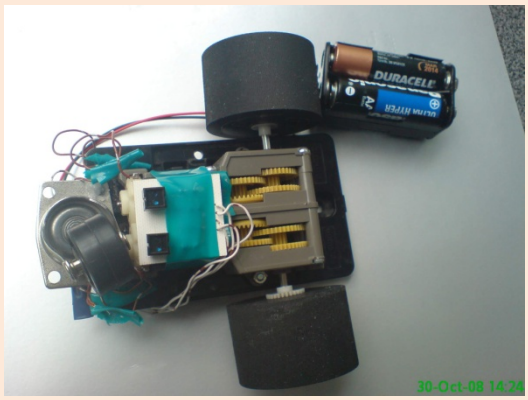
Para este diseño se proponen, pero como el diseño se realiza a bloques podemos usar tan solo dos sensores infrarrojos interruptivos que se pueden conseguir con mucha facilidad, el emisor y el receptor están pareados y se colocan uno a lado del otro, emulando el CNY70, claro que deben respetar la distancia con la superficie para que no sea mayor a 3mm. En caso de conseguir el CNY70, familias QRD o equivalentes pueden usarlos y el resultado será el mismo.

¿Por qué ahora se usa línea blanca en fondo negro?

Mas que nada es por cuestión técnica, ya que algunos eventos establecen como pista para la prueba esas características, si desean que funcione con una línea negra en fondo blanco, es sencillo, sólo deben adicionar otro inversor antes de cada entrada del Flip-Flop. También se podría eliminar simplemente el inversor y usar sólo el Flip-Flop, lo cual no se aconseja por la posibilidad de errores ya que el primer inversor es para acoplar el sensor de la etapa lógica.



El Robot Seguidor de Líneas.



Vista Inferior.

Referencias:

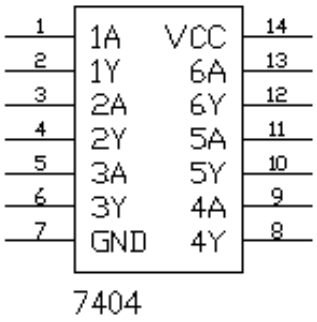
Revista Saber Electrónica.
www.jameco.com
Datasheet del integrado CNY70.

IDENTIFICADOR DE NUMEROS PRIMOS CON COMPUERTAS LÓGICAS

El objetivo de esta práctica es crear un circuito que identifique los números primos, de un rango comprendido de entre el número 1 hasta el 15, utilizando las compuertas lógicas, los mapas de Karnaugh y la reducción de funciones, mediante álgebra Booleana.

A continuación haremos una breve explicación de las configuraciones de las compuertas que vamos a utilizar en nuestro circuito.

SN74LS04 – INVERSOR



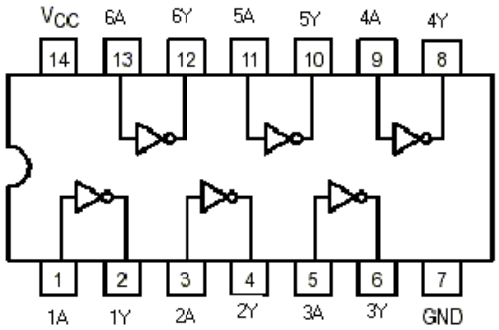
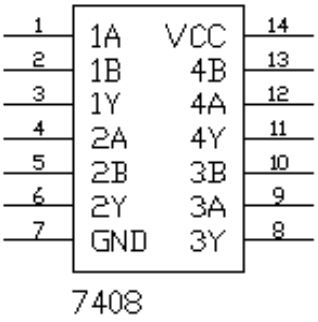


Tabla de verdad	
X	Y
0	1
1	0

Diagrama lógico

SN74LS08 – AND



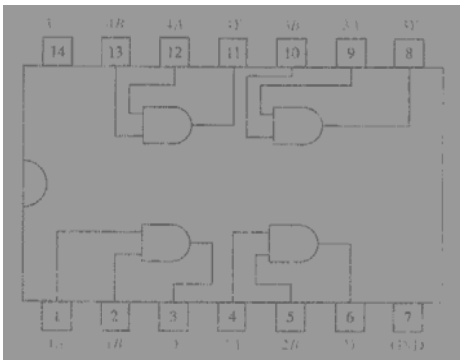
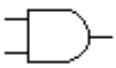
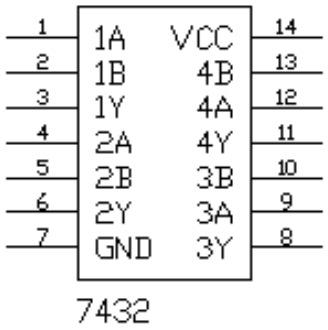


Tabla de verdad		
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Diagrama lógico



SN74LS32 – OR



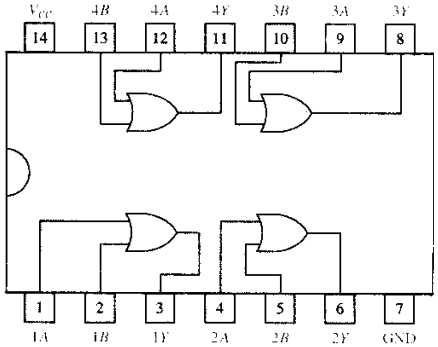


Tabla de verdad		
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Diagrama lógico

Después de planteado el problema y explicado las configuraciones básicas de las compuertas lógicas a vamos a utilizar, procedemos a resolver nuestro problema.

Lo primero será hacer nuestra tabla de verdad, de la cual extraeremos todos los datos tanto de las entradas (las variables que vamos a controlar), como el de la salida que tendremos.

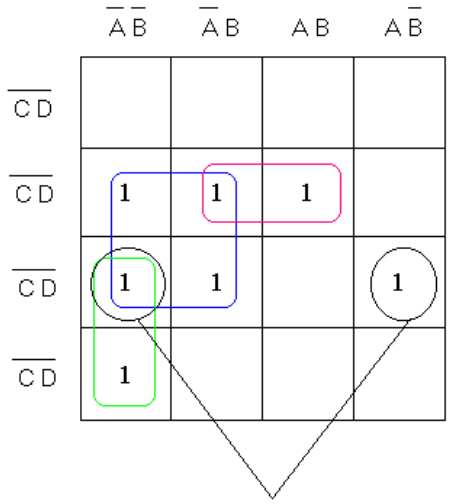
Sabemos que tendremos 16 distintas entradas, equivalente a 2.^4, las cuales serán los números en los que haremos la comparación y tendremos dos únicas salidas, "cierto" o "falso", y éstas son las que determinarán cuando el número introducido es primo o no.

Decimal	A	B	C	D	f
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

En nuestra tabla tomaremos los números de f en 1 cuando el número sea primo, y será cero si el número de la combinación introducida no es primo (lógica positiva); así, los que tengan el número 1, serán los mintérminos en nuestro mapa de Karnaugh.

A continuación, haremos nuestro mapa de Karnaugh de cuatro variables, tomando los mintérmimos como los números que son primos de la tabla de verdad.

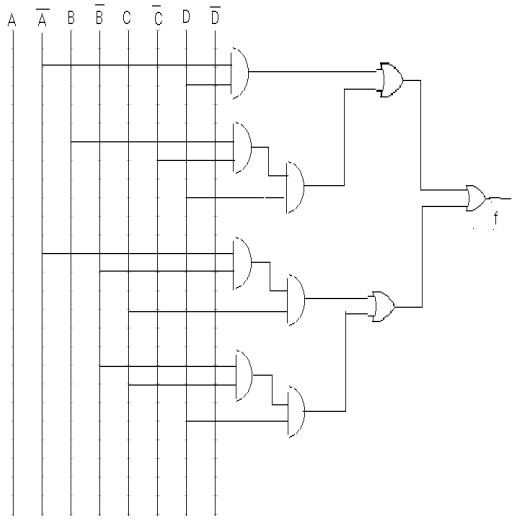
El mapa queda como a continuación:



Y la reducción de mintérminos, queda:

$$\Sigma m\ (1,\ 2,\ 3,\ 5,\ 7,\ 11,\ 13) = \bar{A}\,D + \bar{A}\,\bar{B}\,C + B\,\bar{C}\,D + \bar{B}\,C\,D$$

Y nuestro diagrama lógico:



El material requerido para el armado de nuestra práctica fue el siguiente:

- 2 compuertas AND (74LS08)
- 1 compuerta INV (74LS04)
- 1 compuerta OR (74LS32)
- 1 protoboard
- 1 interruptor de cuatro entradas
- 1 resistencia de 100 Ω
- 1 led
- cable para protoboard
- 1 fuente de voltaje

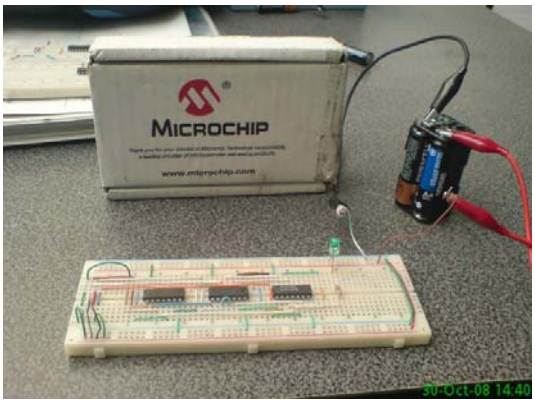
Después de diseñar nuestro circuito lógico, procedemos a armarlo. Una vez armado, lo alimentamos con una fuente de dc y con el dip switch haremos las combinaciones resultantes, tomando en cuenta, que es de cuatro entradas, el número de bits que serán introducidos también. El circuito encenderá el led cuando la combinación introducida sea un número primo; y no tendrá reacción alguna (se mantendrá apagado), cuando el número de la combinación no sea un número primo.

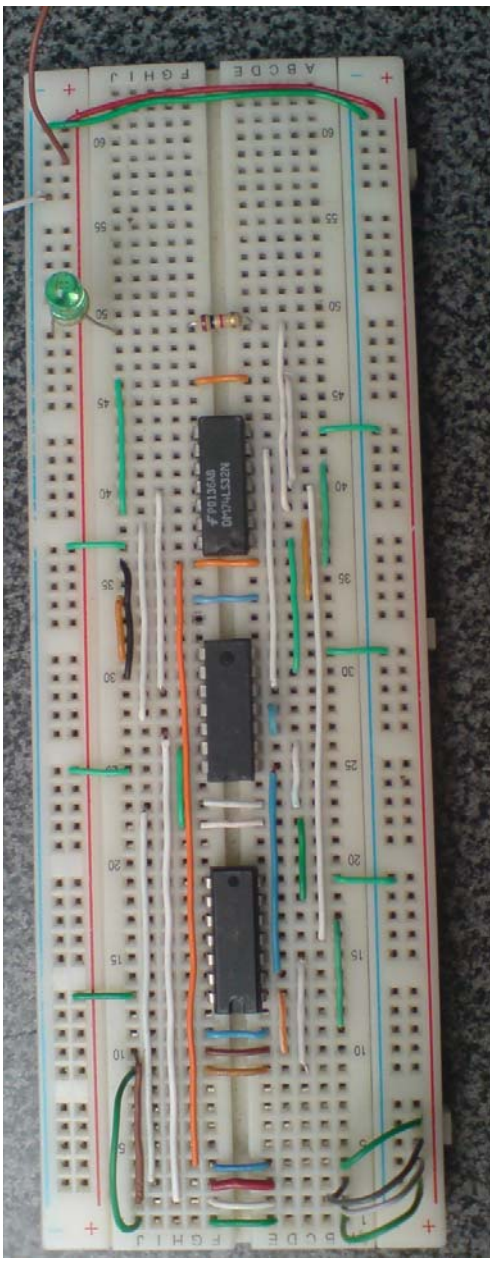
Decimal	Combinación del interruptor				LED
0	0	0	0	0	Apagado
1	0	0	0	1	**Encendido**
2	0	0	1	0	**Encendido**
3	0	0	1	1	**Encendido**
4	0	1	0	0	Apagado
5	0	1	0	1	**Encendido**
6	0	1	1	0	Apagado
7	0	1	1	1	**Encendido**
8	1	0	0	0	Apagado
9	1	0	0	1	Apagado
10	1	0	1	0	Apagado
11	1	0	1	1	**Encendido**
12	1	1	0	0	Apagado
13	1	1	0	1	**Encendido**
14	1	1	1	0	Apagado
15	1	1	1	1	Apagado

La tabla anterior muestra, de acuerdo a la combinación introducida, e identifica si se trata o no de un número primo.

Al haber realizado esta práctica, y después de haber probado todas las combinaciones de nuestro circuito, para averiguar cuáles de esos son números primos, el circuito ha tenido éxito, ya que en ninguna combinación hubo algún error, y en ninguno de los casos hubo alguna anomalía. Esto es demostrable también por que en caso de que hubiese existido algún error, tal vez hubiesen fallado alguna, o algunas de las combinaciones de identificación de los números introducidos.

Puedo agregar también, que los mapas de Karnaugh nos hicieron el trabajo de reducción de funciones bastante sencillo, mediante álgebra Booleana. El diagrama lógico fue sencillo de realizar una vez teniendo la reducción final de nuestro circuito.





Bibliografía.

- Análisis y diseño de circuitos lógicos digitales
  Nelson, Nagle, Carroll, Irwin
  Editorial Prentice Hall

- TTL Logic (Data book)
  Texas instruments

- SGS – Thomson
  (CD-Room Datasheets)

Autor:

Raúl García Chetla.

Introducción a PICC de CCS.

En el mundo de los microcontroladores, encontramos siempre opciones económicas, eficaces, rápidas, eficientes y útiles para diseñar, tal es el caso de los microntroladores PIC (Peripheral ***I****nterface* ***C****ontroller* /Controlador de Interfaz Periférico) de la marca Microchip Technology Inc., los cuales hoy en día son encontrados en un sin número de versiones y características especiales.

Estos microcontroladores usan un juego de instrucciones tipo RISC, cuyo número puede variar desde 35 para PICs de gama baja a 70 para los de gama alta. Las instrucciones se clasifican entre las que realizan operaciones entre el acumulador y una constante, entre el acumulador y una posición de memoria, instrucciones de condicionamiento y de salto/retorno, implementación de interrupciones y una para pasar a modo de bajo consumo llamada *sleep*.

Microchip proporciona un entorno de desarrollo freeware llamado MPLAB que incluye un simulador software y un ensamblador. Otras empresas desarrollan compiladores C y BASIC. Microchip también vende compiladores para los PICs de gama alta ("C18" para la serie F18 y "C30" para los dsPICs) y se puede descargar una edición para estudiantes del C18 que inhabilita algunas opciones después de un tiempo de evaluación.

En la actualidad lo que los programadores de estos dispositivos buscamos es un entorno más amigable de programación, mas rápido y más fácil de usar, sin tener que estar horas creando rutinas en lenguaje ensamblador, es por ello que actualmente estas empresas que desarrollaron compiladores en lenguajes como C y BASIC se han vuelto famosas. Por tal motivo en este artículo retomamos la programación de los microcontroladores PIC con el lenguaje de alto nivel C, con el compilador PICC de la empresa CCS (Custom Computer Services Inc.) en su versión 4.038, que tratándolo es sumamente amigable y fácil de usar.

El compilador PICC tiene tres compiladores integrados PCB, PCM y PCH, cada uno para las versiones de código de PICs de 12, 14 y 16 bits respectivamente, siendo editadas las instrucciones en la ventana del command window llamada PCW, donde la apariencia de este compilador tiene un formato estándar ANSI. Este compilador es capaz de crear el archivo .HEX, .COF, y otros tantos archivos que son generados al crear nuestro proyecto, tales archivos con su respectiva función se pueden ver en el cuadro. La instalación del programa es sumamente sencilla y sólo hay que seguir los pasos del fabricante para instalarlo. Existe una versión trial o demo en la página del fabricante.

.C	This is the source file containing user C source code.
.H	These are standard or custom header files used to define pins, register, register bits, functions and preprocessor directives.
.PJT	This is the project file which contains information related to the project.
.LST	This is the listing file which shows each C source line and the associated assembly code generated for that line.
.SYM	This is the symbol map which shows each register location and what program variables are stored in each location.
.STA	The statistics file shows the RAM, ROM, and STACK usage. It provides information on the source codes structural and textual complexities using Halstead and McCabe metrics.
.TRE	The tree file shows the call tree. it details each function and what functions it calls along with the ROM and RAM usage for each function.
.HEX	The compiler generates standard HEX files that are compatible with all programmers.
.COF	This is a binary containing machine code and debugging information.
.COD	This is a binary file containing debug information.
.RTF	The output of the Documentation Generator is exported in a Rich Text File format which can be viewed using the RTF editor or wordpad.
.RVF	The Rich View Format is used by the RTF Editor within the IDE to view the Rich Text File.
.DGR	The .DGR file is the output of the flowchart maker.
.ESYM	This file is generated for the IDE users. The file contains Identifiers and Comment information. This data can be used for automatic documentation generation and for the IDE helpers.
.OSYM	This file is generated when the compiler is set to export a relocatable object file. This file contains a list of symbols for that object.

Cuadro con los archives generados para un proyecto en PICC.

Para comenzar a utilizar este compilador qué mejor que practicando, y para ello haremos un programa para una aplicación típica, el encendido y apagado de LEDs, y para más familiaridad en el diseño utilizaremos un viejo amigo, el PIC16F877A, que por su sencillez nos será apto para este sencillo diseño, más adelante veremos que podemos programar microcontroladores de este tipo y más avanzados de la misma forma.

El objetivo será encender y apagar 8 LEDs conectados a el puerto B del Pic16F877A en forma de corrimiento (Tipo auto increíble); cuando conectemos el dispositivo los 8 LEDs estarán encendidos y cuando oprimamos un pushbutton conectado al pin A0 entonces comenzará el corrimiento de un bit en un sentido y regresará en sentido contrario en forma de circuito de corrimiento, esto le dará mas elegancia a nuestro diseño que podemos implementar en juguetes o iluminación, sólo cuando oprimamos el pushbutton conectado al pin A0 el corrimiento se efectuará.

El diagrama de flujo de nuestro programa se muestra en la fugura.

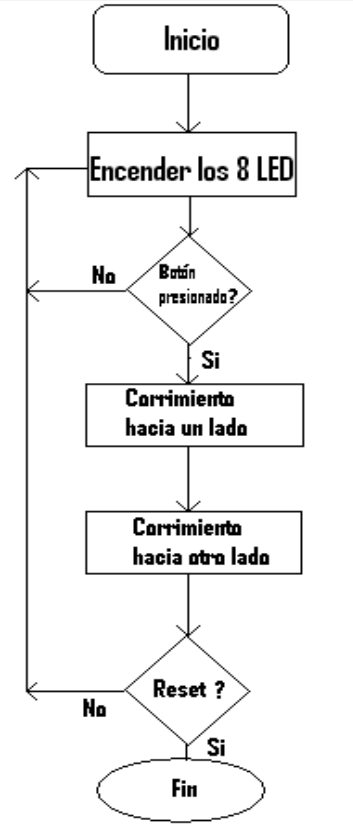
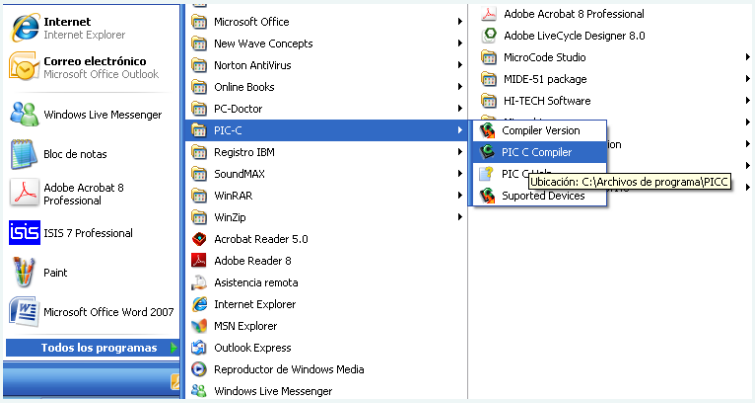


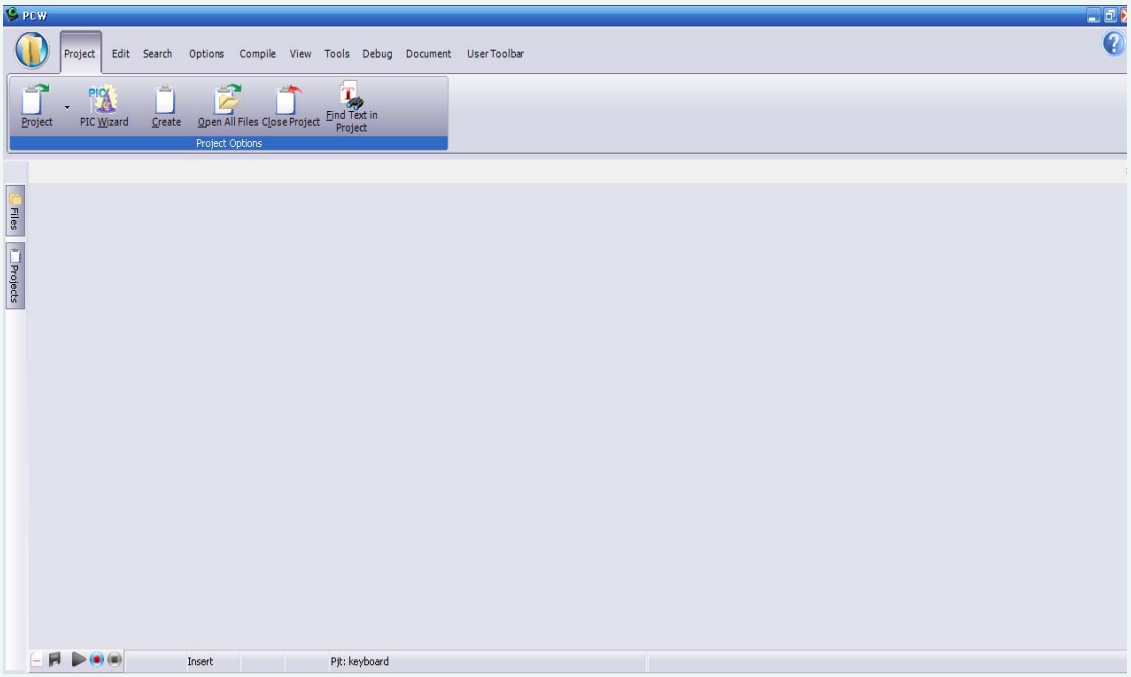
Diagrama de flujo para nuestro programa.

Como paso inicial para crear el código fuente .C donde podemos editar nuestro programa, procedemos a indicar la secuencia clásica de apertura del programa, en Windows, esto es, hacemos clic en Inicio > Todos los programas >PIC-C > PIC C Compiler.



Secuencia para abrir el PICC.

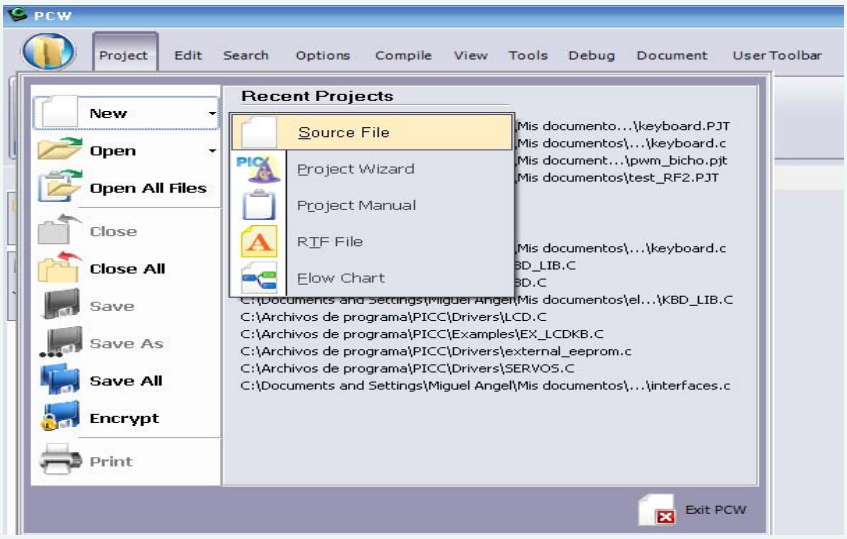
Se abrirá la siguiente ventana con el compilador PCW.



Ambiente del compilador.

En él encontramos la clásica barra de menús Archivo, Proyecto, Edición, Buscar, Opciones, Ver, Herramientas, etc.

Para iniciar y escribir nuestro primer programa hacemos un clic en el menú Archivo, que tiene un ícono de carpeta amarilla. Después un clic en Nuevo y después en Archivo fuente o Source File como se ve en la figura.



Secuencia para crear un nuevo archivo de código fuente.

Inmediatamente aparece un cuadro de diálogo para indicar dónde y con qué nombre guardar nuestro código fuente, en este caso lo guardamos en la carpeta Blasted con el nombre de introducción.c

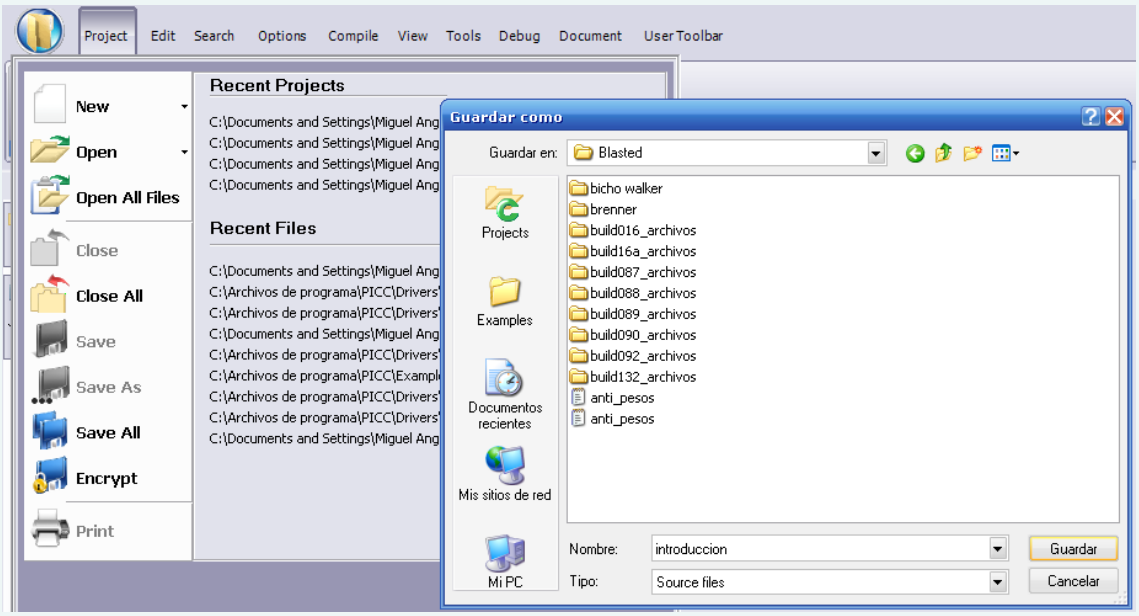


Figura que muestra la carpeta destino de nuestro archivo de código fuente.

Hecho esto tendremos una nueva pestaña en el command window llamada introducción, en la cual ya podemos comenzar a escribir nuestro código del programa.

Este compilador tiene las mismas reglas de sintaxis del lenguaje C, los comentarios estándar se escribirán iniciando en // y otros globales como //* ó como /*.

Podemos incluir las típicas librerías de C, utilizar las clásicas estructuras if, ciclos while, for, etc., las definiciones básicas de números como int, int8, char, short, long, etc.

Entonces sabiendo todo esto podemos compilar fácilmente cualquier programa.

Como primer paso en C para la programación deberíamos incluir las librerías que nos son útiles, en C clásico incluimos por ejemplo #include<stdio.h> , aquí podemos incluir librerías de este tipo, una de mucha utilidad es la librería que nos define cada pin de cada puerto de nuestro PIC, y por lo general llevan el nombre del PIC. Para este ejemplo anotaremos #include<16F877A.h> que es por el mirocontrolador que vamos a utilizar. Así pudimos haber incluido la definición para el PIC de nuestro agrado como #include <18F452.h> u otra que contenga el compilador. Al hacer esto entonces ya tenemos definido el puerto A en los pines 2 al 7, B en los pines del 33 al 40, como lo debe estar indicado en el datasheet del PIC.

Algo nuevo en este compilador es que podemos ordenar desde código al el "quemador de PICs" las opciones con las que debe quemar nuestro dispositivo, por ejemplo el Watchdog, el famoso LVP, la protección a código, el MCLR, el oscilador, la protección de código y otros.

Para hacer esto lo indicamos como #FUSES y la palabra que indica qué settings necesitamos, aquí configuraremos los siguientes:

```
#FUSES NOWDT                    //Deshabilitar el watchdog
#FUSES XT                       //Oscilador de cuarzo <= 4mhz (HS para >4MHz)
#FUSES NOPROTECT                //Sin protección de código de lectura
#FUSES NOLVP                    //Sin low voltage prgming,
                                // B3(PIC16) o B5(PIC18) usado para I/O
#FUSES NOCPD                    //Sin protección EE
```

Después una parte importantísima, que es la frecuencia del oscilador y esto se hace con #use, de la siguiente manera:

```
#use delay(clock=4000000) //Usar un oscilador de cuarzo de 4 MHz
```

Note que esta es una de las partes que nos necesitamos aprender ya que en cualquier programa debemos incluir esta parte, donde el número indica el número de Hertz del oscilador, entonces,

podemos cambiarlo a nuestro criterio a 20000000 por ejemplo para un Finalmente llegamos a la parte familiar del C, donde escribimos las rutinas, los objetos, etc., aquí escribimos el famoso `void main(){}`, que es la rutina principal que hará que nuestro programa funcione.

Entre las llaves {} escribiremos el programa que tomaremos de nuestro diagrama de flujo.

Para ello necesitamos encender por default los 8 bits del puerto B, y si el oscilador de 20MHz.

pushbutton es presionado se debe hacer el corrimiento de tipo auto fantástico.

Dentro del programa principal necesitamos declarar una variable local, que será el valor leído del pin A0 para saber qué hacer, si seguir mostrando todo el puerto B en alto o hacer el corrimiento de un bit a la derecha y luego a la izquierda, todo esto dentro de una estructura do-while, para que el programa se ejecute infinitamente.

Escribimos entonces:

```
void main()                    //El clásico Void main de C.
{                              //Inicio del programa
do{                            //Ciclo Do/While
   int valor;                  //Variable local entera
   valor=input(PIN_A0);        //el valor de la variable se lee en el PIN_A0
   }
//Aquí va el resto del código que necesitamos.
while(TRUE);                   //Condición de bucle infinito
}                              //Fin del Programa. Gracias!!!
```

Donde la variable local la llamamos valor y es de tipo entero ó entero de 8 bits. Ahora bien para leer esta variable usaremos otra instrucción muy importante que podemos ver en el código escrito y es la instrucción input(valor_a_ser_leido); entonces la variable valor la igualamos a la entrada que hay en A0, es decir:

```
valor=input(PIN_A0);
```

Ahora el programa está leyendo continuamente la entrada del pushbutton, resta hacer que cuando lea un “1” lógico todos los LEDs estén encendidos y cuando lea un “0” se haga el corrimiento una vez.

Eso se hace tan fácil agregando un simple if.

```
if(valor==0)
{ }
else{ }
```

Entonces de acuerdo a nuestra condición agregaremos una salida de 11111111_b cuando leamos un 1, en la parte else con el comando output_puerto(valor_en_hexadecimal); en este caso será el puerto B y la salida será $0xFF_h$

```
if(valor==0)
{ }
else{
output_b(0xFF);
}
```

Y para la parte del otro valor escribiremos muchos output_b(); con las diferentes salidas del corrimiento, claramente se nota que para que lo podamos ver necesitamos colocar retardos entre cada salida, aquí los retardos son hechos con el comando delay_us(retardo); ,delay_ms (retardo); ó delay_cycles(retardo); , para retardos de microsegundos, milisegundos ó personalizados respectivamente. Como queremos que sean visibles, entonces colocaremos retardos en milisegundos, con 70 milisegundos son suficientes. Note que en lenguaje ensamblador esto nos llevaría a hacer rutinas de retardos anidados muy grandes y llamarlos a cada rato.

Escribimos entonces dentro del if

```
output_b(0x00);              //Salida del dato hexadecimal en el puerto B
delay_ms (70);               //Esperar retardo de 70 milisegundos
```

Y así sucesivamente para cada valor del corrimiento como si fuera una memoria con su respectivo retardo.

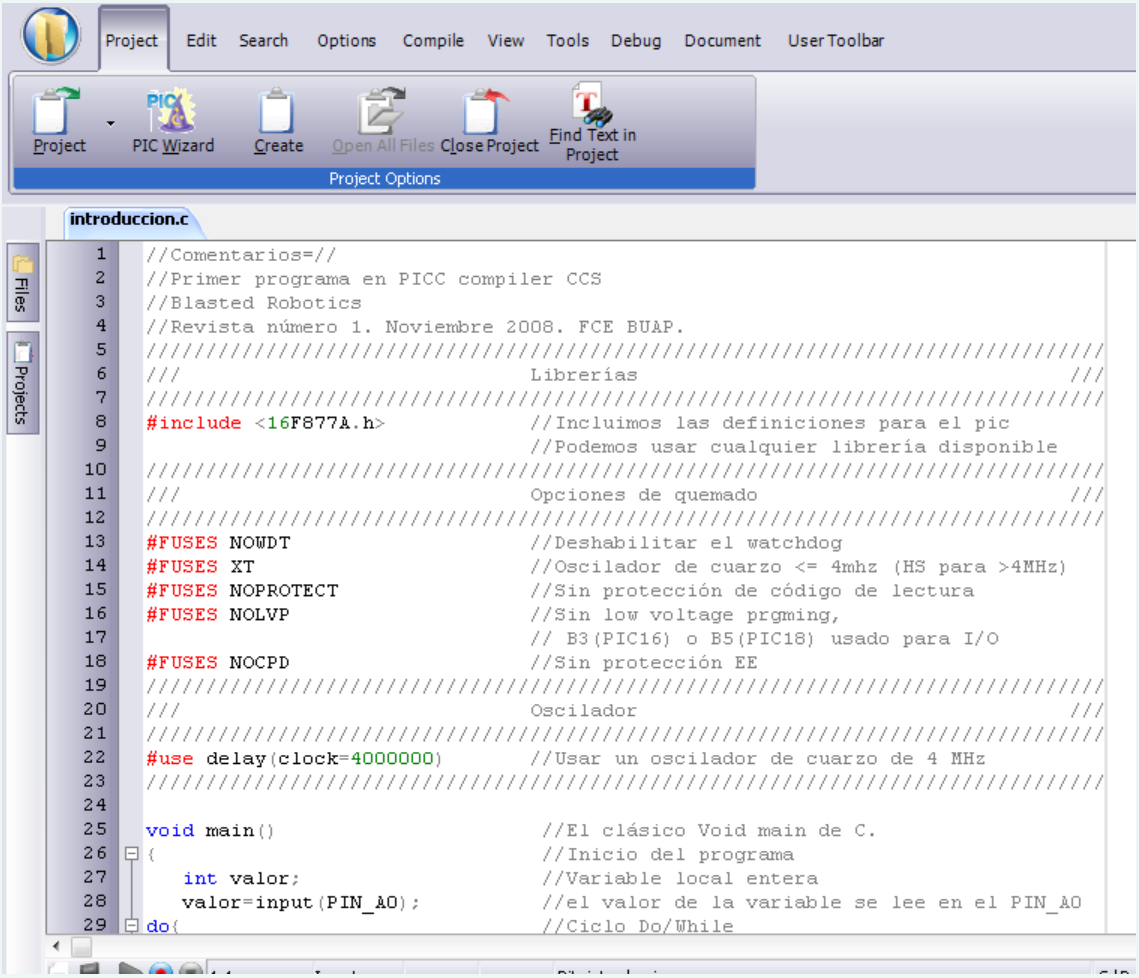
Como en la parte inicial de #FUSES no incluimos la de MCLR, esta opción funcionará y esta terminado nuestro programa que mostramos en el diagrama de flujo.

El programa entero se muestra a continuación.

```
//Comentarios=//
//Primer programa en PICC compiler CCS
//Blasted Robotics
//Revista número 1. Noviembre 2008. FCE BUAP.
////////////////////////////////////////////////////////////////////////
//
```

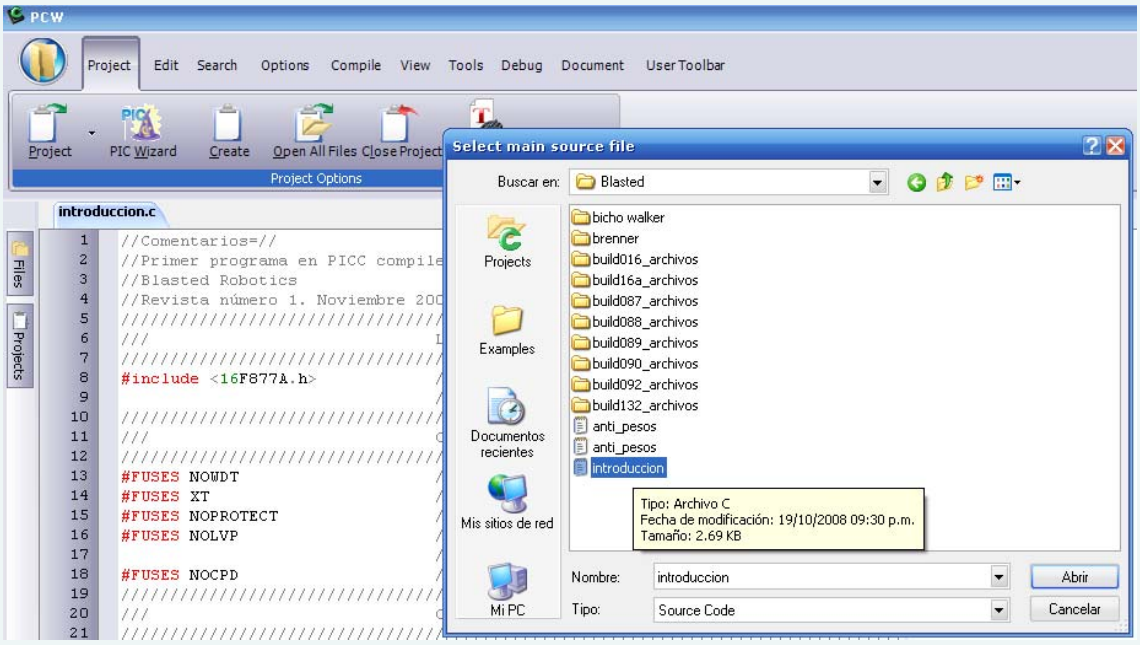
```
///                             Librerías
///
//////////////////////////////////////////////////////////////////////////
//
#include <16F877A.h>            //Incluimos las definiciones para el pic
                                //Podemos usar cualquier librería disponible
//////////////////////////////////////////////////////////////////////////
//
///                             Opciones de quemado
///
//////////////////////////////////////////////////////////////////////////
//
#FUSES NOWDT                    //Deshabilitar el watchdog
#FUSES XT                       //Oscilador de cuarzo <= 4mhz (HS para >4MHz)
#FUSES NOPROTECT                //Sin protección de código de lectura
#FUSES NOLVP                    //Sin low voltage prgming,
                                // B3(PIC16) o B5(PIC18) usado para I/O
#FUSES NOCPD                    //Sin protección EE
//////////////////////////////////////////////////////////////////////////
//
///                             Oscilador
///
//////////////////////////////////////////////////////////////////////////
//
#use delay(clock=4000000)       //Usar un oscilador de cuarzo de 4 MHz
//////////////////////////////////////////////////////////////////////////
//

void main()                      //El clásico Void main de C.
{                                //Inicio del programa

do{                              //Ciclo Do/While
   int valor;                    //Variable local entera
   valor=input(PIN_A0);          //el valor de la variable se lee en el PIN_A0

   if(valor==0){
   output_b(0x00);               //Salida del dato hexadecimal en el puerto B
   delay_ms (70);                //Esperar retardo de 70 milisegundos
   output_b(0x01);
   delay_ms (70);
   output_b(0x02);
   delay_ms (70);
   output_b(0x04);
   delay_ms (70);
   output_b(0x08);
   delay_ms (70);
   output_b(0x10);
   delay_ms (70);
   output_b(0x20);
   delay_ms (70);
   output_b(0x40);
   delay_ms (70);
   output_b(0x80);
   delay_ms (70);
   output_b(0x40);
   delay_ms (70);
   output_b(0x20);
   delay_ms (70);
   output_b(0x010);
   delay_ms (70);
   output_b(0x08);
   delay_ms (70);
   output_b(0x04);
   delay_ms (70);
   output_b(0x02);
   delay_ms (70);
   output_b(0x01);
   delay_ms (70);
```

```
   }
   else{
   output_b(0xFF);
   }
}
while(TRUE);                    //Condición de bucle infinito
}                               //Fin del Programa. Gracias!!!
```

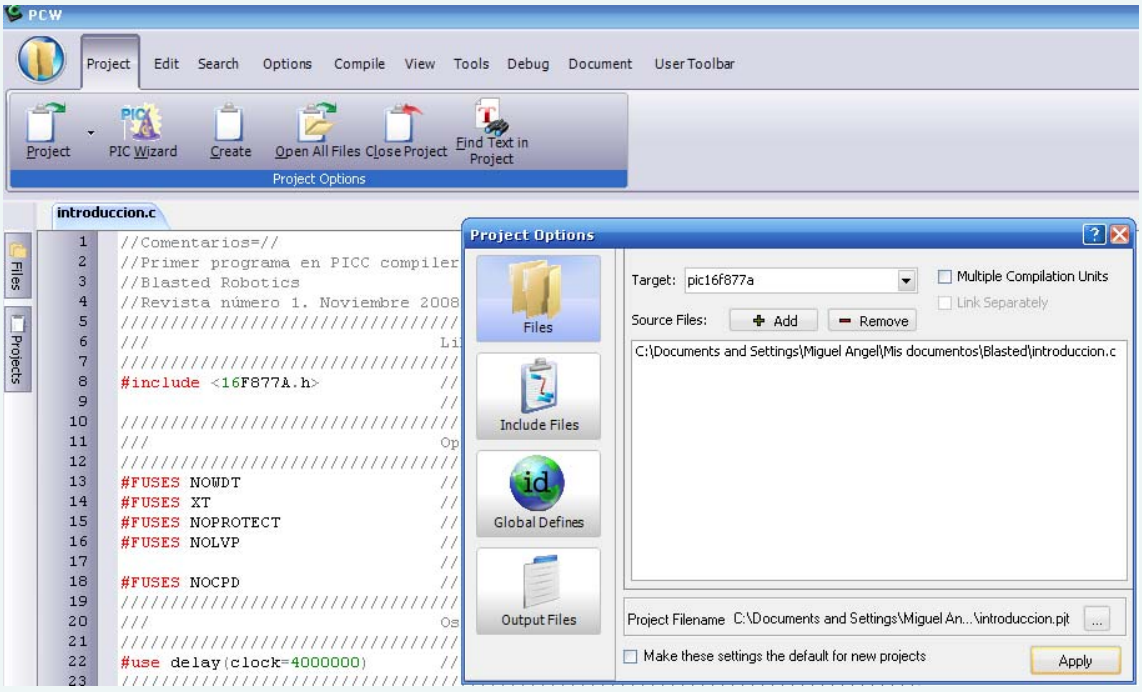


El programa escrito en C.

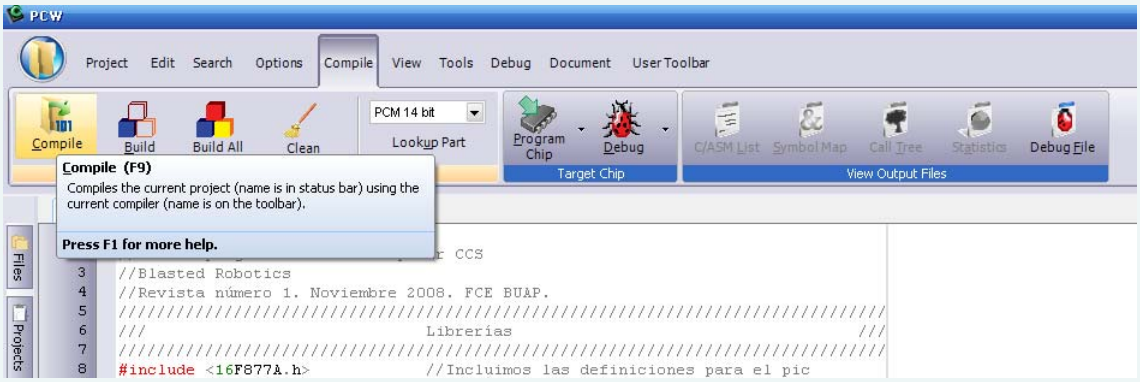
Tenemos ya el código fuente, entonces necesitamos compilarlo y generar por lo menos el código para programarlo al PIC, es decir, el .hex, pero para generarlo es necesario crear un nuevo proyecto, para hacerlo seleccionamos el menú Project >Create, y aparecerá el siguiente cuadro de diálogo que nos pedirá una ruta para abrir, tenemos que abrir el código que acabamos de generar, el introducción.c, lo buscamos donde lo guardamos, recordemos que lo guardamos en la carpeta Blasted, y hacemos un clic en abrir.



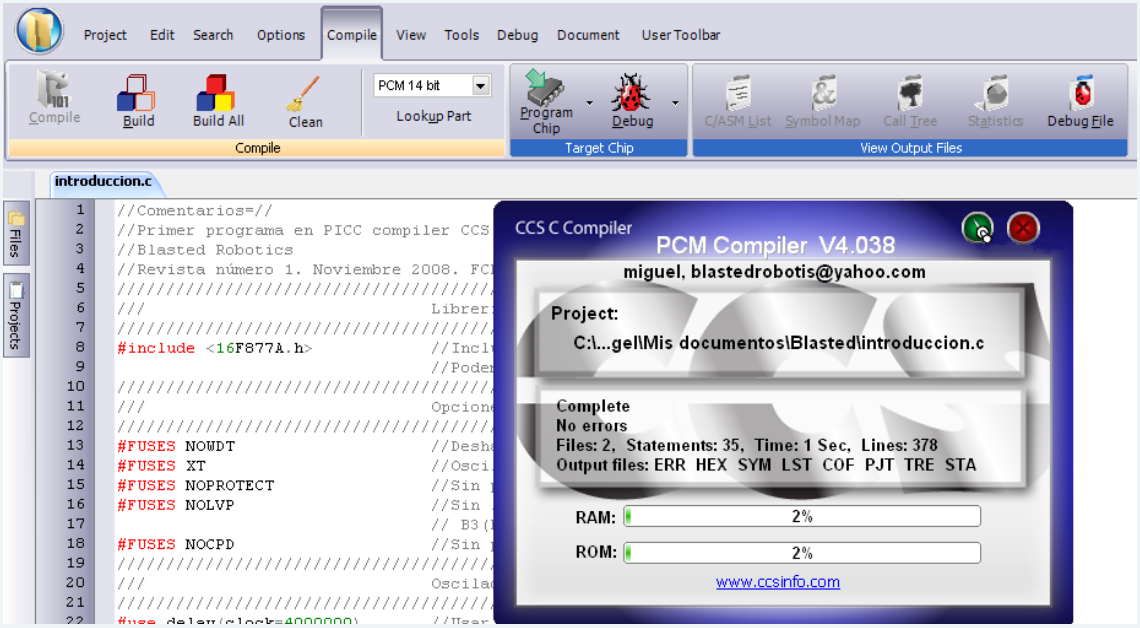
Inmediatamente otro cuadro de diálogo nos preguntará qué PIC utilizaremos, para este diseño hemos elegido el PIC16F877A, si hubiésemos elegido el pic18f452 igual tendríamos que anotar ese, lo anotamos y hacemos clic en Apply.



Una vez creado nuestro proyecto ya podemos compilar nuestro código hecho. Hacemos n clic en el menú Compile, y luego un clic en Compile o presionemos F9. Automáticamente el programa se compilará con su compilador respectivo de la pestaña Lookup Part y nos marcará si tenemos errores, nos generará traducciones al asm y otros varios archivos útiles.

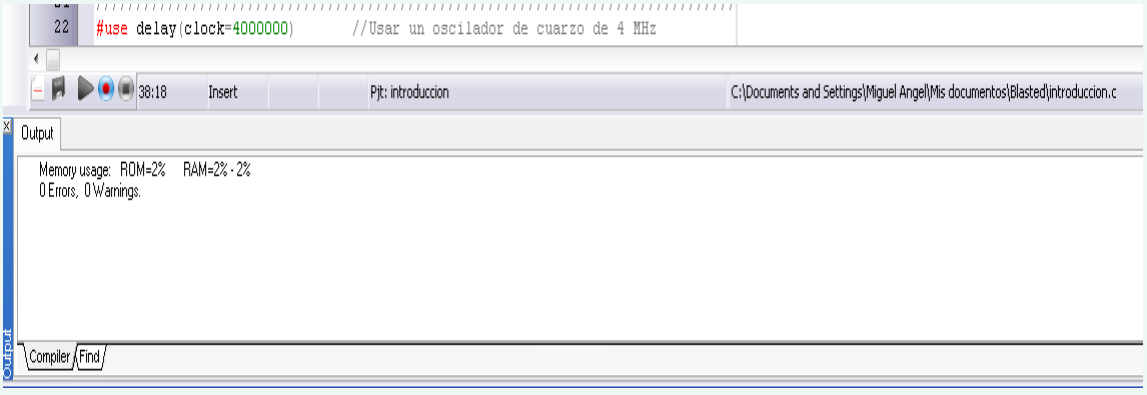


Botón Compilar.

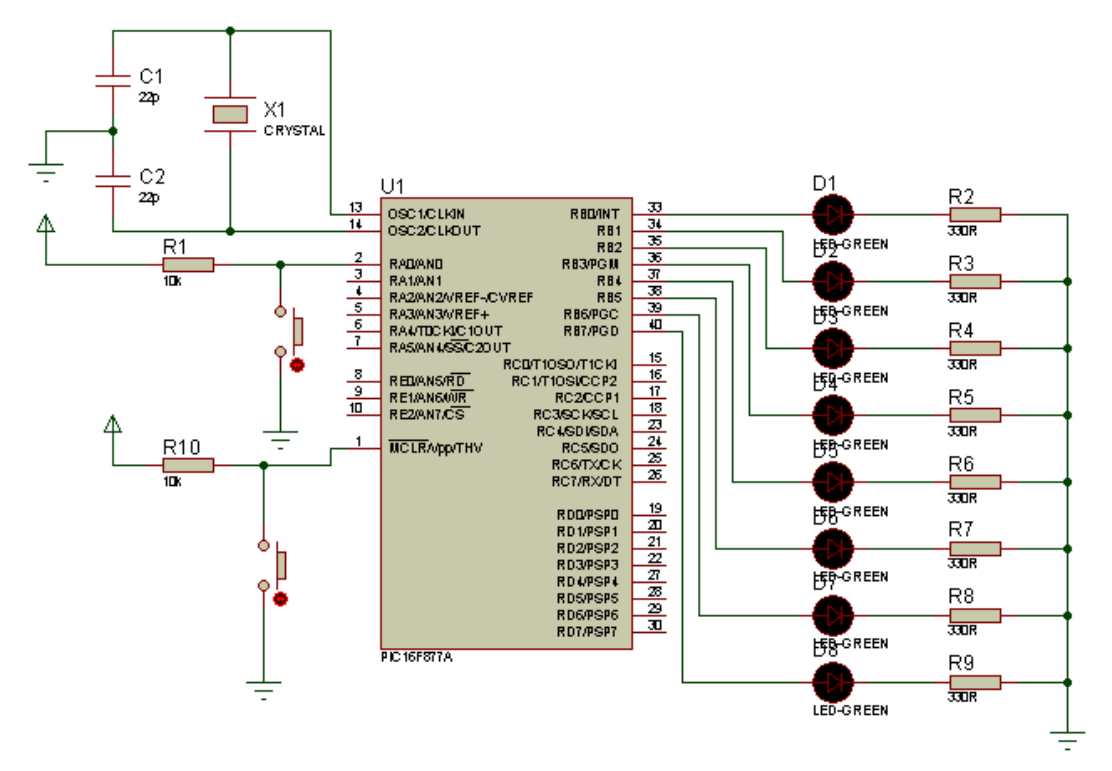


El programa compilándose, con su versión PCM para el 877ª, genera el .err, .hex, etc. Y nos muestra la cantidad de memoria que estamos utilizando en el dispositivo.

En este caso tenemos 0 errores y 0 advertencias, queriendo decir que el programa ya puede ser quemado en un PIC.



El esquema del circuito se muestra a continuación y puede ser simulado en Proteus 7.1 y funciona correctamente, una buena opción cuando no contamos con recursos para comprar un PIC.



Esquemático.

El PIC16f877A contiene el código introducción.hex programado, para programar el PIC utilizamos el quemador Winpic800 con el GTP_USB_Lite pero pueden usar cualquier otro programador como el incluido en PICC que es el ICD cuya información la encuentran fácilmente en la red. Y podemos usar cualquier otro con el .hex ó el .cof.

Finalmente comentamos que este compilador tiene opciones de Wizard para PICs más sofisticados y en general para cualquier PIC, tiene librerías HID, USB, tiene un monitor de puerto serial, convertidores de código a C, buscador de datasheets, convertidor de números, editor de macros y una muy buena ayuda que podemos leer y continuar aprendiendo a usar este programa y sus demás funciones que no son explicadas en este artículo. :-) R.D.

Autor: Miguel Ángel San Pablo J.
blastedrobotics@yahoo.com.mx

Referencias:

www.ccsinfo.com/ El demo del picc lo puedes descargar aquí.

CONVERTIDORES
ADC Y DAC

INTRODUCCION

En la actualidad existen diversos sistemas electrónicos que van desde la aplicación de la ley de ohm hasta sistemas desarrollados para las comunicaciones, aplicados en la electrónica de potencia, bioelectrónica, robótica en sistemas digitales como son las microcomputadoras y haciendo una interfaz con el mundo real pasándolo por un sistema analógico y llevándolo a los sistemas digitales. El espectacular desarrollo de la microelectrónica ha intensificado la tendencia generalizada, a aumentar la complejidad del sistema físico (hardware) para elevar la velocidad de los procesadores digitales y ampliar de esa forma su campo de aplicación.
Esto hace que como estudiantes nos veamos obligados a cambiar nuestros métodos de diseño y a elevar la capacidad de síntesis de estructuras digitales complejas.
En esta ocasión el desarrollo de esta práctica esta fundamentada en los convertidores analógico-digital y digital-analógico veremos su estructura, su funcionamiento y la aplicación que le damos.

CONVERTIDOR ANALÓGICO DIGITAL (ADC)

Un **conversor analógico-digital** es un dispositivo electrónico capaz de convertir un voltaje determinado en un valor binario, en otras palabras, este se encarga de transformar señales análogas a digitales (0's y 1's).

¿Cómo funciona?

Poseen dos señales de entrada llamadas Vref+ y Vref- y determinan el rango en el cual se convertirá una señal de entrada.

El dispositivo establece una relación entre su entrada (señal analógica) y su salida (Digital) dependiendo de su resolución. Esta resolución se pude saber, siempre y cuando conozcamos el valor máximo que la entrada de información utiliza y la cantidad máxima de la salida en dígitos binarios. A manera de ejemplo, el convertidor análogo digital ADC0804 tiene la capacidad de convertir una muestra analógica de entre 0 y 5 voltios y su resolución es:

Resolución = valor analógico / (2^8)-1

Resolución = 5 V / 255

Resolución = 0.0196v o 19.6mv.

Lo anterior quiere decir que por cada 19.6 milivoltios que aumente el nivel de tensión entre las entradas nomencladas como "Vref+" y "Vref-" que ofician de entrada al conversor, éste aumentará en una unidad su salida (siempre sumando en forma binaria bit a bit).

CONVERTIDOR DIGITAL ANALÓGICO (DAC)

Características

- Las señales son convertidas a formato discreto (digital) para facilitar su transmisión o almacenamiento.
- Es posible realizar mediante procesamiento digital acciones imposibles de obtener mediante el procesamiento analógico (por ejemplo, filtros con respuesta de frecuencia arbitraria).

La conversión se hace en forma digital porque ésta es usualmente más simple de realizar y más barato de implementar que en la conversión analógica. Además las señales digitales requieren usualmente menos ancho de banda y pueden ser comprimidas. Sin embargo, hay una pérdida inherente de información al convertir la información continua en discreta.

APLICACIONES

- Procesamiento Digital de Sonido
- Procesamiento Digital de Voz
- Procesamiento Digital de Imágenes
- Procesamiento Digital de Video

La conversión de digital analógico se utiliza en el procesamiento de música (por ejemplo MP3), de voz, teléfonos celulares, transmisión de imágenes y video.

Uno de los beneficios principales del DAC es que las transformaciones de señales son más sencillas de realizar. Una de las más importantes transformadas es la Transformada de Fourier discreta (TFD). Esta transformada convierte la señal del dominio del tiempo al dominio de la frecuencia. La TDF permite un análisis más sencillo y eficaz sobre la frecuencia, sobre todo en aplicaciones de eliminación de ruido y en otros tipos de filtrado.

CONVERTIDOR ANALÓGICO DIGITAL (ADC)

Una conversión analógica-digital consiste en la trascripción de señales analógicas en señales digitales, con el propósito de facilitar su procesamiento (encriptación, compresión, etc.) y hacer la señal resultante (la digital) más inmune al ruido y otras interferencias a las que son más sensibles las señales analógicas.

TIEMPO DE CONVERSIÓN:

Es el tiempo que tarda en realizar una medida el convertidor en concreto, y dependerá de la tecnología de medida empleada. Evidentemente nos da una cota máxima de la frecuencia de la señal a medir.

Si no respetamos el tiempo de conversión, en la salida tendremos un valor, que dependiendo de la constitución del convertidor será:

1. Un valor aleatorio, como consecuencia de la conversión en curso
2. El resultado de la última conversión

OBJETIVO

Crear primeramente un Convertidor Digital-Analógico (DAC) a base de transistores PNP, en este caso usaremos el transistor BC557, como nuestro DAC queremos que sea de 8 bits, entonces necesitamos 8 transistores BC557. Ya estando funcionando el DAC, procederemos a conectar estas salidas analógicas a un Convertidor Analógico-Digital (ADC), este ADC siendo creado también por componente a componente; al conectar el DAC con el ADC necesitaremos Buffers, contadores y una latch, obteniendo asi en la salida una señal de tipo RAMPA.

MATERIAL:

- Transistores BC557
- Contadores (74LS293)
- Amplificadores operacionales (OP_AMPs)
- Resistencias de 1 KOhm
- Resistencias variables
- leds
- capacitares ceramicos
- Protoboards
- cable para las conexiones
- Buffer(CD4049)
- Latch (74LS374)
- Comparador (LM339)
- CLK (LM555)

DESARROLLO

1. Como primera parte se implemento una etapa de oscilación con un CI 555 en configuración astable como se presenta a continuación:

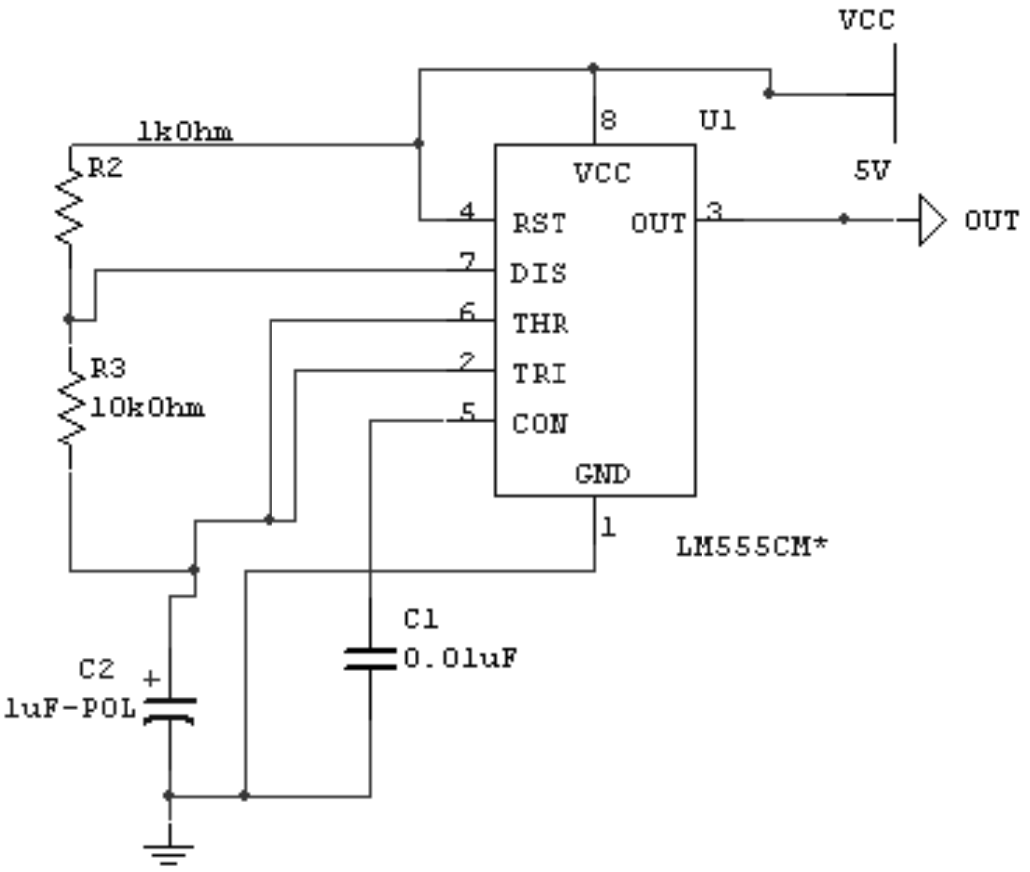


Diagrama de un oscilador con 555

2. Como segunda etapa se implementaron contadores pata mandar unos y ceros a nuestros transistores.

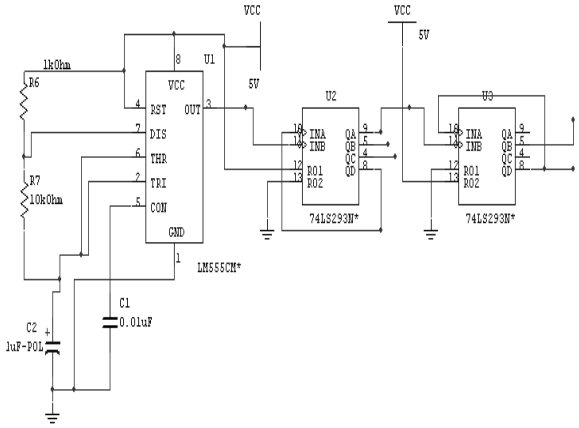


Diagrama de contador

3. En la tercera etapa conectamos la salida de los contadores a buffers estos para tener una mejor señal para los transistores, y así tenemos nuestro Convertidor Digital/Analógico (DAC) completo como lo mostramos en la figura.

4. En esta ultima etapa mostramos nuestro Convertidor Analógico/Digital (ADC), a la cual se le agrego amplificación en modo inversor para que la señal sea positiva y de allí a un comparador la cual nos compara nuestra señal de entrada del DAC y nuestro voltaje de entrada, la salida del comparador es la que nos da el pulso de activación a la LATCH y esta es la que nos da nuestra salida digital.

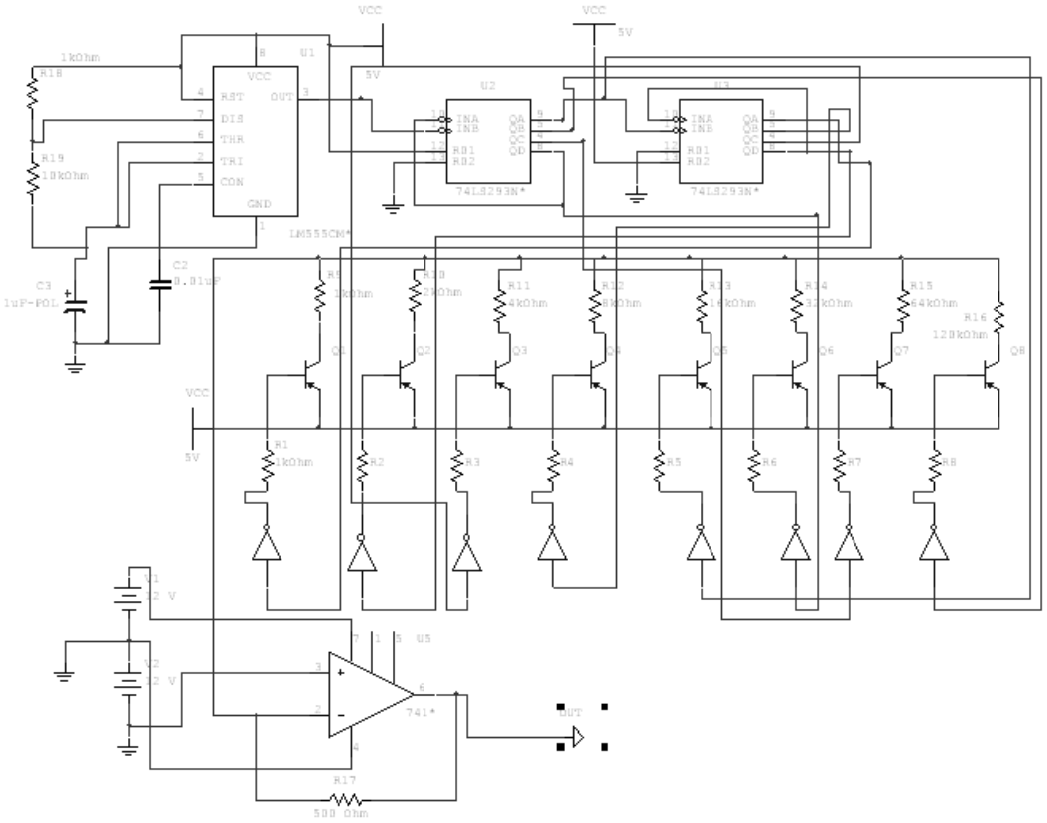


Diagrama para el DAC.

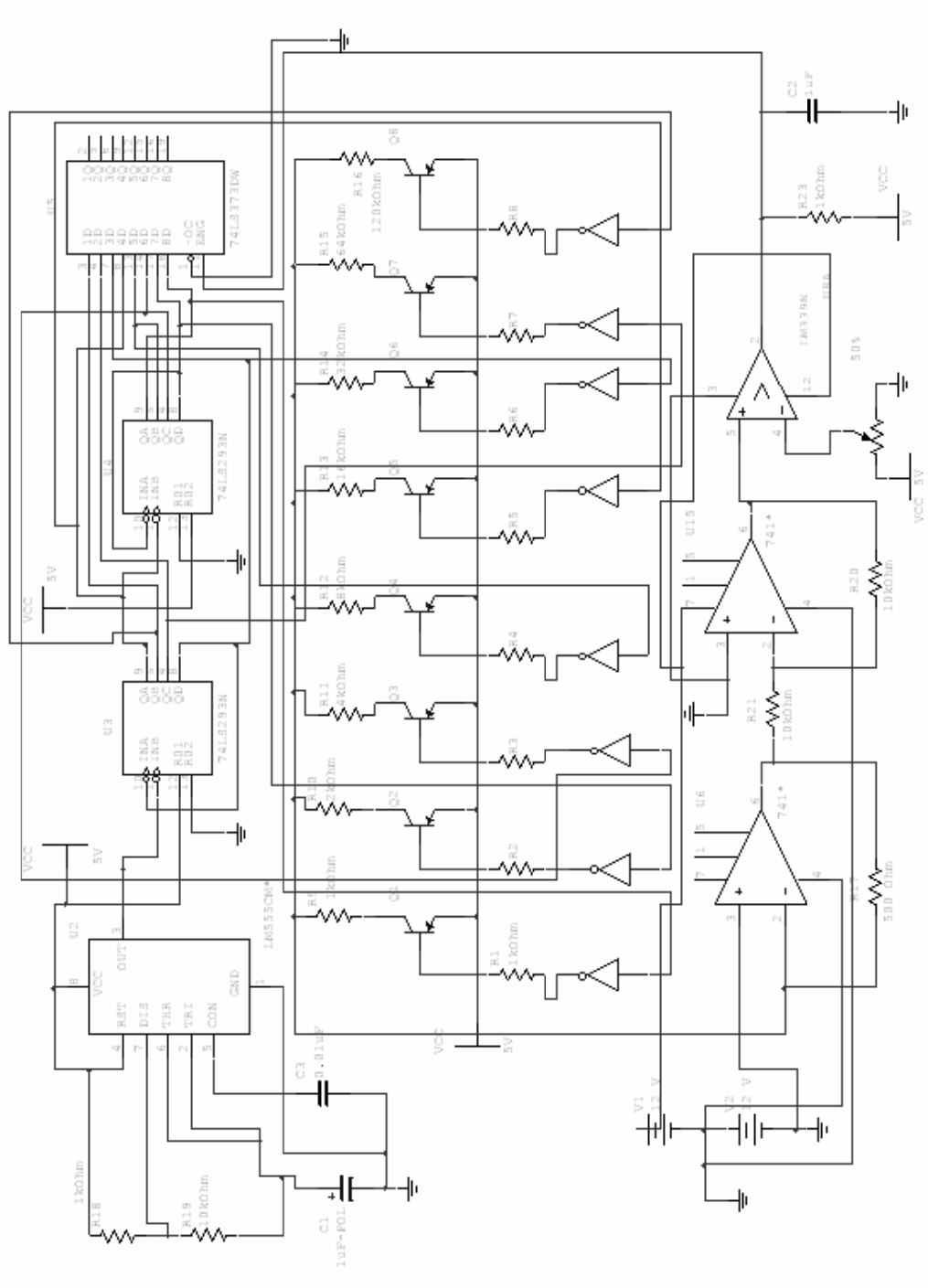
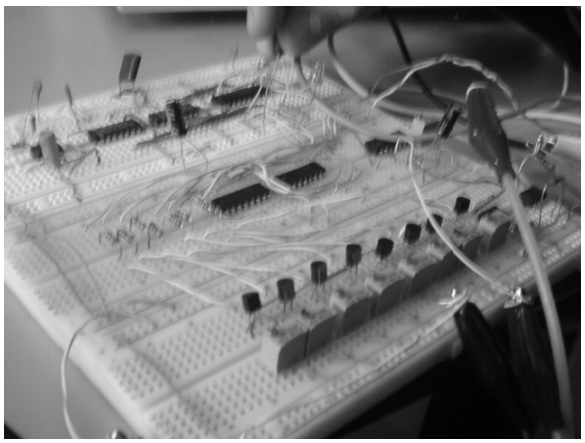


Diagrama para el ADC.

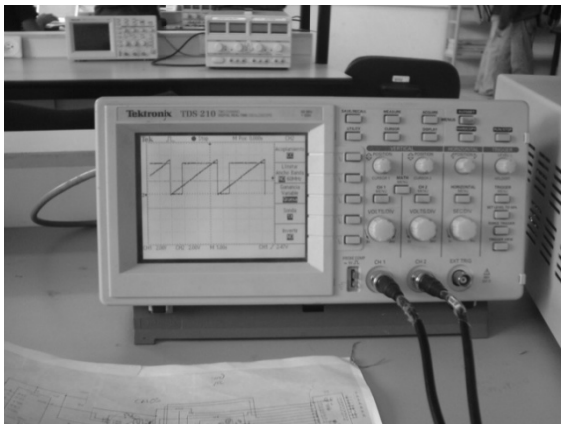
IMPLEMENTACION

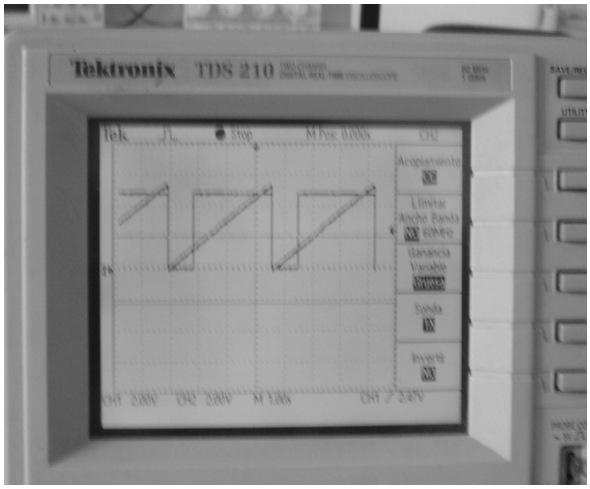
A continuación presentamos el circuito implentado para esta práctica.



RESULTADOS

A continuación presentamos los resultados obtenidos en esta práctica. Se observa la rampa obtenida por nuestro DAC.





BIBLIOGRAFIA

1. RUMSEY, Francis y McCORMICK, Tim. *Sonido y grabación. Introducción a las técnicas sonoras*. 2004.
2. WATKINSON, J. *El arte del audio digital*. IORTV, Madrid, 1993
3. Robert F. Coughlin, Frederick.Driscoll, *Amplificadores operacionales y circuito integrados lineales,* quinta edicion.
4. http://www.enterate.unam.mx/Articulos/dos/abril/audiodig.htm
5. http://www2.canalaudiovisual.com/ezine/books/acjirINFORMATICA/3info01.htm

Autor:

Pablo Gonzales Vellano.

Cuestionario.

Tema: Mecánica Clásica. La Fuerza de Gravedad.

Para Newton la gravedad es un caso particular de la fuerza que al actuar sobre la masa inercial de un cuerpo lo acelera. La fuerza de gravedad, que al menos obra entre dos cuerpos, actúa sobre sus masas gravitacionales instantánea y a distancia, en un espacio tridimensional plano absoluto e independiente de un tiempo absoluto uniforme y hace que los cuerpos aceleren hacia cada otro.

Para los físicos cuánticos la gravedad es la fuerza de interacción, entre cuerpos, que actúa a través del campo gravitatorio estático, compuesto de gravitones virtuales, emitidos de su masa-energía y viajan cerrados a c.

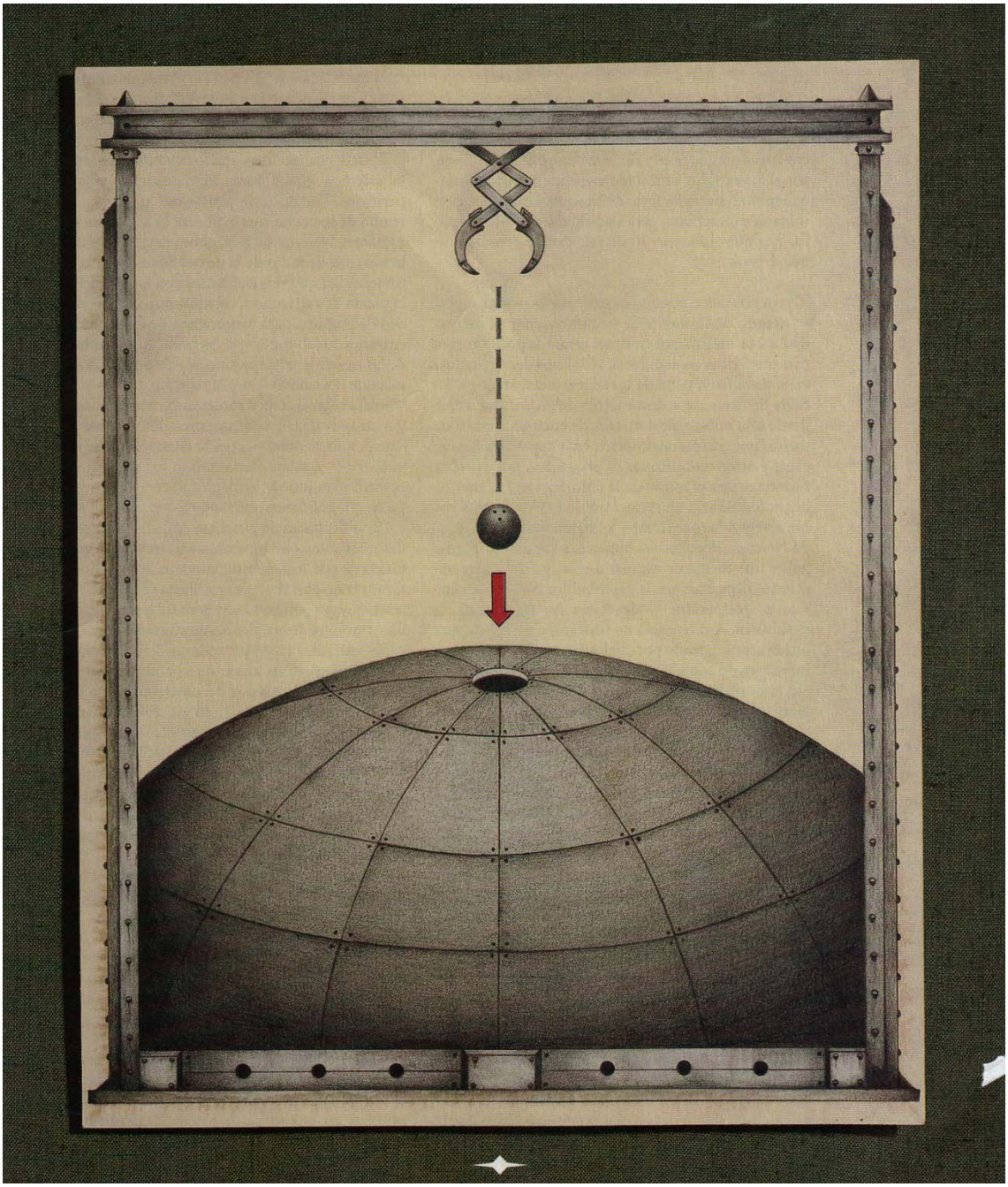
En la Relatividad General la inercia, del espacio plano, se generaliza al espacio-tiempo curvo. Así, los cuerpos, bajo la acción de un campo gravitatorio estático, sin gravitones y velocidad, inercialmente se mueven, con aceleración uniforme, dentro de las geodesias de la curvatura del espacio-tiempo de una región local, que los tiende a unir en su centro de masas.

A continuación presentamos unas serie de preguntas para que las medites y pienses un poco, resuélvelas y veras que será entretenido.

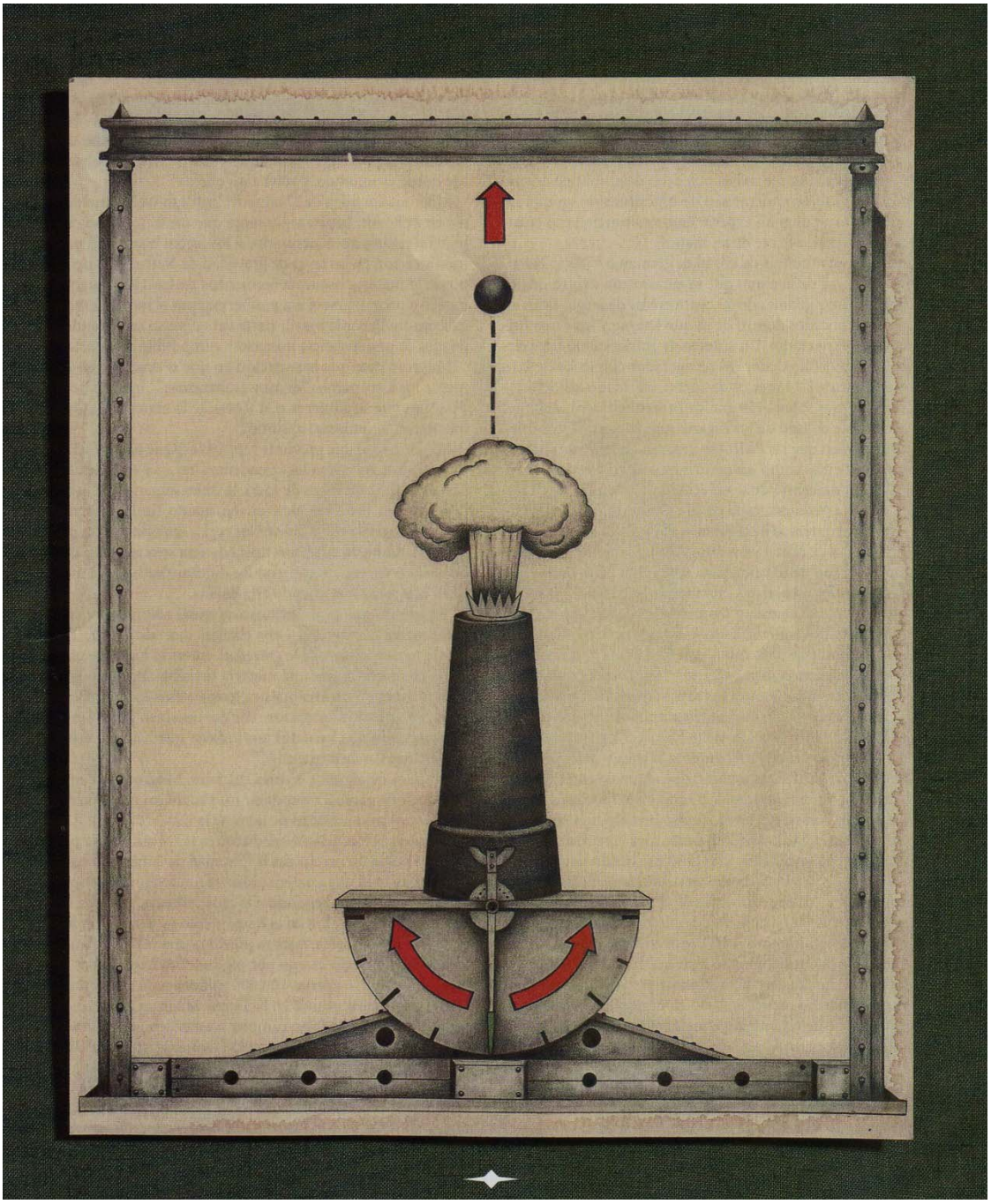
No es necesario hacer operaciones, sólo tienes que pensar y razonar sin estresarte.

¡Suerte!

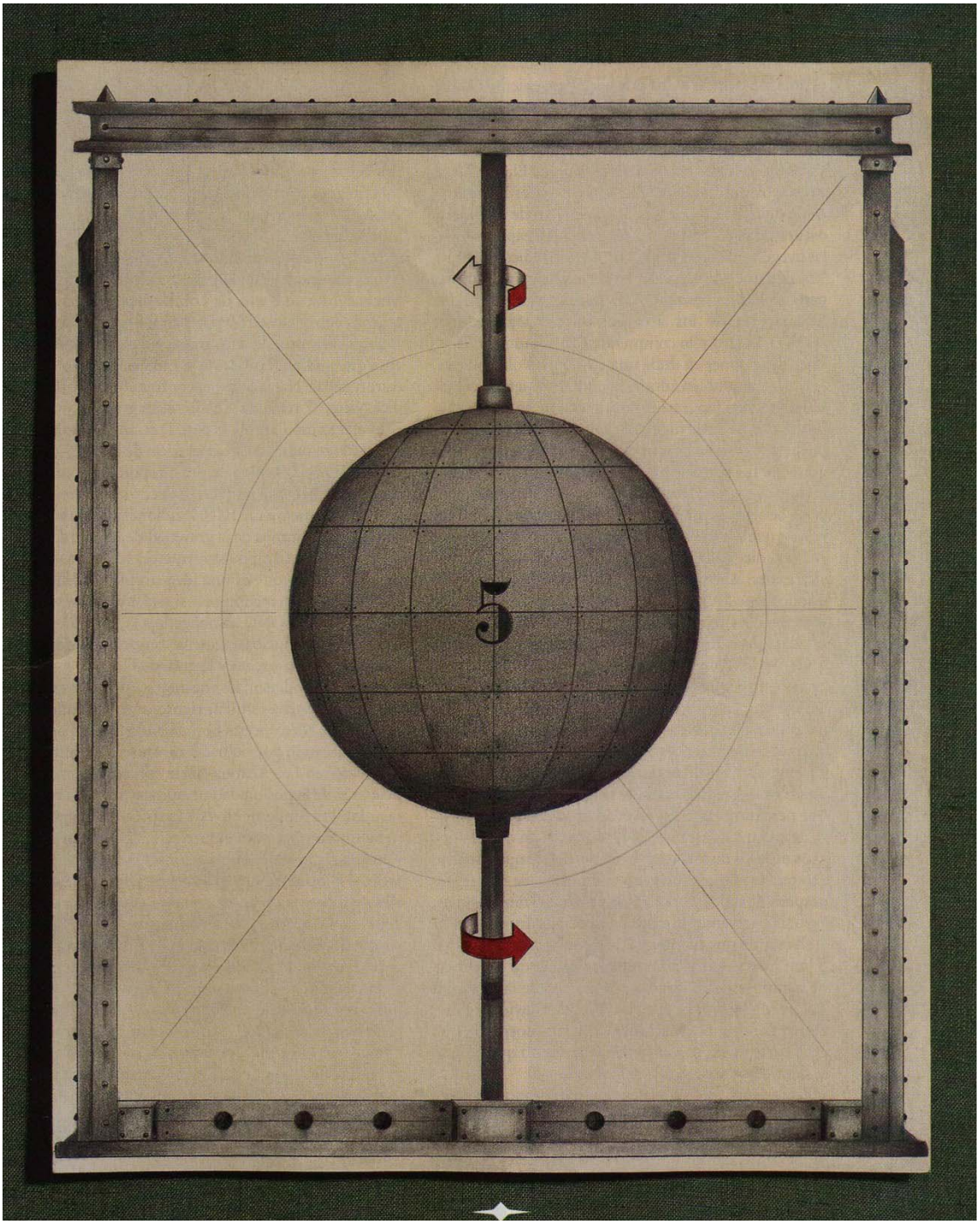
Pregunta 1: ¿Qué sucedería si usted horadara un hueco desde el Polo Norte hasta el Polo Sur, y dejara caer una bola dentro del hueco?

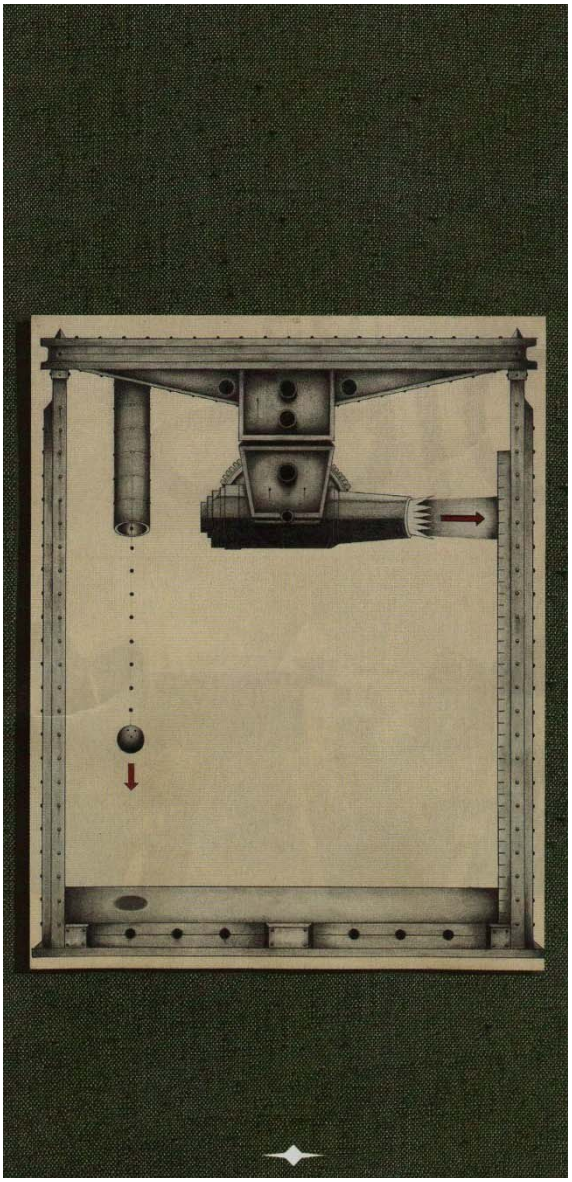


Pregunta 2: Si usted disparara una bola directamente hacia arriba (a 90 grados del horizonte), y una bala a un ángulo de 45 grados del horizonte, ¿cuál tocaría el suelo primero?



Pregunta 3: Si usted pudiera hacer girar la Tierra cinco veces más de prisa que su rotación actual de una vuelta diaria, ¿aumentaría notablemente la gravedad?





Pregunta 4: Si usted dejara caer una bola desde una altura de 3 metros, y simultáneamente disparara una bala paralela al horizonte, ¿cuál llegaría antes al suelo?