ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"Banco de pruebas para comunicaciones seriales I2C dedicado al trabajo con microcontroladores Atmel con aplicaciones especificas debidamente documentadas"

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentado por:

Manuel Lenín Mizhquero Ramos

Saúl Ignacio Ashqui Pagalo

GUAYAQUIL – ECUADOR

AÑO 2012

AGRADECIMIENTO

A Dios, a todos nuestros instructores que hicieron posible cumplir con este objetivo planteado en el transcurso de nuestra formación profesional.

Manuel Mizhquero R.

A mis padres y hermanos, por todo su esfuerzo, su apoyo y por la confianza que depositaron en mí. A Fundación Cristiana Esteban, por toda la ayuda que me brindaron desinteresadamente para poder culminar este objetivo. Y en especial a mi Padre Celestial, por su infinito amor y fidelidad que me rodea en los momentos más difíciles de mi vida.

Saúl Ashqui P.

DEDICATORIA

A nuestros padres, por todo el apoyo y esfuerzo brindado en los buenos y malos momentos.

TRIBUNAL DE SUSTENTACIÓN

_____

MSc. Carlos Valdivieso

Profesor Seminario de Graduación

_____

MSc. Hugo Villavicencio

Profesor Delegado del Decano

DECLARACIÓN EXPRESA

"La responsabilidad por los hechos, ideas y doctrinas expuestos en esta tesina, nos corresponde exclusivamente, y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL." (Reglamentos y exámenes y títulos profesionales de la ESPOL)

_____

Manuel Mizhquero Ramos

_____

Saúl Ashqui Pagalo

RESUMEN

En este proyecto se trata de desarrollar una Guía de referencia de Prácticas en base a la Comunicación I2C para MCU AVR-ATMEL. La cual se establece mediante el uso del módulo TWI, desarrollado en este proyecto, los ejercicios desarrollados incluyen los siguientes temas: Lectura y Escritura en Ensamblador a través de un Esclavo (EEPROM con interfaz 2C), Lectura y Escritura en Lenguaje C a través de un Esclavo (EEPROM con interfaz I2C).

ÍNDICE GENERAL

CONTENIDO

ÍNDICE DE GRÁFICOS

ÍNDICE DE TABLAS

ÍNDICE DE ABREVIATURAS

AVR

Advanced Virtual RISC, RISC Virtual Avanzado. También significa Alf Vergard RISC, en honor a sus creadores Alf Egil Bogen y Vergad Wollan.

COFF

Common Objet File Format. Formato del Archivo utilizado en el proceso de Depuración de software para microcontroladores.

EEPROM

Electrically Erasable Programmable Read Only Memory. Tipo de memoria de lectura solamente borrable y programable léctricamente.

HEX

Tipo de archivo hexadecimal que contiene únicamente datos para la memoria de programa de los microcontroladores.

IDE

Integrated Development Environment, Ambiente Integrado de Desarrollo. Software usado para el Desarrollo de Proyectos con Microcontroladores.

ISP

In System Programming. Característica que permite grabar o leer un dispositivo sin tener que extraerlo de su aplicación/sistema.

JTAG

Joint Test Action Group. Estándar IEEE 1149-1190 (Standard Test Access Port and Boundary-Scan Architecture). Se define como la electrónica que puede incorporar un

circuito integrado para asistir en el test, mantenimiento y soporte de placas de circuito impreso. Esta circuitería incluye una interfaz estándar a través de la cual se transfieren datos y comandos con los que el componente puede responder a un conjunto mínimo de instrucciones.

LCD

Liquid Crystal Display, Pantalla de Cristal Líquido. Están formadas por dos filtros polarizantes con filas de cristales líquidos alineados perpendicularmente; aplicando una corriente eléctrica a los filtros se consigue que la luz pase o no dependiendo de que lo permita o no el segundo filtro. Si se intercalan tres filtros adicionales de color, uno por cada color primario, se obtienen pantallas que reproducen imágenes en color.

LED

Light Emitting Diode, Diodo Emisor de Luz. Una tensión aplicada al semiconductor da como resultado la emisión de energía luminosa. Los LED se utilizan en paneles numéricos como los de los relojes digitales electrónicos y calculadoras de bolsillo

MCU

Microcontroller Central Unit. Es la unidad de procesamiento o núcleo de los microcontroladores.

Memoria FLASH

Chip de memoria no volátil (su contenido permanece aunque se desconecte de la fuente) que se puede reescribir. En cierto sentido se considera una variante de la EEPROM; la diferencia está en que mientras esta última se borra y programa al nivel de byte, la memoria flash se puede borrar y reprogramar en unidades de memoria llamadas bloques, cuyo tamaño puede ir desde los 512 bytes hasta los 256 KB. Esto hace que la memoria flash sea muy útil para actualizar la BIOS de un ordenador o computadora, o para almacenar cantidades de información importantes, como una

colección de imágenes o de documentos de texto, que se renuevan en una sola operación. Sus posibilidades de lectura y escritura son limitadas.

MIPS

Million Instructions Per Second. Medición de velocidad y el poder de un procesador tomando como referencia el número de instrucciones máquina que puede ejecutar en un segundo.

MLF

Micro Lead Frame. Tipo de Encapsulado de Chips.

RISC

Reduced Instruction Set Compiler. Procesadores que contienen una Colección Reducida de Instrucciones.

SCC

Segment Control Code, Código Controlador de Segmento del LCD.

SET

Escribir o configurar un bit con el valor de uno lógico.

SRAM

Static Random Access Memory. Memoria Estática de solo Lectura que conserva la información mientras esté conectada a la tensión de alimentación.

TAP

Test Access Port. Tipo de puerto que permite acceder y cambiar el estado de cada uno de los pines del componente, para la ejecución de pruebas con la interfaz JTAG.

TQFP

Thin Quad Flat Pack. Tipo de Encapsulado de Chips.

INTRODUCCIÓN

Este trabajo representa un preámbulo para la iniciación en el amplio universo de los sistemas de comunicación, tratando como tema central un sistema basado en la norma I2C. El contexto trata de una forma clara y amigable las principales características definidas por el protocolo que rige dicha comunicación. Previamente a desarrollar este tema, citaremos la clasificación de los sistemas de comunicación, los cuales se han dividido, según los que permiten conectar equipos electrónicos separados y los que permiten conectar circuitos integrados dentro de un mismo sistema.

a. Interconexión entre circuitos electrónicos dentro de un mismo equipo:

Comunicación en Paralelo	**Comunicación en Serie**
Paralelo	Microwire
Europa	SPI
VME	I2C
Futurebus	SCI
PCI	
AGP	

b. Interconexión entre equipos electrónicos:

Comunicación en Paralelo	**Comunicación en Serie**	
SPP	RS232	CAN
EPP	RS422	GPIB
ECP	RS485	Profibus
LVDS	4-20 mA	Lonworks
SCSI	V/F – F/V	InstaBus
	IrDA	One Wire
	Fibra Óptica	USB
	FDDI	FIRE WIRE
	HART	Ethernet
	MIDI	RF
	Power Line MODEM	GSM

Capítulo 1

Descripción General del Proyecto

1.1 Bus I2C

La abreviatura IIC o I2C significa Inter Integrated Circuit; es decir, que al momento de hablar del mismo, nos referimos a un bus cuyo campo de aplicación principal es la comunicación entre circuitos integrados. Al principio puede resultar un poco extraño este concepto, puesto que posiblemente se piense en la manera natural en la que se interconectan diversos circuitos integrados. Por ejemplo, se puede considerar un microprocesador y los diferentes circuitos que comúnmente forman parte de su sistema (memorias RAM, EEPROM, y periféricos como conversores A/D y D/A, puertos paralelos, pantallas alfanuméricas de cristal líquido, etcétera) entonces luego se puede pensar en que el modo normal de conectar dichos circuitos es por medio del uso del bus paralelo al nivel de componente o de sistema.

Ciertamente, esto puede permitir, conseguir el máximo rendimiento en términos de velocidad de la unidad central de procesos. Pero no se debe pensar solamente en un microprocesador y su circuitería asociada sino en un sistema de control basado en microcontrolador; en este caso muy frecuentemente el MCU ya contiene toda la

circuitería periférica que requiere el sistema, de manera que no se necesitará circuitería externa. No obstante, también es frecuente que el sistema que se pretenda diseñar requiera circuitería periférica con características de las que no dispone ningún miembro de la familia de controladores que se utiliza. Esto implica la necesidad de generar externamente al MCU los buses de dirección, de datos y de control para así poder conectarle externamente los circuitos oportunos. Esta cuestión, que conceptualmente es verificada y no presenta ningún tipo de dificultad, resulta un inconveniente en ciertos casos. Ciertamente en aquellos en los que es de primordial importancia minimizar el tamaño del diseño. No se pierda de vista que un circuito integrado cuantas más señales contemple mayor será el número de patillas que necesite y por tanto mayor será el tamaño del encapsulado utilizado y mayor espacio de placa de circuito impreso requerirá el trazado de las pistas de interconexión; por no hablar del coste. Por otro lado, cuanto mayor número de señales circulan por una placa de circuito impreso o interconectan sistemas tanto mayor es la posibilidad de emisión o captación de EMI. La clave del bus I2C está en la manera de ver los circuitos integrados. Aquí el concepto de comunicación entre sistemas se aplica a su nivel más bajo; es decir, cualquier tipo de circuito integrado, sea de la naturaleza que sea, se considera como un sistema cuyo modo de funcionamiento es controlable y que además requiere un canal de comunicación bidireccional con el sistema que lo controla. Este canal, que con ser semi-bidireccional es suficiente, se utilizará por un lado para que el sistema central defina cómo se comportará tal circuito – es decir, para transmitir órdenes – y por el otro lado para transmitir los datos que sean precisos en uno u otro sentido. Si se piensa en un módulo conversor dual A/D y D/A podrá captarse la idea: el conversor A/D podrá tener varios modos de funcionamiento (conversión única o continua, y en este último caso la frecuencia de la conversión); en

lugar de usar señales específicas se utilizaría el canal de comunicación para transmitir una orden de configuración operativa. Por otra parte, una vez realizada una conversión es necesario que el CAD envíe el dato convertido al procesador; y de igual manera éste deberá enviar al CDA el dato a convertir a analógico. Aunque se haya captado la idea, tal vez no se acabe de ver la ventaja de esta aproximación al problema. En cualquier caso, podrá pensarse, se necesitarán las señales que implementen el canal de comunicación así como las demás necesarias para sincronizar de alguna manera las transferencias por él de órdenes y de datos, con lo que la economía de señales no parece evidente en comparación con las que poseen los circuitos normales. Grave error, puesto que el número de señales que necesita un determinado bus de comunicación depende del modo de transporte de la información así como de la funcionalidad deseada.

En primer lugar, si se opta por una comunicación serie en lugar de paralela entonces se puede ahorrar un gran número de señales en el bus. Queda ahora por determinar las características funcionales de este para poder así definir el número imprescindible de señales adicionales de protocolo. En este punto es posible pensar, a semejanza del bus EIA-232, en la posibilidad de utilizar una técnica serie asíncrona totalmente bidireccional. De esta manera existiría una línea de transmisión, otra de recepción y el común de las señales; es decir, se tendría un bus de tres hilos que permitiría el intercambio de información entre sistemas; y en el plano eléctrico podrían utilizarse, por simplicidad, señales con niveles TTL si la distancia máxima del bus no pudiese ser nunca excesiva. En el plano lógico quedaría por definir un protocolo de estructura de mensajes para los distintos tipos de circuitos que se pudiesen conectar. Pues bien, esta solución no es aceptable en este caso por múltiples motivos que más adelante se

irán analizando, aunque uno de ellos – que no es el principal – sería el pobre rendimiento que ofrecería un enlace serie asíncrono en lo que se refiere a la información neta transmitida (no se olvide que cada caracter ha de ir enmarcado por un bit de inicio y otro de parada); por no hablar de la imposibilidad de las topologías multipunto.

Por tanto, ¿qué características debería tener un bus pensado para el enlace entre los circuitos integrados de un sistema microcomputador o microcontrolador? Si se analiza la cuestión, teniendo en mente que el número de señales ha de ser el menor posible, podrá comprenderse que son las siguientes:

- Será un bus serie síncrono. El término síncrono deberá entenderse como que existirá, además de la línea de datos, una señal de sincronía explícita para validar los datos en el canal serie, y no en el sentido de que en la línea de datos siempre deberá existir un flujo de datos, bien propiamente como tales o bien como medio para mantener la sincronía.
- El sentido del enlace será semibidireccional. Es decir, existirá una única línea de datos que podrá utilizarse para el flujo en ambos sentidos, pero no simultáneamente. De esta manera se ahorra el número de señales del bus. Esta limitación es un hecho menor debido al ámbito de aplicación de este tipo de bus, en el que no se necesita la bidireccionalidad.
- Deberá admitir topologías multipunto. Es decir, podrán conectarse al bus varios dispositivos, pudiendo actuar varios de ellos como emisores, aunque no simultáneamente.

- Un mismo dispositivo podrá actuar como emisor o como receptor, en distintos momentos.
- Deberá admitir topologías multimaestro. Es decir, más de un dispositivo puede intentar gobernar el bus. Un maestro podrá actuar tanto como emisor como receptor; y lo mismo puede decirse de un esclavo. Lo que diferenciará a un maestro de un esclavo es la capacidad de gobierno del bus (suministro de la señal de sincronía).
- Deberá establecer un mecanismo de arbitraje para el caso en que simultáneamente más de un maestro intente gobernar el bus.
- Un maestro podrá funcionar también como un esclavo. El motivo es que en las topologías multimaestro un maestro que haya ganado el gobierno del bus pueda dirigirse a cualquier otro maestro, que entonces deberá comportarse como un esclavo.
- Deberá establecer un mecanismo de adaptación de velocidad, para el caso en que un esclavo no pueda seguir la velocidad impuesta por el maestro.
- La velocidad de las transferencias, marcada por el maestro, podrá ser variable dentro de unos límites. Además, por su ámbito de aplicación, la tasa de transferencia máxima podrá ser relativamente pequeña, entre 100 y 400 kbps normalmente.
- Deberá establecer un criterio de direccionamiento abierto por medio del cual se sepa a qué esclavo se dirige un maestro. Además, la dirección asignada a un circuito podrá ser modificada dinámicamente (no confundir el concepto de direccionamiento de sistema con el de direccionamiento de una memoria). Por otra parte, el mecanismo de direccionamiento deberá admitir futuras ampliaciones del número máximo de elementos direccionables.

- El formato de las tramas transmitidas habrá de ser suficientemente flexible como para poder adaptarse a la funcionalidad de cualquier tipo de circuito pasado, presente o futuro.
- Desde el punto de vista de la capa física de la interfaz, deberá admitir la conexión de circuitos de diferentes tecnologías (diversas bipolares, MOS, etcétera).

El bus I2C fue desarrollado por Philips al inicio de la década de 1980, teniendo en mente todos estos aspectos, y ha encontrado una gran aceptación en el mercado. Actualmente los principales fabricantes de dispositivos semiconductores ofrecen circuitos que implementan un bus I2C para su control.

El Comité I2C es el encargado, entre otras cuestiones, de asignar las direcciones I2C a cada tipo de circuito. De esta manera cada función implementada, independientemente del fabricante, posee la misma dirección; es decir, circuitos que realicen funciones equivalentes deberán poseer la misma dirección oficial I2C independientemente del fabricante. Antes de continuar es conveniente aclarar una cuestión; se ha dicho que cualquier circuito integrado puede poseer un bus I2C, y por tanto existen circuitos de memoria RAM y ROM (EEPROM) pero su finalidad no estriba en contener datos normales y programas sino parámetros funcionales que han de accederse poco a poco. En el caso de la EEPROM lo común es utilizarla para contener parámetros que definen la configuración del sistema y que han de retenerse en ausencia de alimentación (por ejemplo, la sintonía en un aparato de TV). Los circuitos de RAM y ROM habituales serán, por supuesto, de tipo paralelo tradicional; dicho de otra manera, las memorias I2C nunca se encuentran mapeadas en la memoria del sistema.

1.2 Protocolo del bus I2C

El bus I2C sólo define dos señales, además del común:

- **SDA**. Es la línea de datos serie (Serial Data, en inglés), semibidireccional. Eléctricamente se trata de una señal a colector o drenador abierto. Es gobernada por el emisor, sea éste un maestro o un esclavo.

- **SCL**. Es la señal de sincronía (reloj serie, o Serial Clock en inglés). Eléctricamente se trata de una señal a colector o drenador abierto. En un esclavo se trata de una entrada, mientras que en un maestro es una salida. Un maestro, además de generar la señal de sincronía suele tener la capacidad de evaluar su estado; el motivo – como se verá – es poder implementar un mecanismo de adaptación de velocidades. Esta señal es gobernada única y exclusivamente por el maestro; un esclavo sólo puede retenerla o pisarla para forzar al maestro a ralentizar su funcionamiento, cosa que se explicará más adelante.

Esta particularidad física de que las salidas de los excitadores I2C hayan de ser a colector o drenador abierto no es casual sino que resulta de vital importancia para que a este bus con tan sólo una señal de datos y otra de sincronía se le pueda dotar de todas las características funcionales apuntadas en el apartado anterior. Recuérdese que un dispositivo lógico con este tipo de salida permite realizar la función producto lógico con una simple conexión eléctrica (montaje Y por conexión). Evidentemente, un enlace I2C necesitará sendas resistencias de elevación en las respectivas líneas

SDA y SCL. De esta manera un excitador I2C realmente sólo gobierna el estado 0 lógico en las líneas I2C, mientras que el estado 1 lógico no es suministrado por el excitador directamente sino por medio de la oportuna resistencia de elevación. Así, el concepto de aislamiento del bus se consigue con tan sólo hacer que el excitador del nodo que se desea aislar ofrezca a su salida el estado alto (1 lógico), al que le corresponde una elevadísima impedancia de salida, equivalente a un aislamiento eléctrico respecto al bus.

1.3 Estados del bus I2C

El bus I2C puede encontrarse en distintos estados, que la norma define como los siguientes:

- Libre. Este estado se caracteriza por encontrase las líneas SDA y SCL a 1, sin que se esté realizando ningún tipo de transferencia.

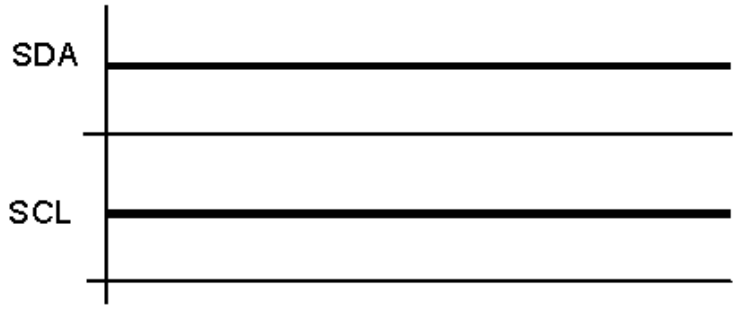


Figura 1.1 Condición de bus libre

- Inicio. Se produce una condición de inicio cuando un maestro inicia una transacción. En concreto, consiste en un cambio de alta a baja en la línea SDA mientras SCL permanece a alta. Esto puede apreciarse en el cronograma

mostrado en la figura 1.2. A partir de que se dé una condición de inicio se considerará que el bus está ocupado y ningún otro maestro deberá intentar generar su condición de inicio.

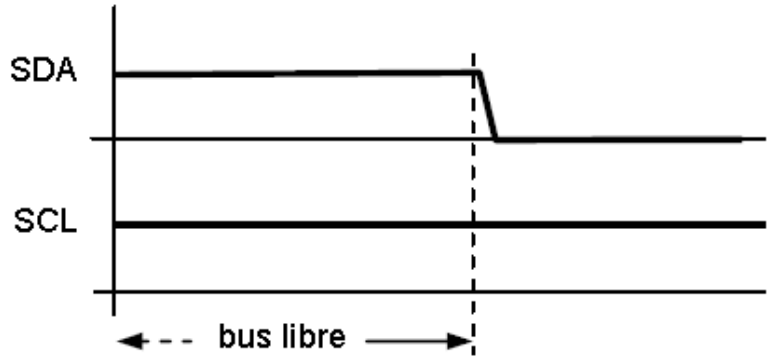


Figura 1.2 Condición de inicio

- Cambio. Se produce una condición de cambio cuando, estando a baja la línea SCL, la línea SDA puede cambiar de estado. En la transferencia de datos por un bus I2C éste es el único instante en el que el sistema emisor (que podrá ser tanto un maestro como un esclavo) podrá poner en la línea SDA cada bit del caracter a transmitir.

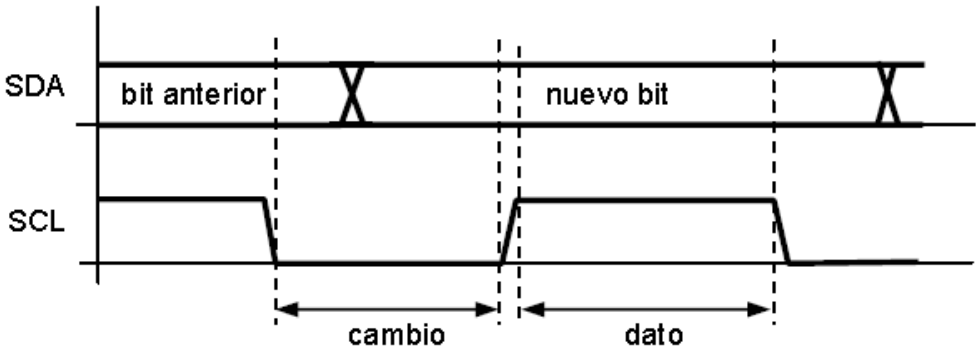


Figura 1.3 Condiciones o estados de cambio y de dato

- Dato. Este estado es el que, una vez iniciada una transacción, queda definido por la fase alta de la señal de sincronía SCL. En este estado se considera que el

dato emitido es válido, y no se admite que pueda cambiar. Recuérdese que, ya iniciada una transferencia, el único instante en que la línea de datos puede cambiar es en el estado de cambio.

- Parada. Se produce una condición de fin o parada cuando, estando la línea SCL a alta, se produce un cambio de baja a alta en la línea SDA. Obsérvese que esto es una violación de la condición de dato, y es precisamente por esto por lo que se utiliza para que un maestro pueda indicar al esclavo que se finaliza la transferencia. Tras la condición de parada se entra automáticamente en el estado de bus libre.

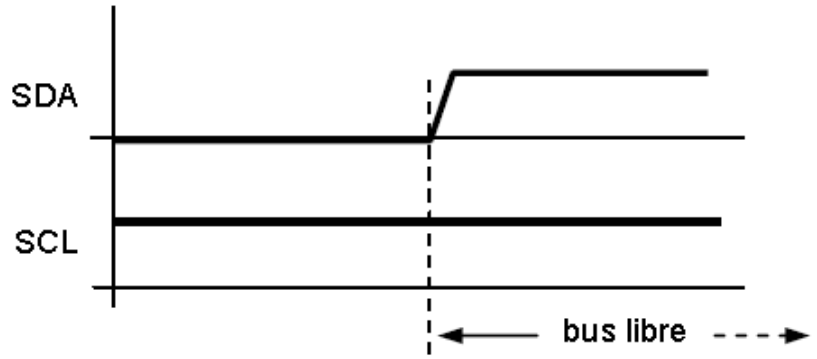


Figura 1.4 Condición de parada

1.4 Formato de una transacción

Entenderemos por transacción todos los eventos desarrollados entre una condición de inicio y una de parada. Una vez producido el inicio de una transacción comienza propiamente ésta, y en ella se lleva a cabo la transferencia de uno o varios caracteres, en uno u otro sentido.

- El caracter de ocho bits es la unidad de transferencia. El orden de emisión de los bits de un caracter es empezando por el más significativo, siguiendo en orden decreciente de pesos y terminando por el menos significativo (como se ve, es un criterio contrario al que siguen las UART).

- Un caracter puede tener diversos significados en función de quién lo emita y en qué instante lo haga.

- Tras cada caracter se debe producir un reconocimiento por parte del receptor; el motivo es dotar al protocolo de un mecanismo de seguridad.

- El primer caracter transferido lo emite siempre el maestro; sus siete bits más significativos indican la dirección del esclavo al que se dirige, y el bit de menor peso indica el sentido de la transferencia de los subsiguientes caracteres (0=escritura, 1=lectura, siempre desde el punto de vista del maestro).

- Dentro de una transacción es posible cambiar el sentido del flujo, pero para ello hace falta generar una nueva condición de inicio (reinicio) y direccionar de nuevo el mismo esclavo.

Como puede verse, para el direccionamiento de un esclavo se tienen siete bits, lo que daría un máximo de 128 dispositivos I2C distintos. No obstante, en la realidad se dispone de muchas menos direcciones puesto que un cierto número de ellas no se emplean como tales sino como identificadores de función especial, como se verá más

adelante; además, ciertos circuitos reservan para sí varias direcciones debido a que es frecuente que en un diseño se tengan que emplear varios del mismo tipo.

En estos casos los circuitos tienen una dirección compuesta por dos partes; una fija establecida por el comité I2C y otra variable (bits inferiores) fijada por el diseñador de cada sistema dado. En el encapsulado de estos circuitos existen ciertas patillas (una, dos o tres) mediante las cuales el diseñador puede determinar la parte programable de la dirección I2C. Por ejemplo, los circuitos EEPROM tienen asignada una dirección 1010XXX; 1010 es la parte fija de los siete bits, y XXX es la parte programable que corresponde al estado aplicado a tres patillas de su encapsulado.

Retomando la cuestión de la limitación del número de direcciones, este hecho trajo aparejado el que pocos años después del lanzamiento del bus I2C, y debido a su éxito, se fuesen agotando las direcciones disponibles, lo que implicaba la imposibilidad de ofrecer nuevos circuitos I2C. Afortunadamente, se previó esta contingencia dado que se reservaron para futuras ampliaciones las direcciones que comenzaban por 1111. De esta manera, ahora existen dos tipos direcciones: las normales de siete bits y las ampliadas de diez bits. En este último caso el direccionamiento de un esclavo implica la transferencia de dos caracteres; el primero deberá comenzar por 11110XX, siendo XX los dos bits de mayor peso de la dirección de diez bits, y el segundo caracter corresponderá a los ocho bits inferiores de la dirección del esclavo. El sentido de la transacción se sigue indicando por el bit menos significativo del primer caracter de dirección (11110-XX-L/E).

En términos generales, una transacción obedecerá a alguna de las siguientes estructuras:

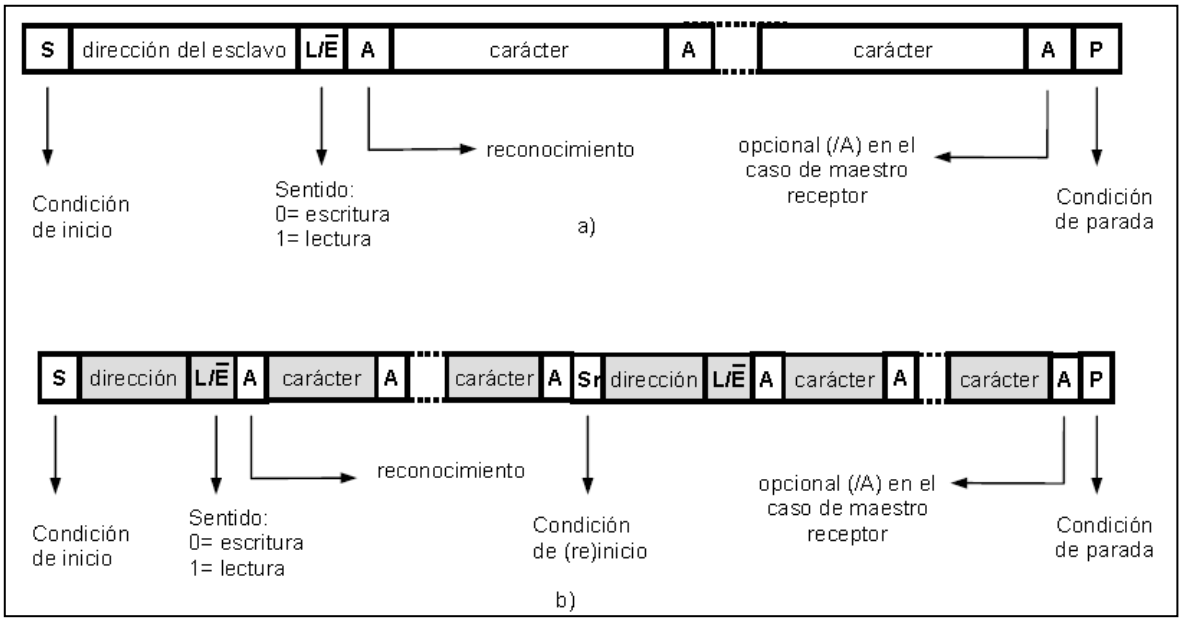


Figura 1.5 Formatos de una transacción: a) sin cambio de sentido b) con redireccionamiento y posible cambio de sentido

En la figura 1.5 se ha ilustrado el direccionamiento con direcciones de siete bits; si se utilizase un direccionamiento de diez bits entonces la parte del direccionamiento quedaría como se observa en la figura 1.6.

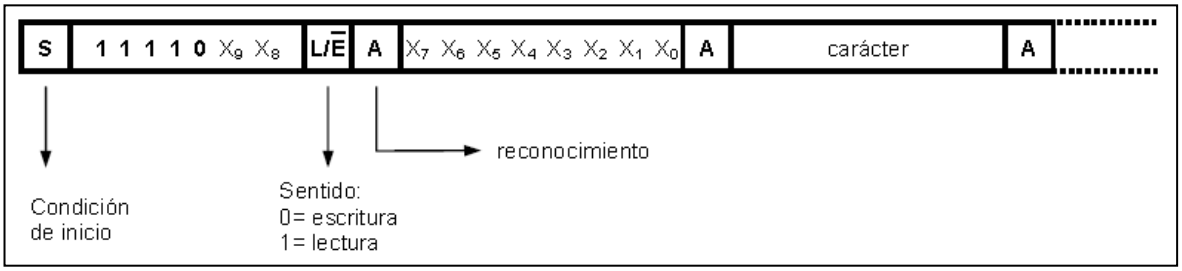


Figura 1.6 Direccionamiento ampliado de diez bits

En la figura precedente se indica por X9...X0 los diez bits de la dirección del esclavo al que se dirige el maestro.

1.4.1 Significado de los caracteres

Como ya se ha anticipado, la norma I2C no indica absolutamente nada sobre el significado de los caracteres que siguen a la fase de direccionamiento. Esto dependerá del circuito I2C de que se trate en cada caso. Por ejemplo, si se tratase de un circuito tipo 24C02, que es una memoria EEPROM, y se desease leer cuatro datos almacenados a partir de la posición relativa 20h desde su origen, entonces la transacción que realizaría el maestro sería la siguiente:

- Generar la condición de inicio.

- Direccionar en modo escritura el circuito 24C02 oportuno.

- Verificar el reconocimiento al direccionamiento.

- Enviar un caracter de valor 20h, que sería interpretado por la EEPROM como la dirección a partir de la que se va a hacer un acceso.

- Como lo que se va a hacer es leer (recibir), y antes se ha establecido el modo de escritura (transmitir), entonces tras el reconocimiento a 20h se genera una condición de reinicio y se vuelve a direccionar el circuito.

- Tras verificar el reconocimiento, se solicitan los ocho bits de cada caracter, generando el reconocimiento tras el último (menos significativo) de ellos. Esto

se repite cuatro veces. A cada lectura de caracter la EEPROM irá enviando los contenidos de sus posiciones 20h, 21h, 22h y 23h.

- Una vez que el maestro ha leído las cuatro posiciones, no reconocerá el último y generará una condición de parada.

Pero, por supuesto, si se tratase de otro circuito con otro tipo de funcionamiento, entonces el significado de los caracteres no tendría absolutamente nada que ver con lo anteriormente descrito. Los fabricantes de dispositivos I2C ofrecen en sus hojas de características información suficiente como para saber el significado de los caracteres así como qué hacer para conseguir realizar las funciones que tal circuito permita.

1.4.2 Gobierno de la señal de sincronía

Ya se ha indicado que la señal SCL sólo la puede generar el maestro. Si el maestro actúa como emisor entonces la señal de sincronía puede entenderse como una validación de los bits que va volcando en la línea de datos SDA, sean estos bits los de un caracter de dirección o los de un caracter propiamente de datos. Si actúa como receptor entonces SCL sirve no sólo para indicar los instantes en los que el maestro toma los bits que le envía el esclavo, sino también para – indirectamente – marcar la velocidad de envío del siguiente bit, el esclavo no debe poner el siguiente bit hasta que se dé una condición de cambio, y ésta no se da hasta que finaliza el estado actual de dato (esto es, hasta que SCL pasa de alta a baja). Por otra parte, ya se ha dicho que la unidad de transferencia es el caracter de ocho bits, y que a cada caracter le debe seguir un bit de reconocimiento por parte del receptor. Será siempre el maestro quien

validará el bit de reconocimiento, sea éste ofrecido por él mismo (maestro receptor) o por el esclavo (esclavo receptor).

Finalmente, sólo hay una situación en la que un esclavo puede interferir la señal SCL gobernada por el maestro. Se trata del mecanismo denominado sincronización, por medio del cual un esclavo puede ralentizar la velocidad de transferencia marcada por el maestro. En este caso el esclavo no gobierna el reloj pero interfiere con cierto criterio su estado para que el maestro advierta que se le está pidiendo que vaya más lento. Esto se analizará más adelante.

1.4.3 Filosofía del reconocimiento

Como se ha dicho antes, el establecer la necesidad de que el receptor reconozca cada caracter transmitido obedece al deseo de establecer un mecanismo de seguridad en las transferencias, pero también a algo más. En primer lugar, el reconocimiento es obligatorio para todo esclavo receptor. Si uno no lo hace entonces el maestro emisor considerará que se ha producido un fallo en el bus y deberá establecer los mecanismos oportunos de respuesta a este fallo. Dependiendo de las circunstancias por las cuales se produzca esta ausencia de reconocimiento, entonces se podrá obrar de manera distinta. Por ejemplo:

- Si la ausencia de reconocimiento del esclavo receptor se produce en la fase de direccionamiento esto podrá significar varias cosas:

    - Que el circuito se encuentra ausente (removido del bus, o apagado).

  - Que el circuito está ocupado realizando algún tipo de tarea interna que le impide responder al direccionamiento del maestro. Esto podrá tener sentido en algunos dispositivos I2C, pero no en otros. Por ejemplo, en las memorias EEPROM con bus I2C si se les escribe un bloque de datos se tarda un cierto tiempo en llevar a cabo tal proceso, durante el cual no dará el reconocimiento si es vuelta a direccionar para realizar un nuevo acceso.
  - Que el circuito está averiado o ha sucedido alguna anomalía en la línea.

- Pero si esta ausencia se produce en medio de una transferencia de datos entonces esto normalmente será síntoma de una avería funcional (a menos que dé la casualidad de que en ese instante ha sido removido o apagado).

El tipo de medida a tomar por parte del maestro dependerá de cada circunstancia, y podrá ir desde volver a intentar direccionar el esclavo un poco más adelante hasta generar algún tipo de alarma.

En concreto, el reconocimiento consiste en poner la línea de datos SDA a baja durante el impulso SCL que sigue al del último bit de un caracter; y esto lo debe realizar el receptor.

En el instante que toca el reconocimiento el emisor deberá aislar del bus su línea SDA; en los bits de los caracteres (estados de datos) es él quien la gobierna –no se olvide que actúa como emisor– pero no sucede tal cosa en el reconocimiento. Aquí quien ha de gobernar ahora SDA es el receptor, para dar el reconocimiento. Por ello, si el emisor no aísla del bus su SDA local, y diese la casualidad de que el bit menos

significativo, y último del caracter transmitido, fuese un 0, entonces podría suceder que si el receptor no diese su reconocimiento el maestro no advertiría tal eventualidad puesto que al sondear el estado de la línea SDA detectaría que está a baja (por el último bit de dato transmitido) e interpretaría erróneamente que ha existido el reconocimiento. En la siguiente figura puede apreciarse lo dicho.

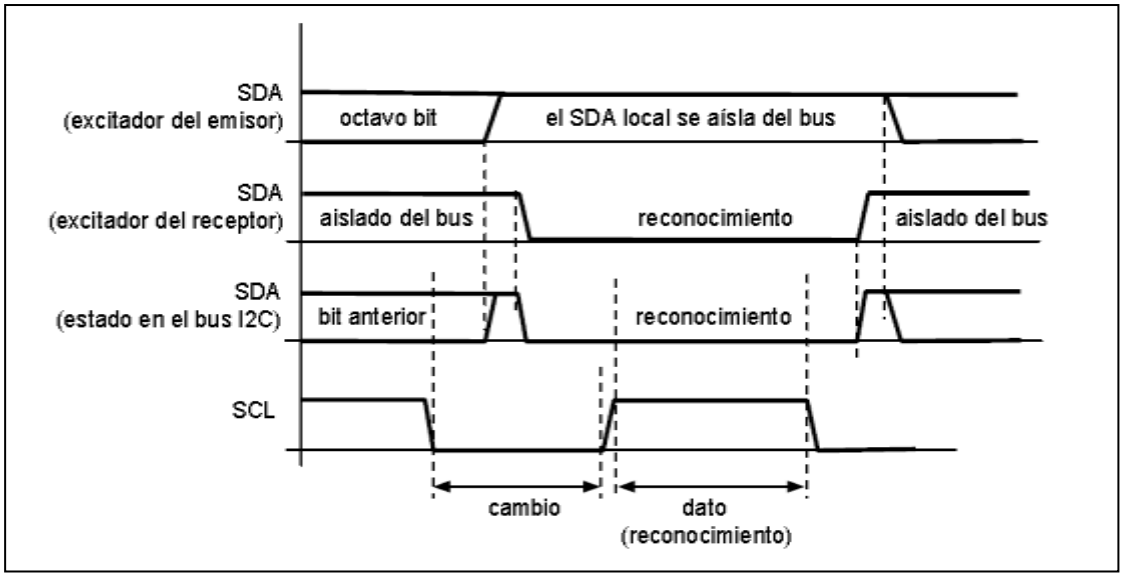


Figura 1.7 Proceso de reconocimiento en el bus I2C

En la figura 1.7, se ha mostrado el estado de los excitadores SDA del emisor y del receptor I2C. No se pierda de vista que la capa física de la interfaz I2C necesita la existencia de un excitador y de una electrónica receptora tanto para SDA como para SCL. Cuando alguien actúa como receptor, debe aislar del bus su excitador SDA (ofrecer un 1 lógico, como ya se ha mencionado), y también SCL si es un esclavo, para que el excitador del emisor sea quien gobierne la línea SDA del bus. Pero cuando llega la hora de dar el reconocimiento entonces es el excitador del emisor quien debe aislarse para que, ahora sí, sea el del receptor quien aplique un nivel a baja en SDA. De esta manera, si se observa la figura 1.7 puede verse que la señal en la línea SDA del bus I2C es el producto lógico de las señales aplicadas por los excitadores de todos

y cada uno de los circuitos conectados al bus. Tal y como ya se ha comentado, ésta es una propiedad de los excitadores con salida a colector o drenador abierto.

Aunque se ha dicho que el reconocimiento es obligatorio por parte del receptor, existe una excepción a esta regla: cuando el receptor es un maestro entonces es posible no dar el reconocimiento al último caracter, justo antes de generar la condición de parada. Para entender el motivo de esto es necesario recordar que es el maestro quien inicia una transacción, la gobierna y la finaliza. Por tanto, aunque un maestro actúe como receptor sabrá perfectamente cuándo no quedan más datos por recibir. Dicho de otra manera, un esclavo emisor si tiene algo que enviar no lo hará por propia iniciativa sino porque el maestro receptor se lo solicite, y en este caso el maestro receptor sabrá perfectamente cuántos datos desea tomar. Supóngase ahora un caso concreto en el que se desea leer un bloque de siete datos de una memoria EEPROM. Podrá pensarse que esta circunstancia no planteará mayor problema dado que bastará, al recibirse el séptimo caracter, reconocerlo y generar entonces una condición de parada para poder iniciar una nueva transacción con otro circuito I2C.

Pues bien, si el maestro receptor reconociese el caracter antes de generar la condición de parada podría suceder que tal condición de parada no se llegase a producir efectivamente en el bus I2C. El motivo es el siguiente: cuando el maestro receptor da su reconocimiento al caracter enviado por el esclavo emisor, éste supondrá que a continuación el maestro le solicitará el primer bit del siguiente caracter, el octavo, dado que no tiene por qué saber cuántos datos le solicitará el maestro. Por ello el esclavo, cuando SCL pase a baja y se produzca un estado de cambio, colocará en la línea SDA el bit más significativo del siguiente dato, y esperará el impulso SCL del

maestro. Supongamos que este bit da por casualidad que es un '0' lógico. ¿Qué sucederá entonces? Pues que el maestro receptor, que desea finalizar la transacción y para ello intenta generar una condición de parada, se encontrará con que no puede hacerlo porque el esclavo emisor no ha liberado la línea SDA del bus. Esto puede apreciarse en la figura 1.8.

Es por este motivo por el que es opcional el último reconocimiento de un maestro receptor; como el esclavo emisor obligatoriamente tiene que liberar la línea SDA para permitir que el maestro receptor reconozca, pues entonces éste puede, si así lo desea, aprovechar este lapso para tener la garantía de poder generar una condición de parada.

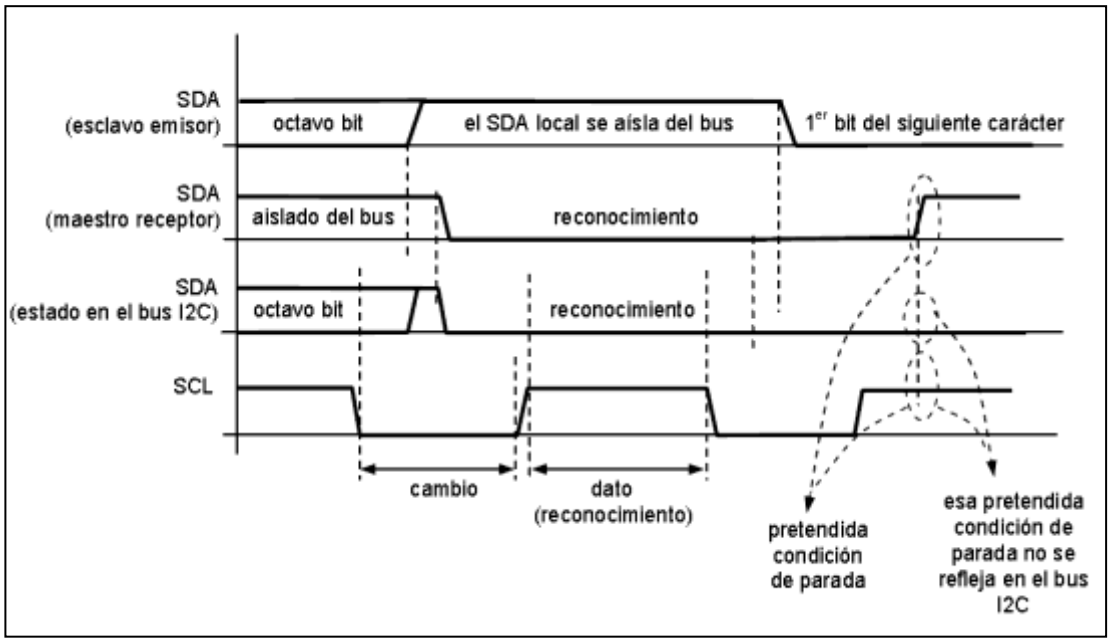


Figura 1.8 Problema planteado de una parada tras un reconocimiento

Cuando el receptor es un maestro también puede intencionadamente no producirse el reconocimiento en medio de una transacción. Esto se haría para abortarla ante situaciones que hacen aconsejable tal acción. Supóngase que en medio de una

transacción, al leerse un caracter cualquiera de una trama, sucede un evento que hace que el maestro opte por abortar la transacción e iniciar otra con otro dispositivo I2C.

Ya se ha visto que si se reconoce ese caracter podría suceder que el maestro no pudiera generar la condición de parada y por tanto no podría iniciar una nueva transacción con el nuevo circuito I2C con el que ahora urge comunicarse. Al no reconocer, se aprovecha este instante para forzar una condición de parada que aborte la transacción en curso, que podrá reanudarse más adelante.

Es necesario advertir que la manera ortodoxa de que un maestro receptor realice con garantía la condición de parada es no aprovechar el instante del reconocimiento sino el siguiente tiempo de bit. En este caso el maestro receptor no reconoce pero aplica el correspondiente impulso de sincronía. Cuando un esclavo emisor detecta que no se ha reconocido el dato enviado, la norma I2C obliga a que justo a continuación aísle del bus su SDA. De esta manera ya el maestro receptor está en condiciones de generar sin problemas su condición de parada.

Este segundo método es el que especifica la norma, pero el que se ha comentado en primer lugar resulta igualmente efectivo y más rápido de llevar a cabo (la única precaución que hay que tener es conocer cómo se implementa la interfaz I2C en los esclavos utilizados; si una condición de parada se detecta de manera cableada entonces no habrá problema, pero si se hace por programa entonces se deberá verificar que se comprueba en cualquier instante y no sólo tras cada reconocimiento). Lo comentado puede observarse en la figura 1.9.

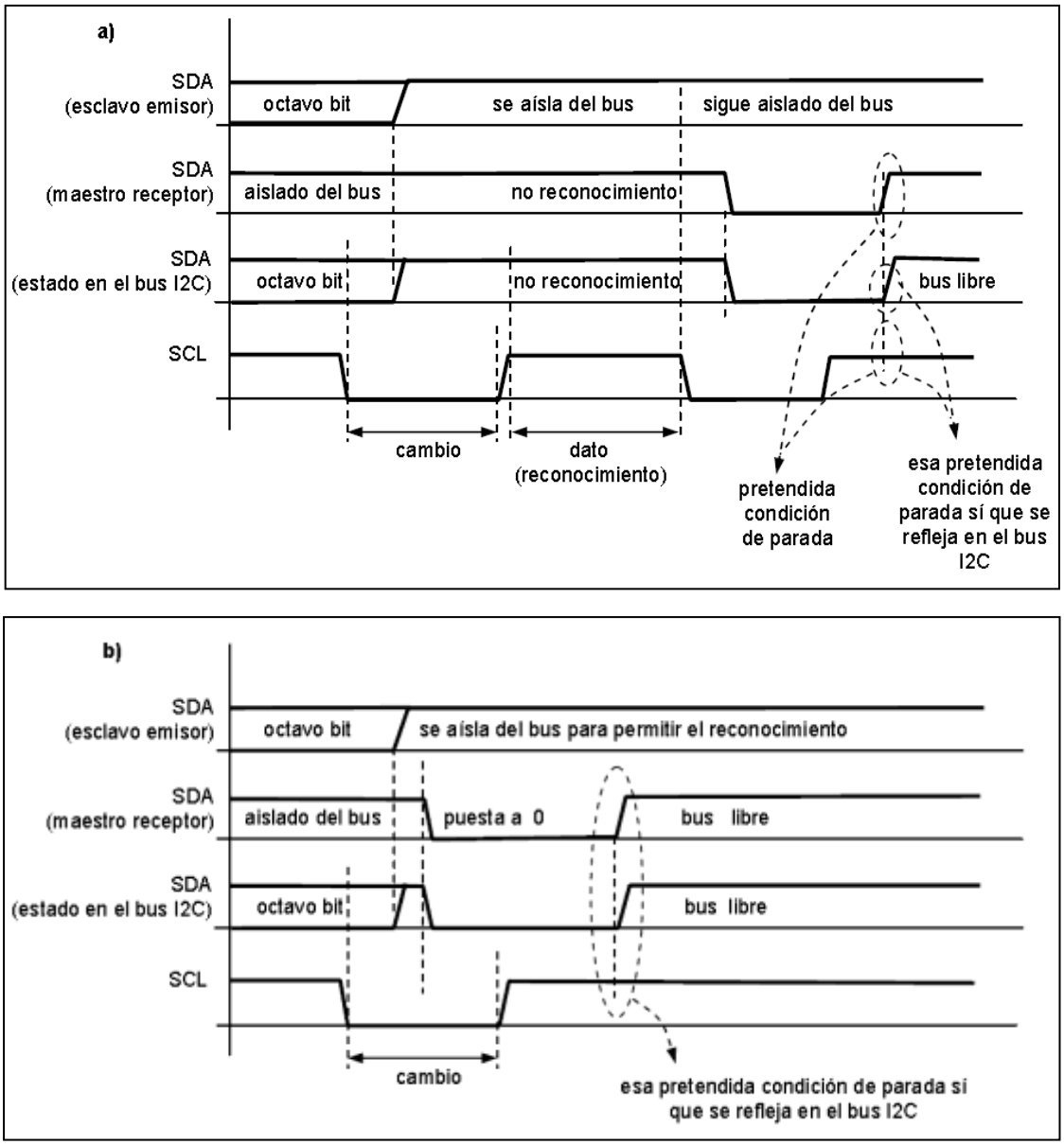


Figura 1.9 Terminación del proceso sin reconocimiento: a) modo normal
b) modo abreviado (puede no ser compatible)

1.4.4 Mecanismo de sincronización

Ya se ha comentado que el bus I2C establece un mecanismo para que un esclavo lento pueda forzar a un maestro rápido a ir más lento; a este mecanismo la norma I2C lo denomina sincronización. Se trata de un procedimiento ingenioso que saca provecho de la naturaleza a colector o drenador abierto de la señal de reloj SCL.

Por defecto, un maestro gobernará la señal de reloj del bus, SCL, a la velocidad que considere oportuno. Tal es así que no tiene por qué ser siempre la misma ni siquiera dentro de una misma transacción ni dentro de un mismo caracter. En este sentido tan sólo se han de satisfacer las condiciones mínimas de temporización entre los instantes de cambio de las señales, aspectos éstos que más adelante se detallarán al hablar de las especificaciones eléctricas y temporales de la interfaz I2C. Por lo dicho, el maestro marcha a su propia velocidad y supondrá que el esclavo con el que se comunique será capaz de seguir el ritmo. En muchas circunstancias esto es así, pero en otras puede resultar que no, bien porque el esclavo de ninguna manera pueda seguir jamás tal velocidad marcada por el maestro, o bien porque en ciertos instantes necesite realizar algunos procesos que le consuman un tiempo que no se pueda dedicar a la comunicación con el maestro. Sea cual sea la causa, el esclavo no estará en condiciones de comunicarse correctamente y se generará un error de comunicación.

Para dar mayor flexibilidad al bus, éste admite mecanismos para establecer una especie de diálogo entre las partes para amoldar dinámicamente la velocidad. El mecanismo consiste en lo siguiente: un maestro que admita la sincronización deberá comprobar el estado de la línea SCL del bus; si al activarla ésta no se activa efectivamente en el bus, entonces querrá significar que el esclavo le está solicitando ralentizar su velocidad. De esta manera, el maestro esperará con su SCL local activada y no considerará que ha comenzado el estado de dato hasta que efectivamente la línea SCL del bus esté a alta. Por la otra parte, todo esclavo que admita sincronización, estando la línea SCL del bus a baja podrá pisarla (poner a baja su SCL local) durante el tiempo que estime oportuno, y sólo liberará la línea SCL del bus en el instante en

que esté preparado para gestionar un nuevo bit o caracter. Todo esto puede observarse en la figura 1.10.

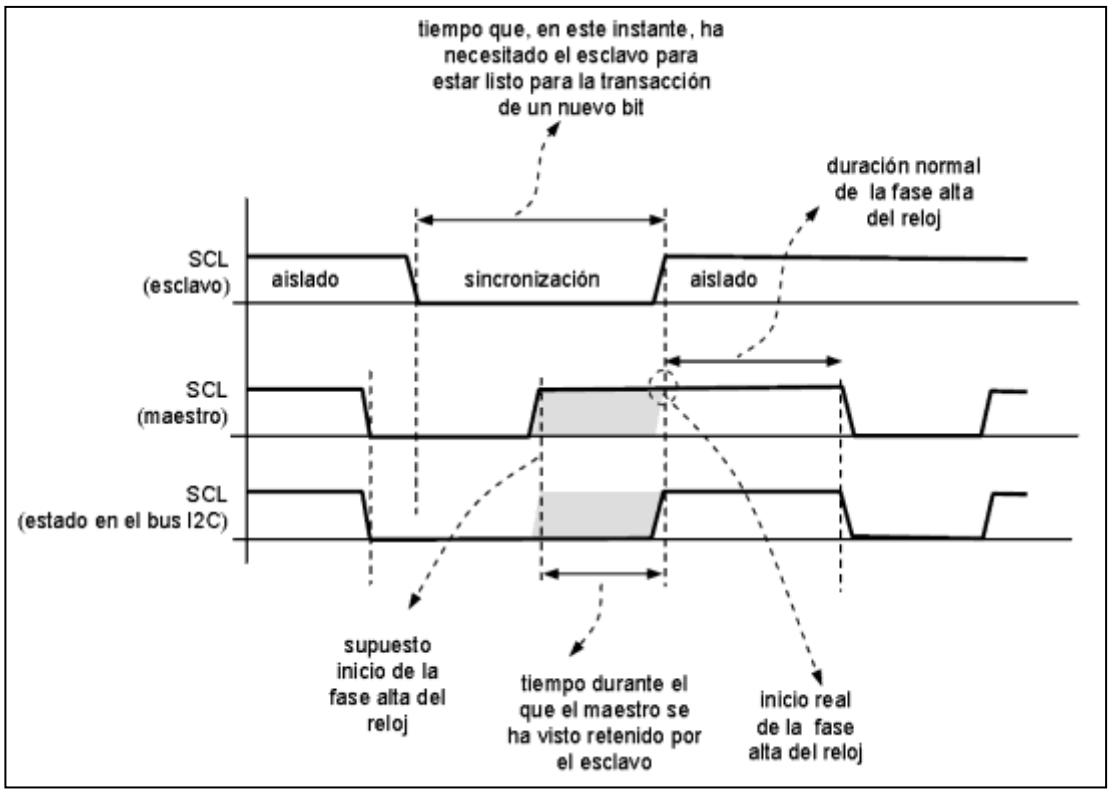


Figura 1.10 Mecanismo de sincronización en el bus I2C

Un esclavo receptor podrá utilizar el mecanismo de sincronización en términos bit a bit, caracter a caracter, o cuando y como lo estime oportuno en función de sus intereses.

1.4.5 Caracteres de dirección

Ya se ha mencionado que el primer caracter que se transmite tras una condición de inicio es la dirección del esclavo. La dirección ya se sabe que es un identificador de siete bits que representa un determinado circuito del bus I2C. El bit menos significativo del caracter de ocho bits, indica el sentido de la transferencia de los siguientes caracteres. También se ha dicho que no todas las combinaciones de siete

bits del primer caracter están disponibles como direcciones normales I2C. Y, por último, se ha apuntado que existen dos modos de direccionar: con direcciones de siete y con direcciones de diez bits. En el primer caso la dirección se transmite en un único caracter; y en el segundo caso con dos, debiendo ser el primero 11110- XX-L/E. Este es el momento de comentar el significado de aquellas combinaciones que no son direcciones normales. En la tabla que sigue se resume esta cuestión.

CARACTER		**FUNCIÓN**
DIRECCIÓN	**L/E**	
0000000	0	Llamada general (*General Call*) que va dirigida a todos los esclavos conectados al bus I2C
0000000	1	Aviso de inicio. Se usa para que los esclavos que implementan la interfaz por programa lento puedan detectar la condición de inicio
0000001	X	Dirección CBUS. En topologías en las que se mezclen circuitos I2C y circuitos CBUS, los I2C deben ignorar este direccionamiento
0000010	X	Reservado para formatos diferentes de bus. Los circuitos I2C que no los admitan deberán ignorar este direccionamiento
0000011	X	Reservado para futuras ampliaciones
00001XX	X	Petición de transacción a alta velocidad (*Hs*)
00010XX (...) 1110XX	X	Direcciones normales asignables a circuitos I2C
11110XX	X	Direccionamiento de diez bits
11111XX	X	Reservado para futuras ampliaciones

Tabla 1.1 Significado del primer caracter de una transacción

1.4.6 Llamado General

La dirección de llamada general se ha implementado para permitir que ciertos mensajes que un maestro necesite enviar a todos los elementos conectados al bus se haga de manera colectiva en lugar de individual direccionándolos uno tras otro. Esta cuestión podrá tener sentido en algunos casos pero no en otros. Aquellos circuitos que no tienen nada que hacer ante una llamada general simplemente la ignorarán.

Una llamada general consta de una trama formada por el caracter de la llamada general, el reconocimiento, caracter de tipo de llamada general, y su reconocimiento. Todos los dispositivos capaces de responder a una llamada general deben reconocer obligatoriamente, pero aquellos que por carecer de sentido no la contemplan deben ignorar toda la transacción y por tanto no reconocer nunca.

El segundo caracter indica qué tipo de llamada general se está realizando. De este octeto tan sólo están definidas algunas combinaciones:

- Valor 00000000: Está prohibido o reservado.

- Valor 00000010: Ha sido suprimido. La última versión de la norma I2C (1.1) ha suprimido este tipo de llamada general, que antes se utilizaba para indicar a los dispositivos que se les va a fijar por programa la parte programable de su dirección. El motivo de esta eliminación es que esta función es muy compleja y realmente casi nunca se llegó a utilizar. Ya se ha comentado que hay dispositivos en los que su dirección posee una parte fija y otra variable.

Normalmente la parte variable se fija por el diseñador aplicando a ciertas patillas los valores de los bits programables; pero también podían existir dispositivos en los que esta parte programable no se realizaba vía conexión eléctrica sino por medio de comandos de programación a través del propio bus I2C. Por tanto, para realizar esto es necesario hacer una llamada general de este tipo. Cuando se produce, los circuitos implicados entran automáticamente en el modo de programación por programa.

- Valor 00000100: Este tipo de llamada general 4 se utiliza para indicar a los circuitos del bus que recarguen la parte programable vía hardware, es decir, vía las patillas del encapsulado dedicadas a indicar la parte programable de su dirección. Esto resulta útil en aquellas topologías en las que un mismo circuito se utiliza en un número que excede el de las direcciones diferentes disponibles. Por ejemplo, supóngase que en un diseño se utilizan seis circuitos del mismo tipo que poseen una dirección del tipo 01000XX; es decir, la parte fija es 01000 y la programable XX cuyo valor se determina mediante dos patillas A1 y A0 de su encapsulado. Como puede verse, tan sólo se tiene la posibilidad de asignar a estos circuitos cuatro direcciones diferentes dentro de la misma topología (0100000, 0100001, 0100010 y 0100011) pero se necesitan seis. La solución a este tipo de problema es realizar un diseño con una especie de conmutación de bancos de circuitos, de modo que una dirección se reserva para los circuitos de los bancos inactivos, y el resto se asigna a los dispositivos del grupo activo. En el ejemplo que estamos utilizando se podría obrar de la siguiente manera: como sólo existen cuatro direcciones disponibles, una se reserva para no direccionar y el resto, tres, para asignarlas a los circuitos de

cada grupo. Por tanto, si en este caso cada grupo está formado por tres circuitos y existen seis del mismo tipo, entonces resulta que hay que formar dos grupos de tres. Si llamamos a cada circuito C0, C1, C2, C3, C4 y C5, y la dirección no utilizada es la 0100011, entonces el diseñador tendrá que poder aplicar a las patillas de dirección de cada circuito uno entre dos valores: el que indica no direccionable (11) y el asignado para poder direccionarlo (00, 01 o 10, según sea el caso). De este modo, el maestro cada vez que desee acceder a uno de los seis circuitos deberá determinar si se encuentra en el banco activo, y de no ser así aplicar a las patillas programables el valor adecuado y hacer una llamada general de tipo 4. La figura que sigue ilustra cómo se podría hacer este diseño desde el punto de vista físico.

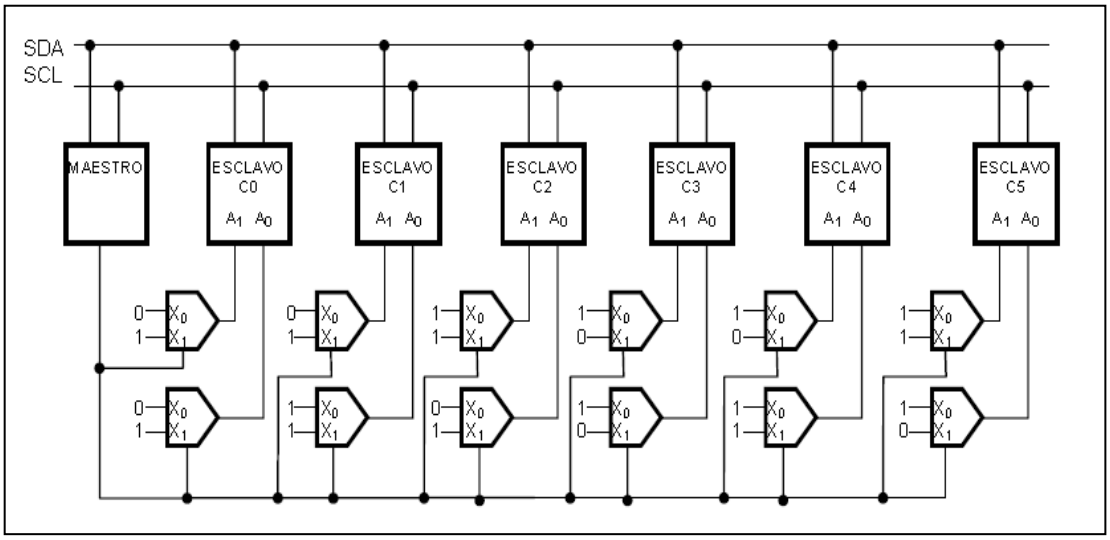


Figura 1.11 Ejemplo de topología con cambio dinámico de direcciones

El maestro dispone de una señal de control que se aplica a la entrada de selección de unos multiplexores. Si la señal vale 0, se seleccionan los canales cero de todos los multiplexores, con lo que a los tres primeros circuitos, C0, C1 y C2, se les asignan las partes programables de dirección 00, 01 y 10 respectivamente, y a los circuitos C3, C4

y C5 el valor 11; y sucede lo contrario si la señal de control vale 1. Esto implicará que si el maestro direcciona el dispositivo 0100010 se estará dirigiendo al C2 o al C5 dependiendo del valor de la señal de control. Por supuesto, cuando se conmuta de banco es necesario que los circuitos sean conscientes de que se les ha cambiado su parte de dirección variable, y para ello es por lo que lo que se les hace una llamada general del tipo 4. De esta manera los circuitos recargan la parte programable externa y, de ser el caso, modifican el valor que tomaron a la puesta en alimentación o en una llamada general previa.

- Valor 00000110: Una llamada general de tipo 6 tiene como efecto el reinicio de los circuitos a su configuración por defecto (la de la puesta en alimentación).

- Resto de valores XXXXXXX0: Todavía no han sido definidos los restantes valores que poseen su bit menos significativo a cero; cualquier circuito I2C actual debe ignorar estos tipos de llamadas generales.

- Valores XXXXXXX1: Una llamada general en la que el indicador de tipo tiene a 1 el bit menos significativo se denomina llamada general hardware. Este tipo de llamada se ha pensado para admitir que en una topología I2C pueda existir un maestro emisor peculiar que se caracterice por su capacidad de emitir al bus información sin dirigirla a ningún receptor en concreto. Es decir, que la información se hace pública, en general, y en lugar de indicar a quién se envía se indica quién la envía, para que un hipotético receptor interesado en ese origen pueda tomarla. Este tipo de circuitos y las topologías

resultantes son infrecuentes, pero resulta interesante que sean admisibles. Los bits XXXXXXX del segundo caracter de esta llamada general indican la dirección del maestro emisor hardware. A este caracter le seguirán una serie de caracteres adicionales (que habrá que reconocer) que son los datos emitidos al público por este tipo de circuito peculiar. Naturalmente, se ha de terminar con una condición de parada. Algunos de estos maestros emisores peculiares también pueden actuar como esclavos receptores. En estos casos, a la puesta en alimentación funcionan como esclavos receptores de manera que otro maestro pueda comunicarse con ellos para indicarles a qué dirección concreta deben dirigirse cuando sea el caso; para ello, ese otro maestro lo direccionará en modo escritura y le enviará un segundo caracter que será la dirección concreta a la que debe dirigirse en su momento. Si esto se ha hecho así, entonces estos circuitos peculiares (maestros hardware) cada vez que necesiten enviar algo al bus no lo harán mediante una llamada general de tipo impar sino de manera normal, direccionando a quien se les ha dicho en concreto.

1.4.7 Caracter de Inicio

El valor 00000001 (dirección 0, en modo lectura) es un caracter especial denominado de inicio que se utiliza al comenzarse una transacción en una topología I2C en la que se sabe que existen esclavos cuya interfaz I2C está implementada fundamentalmente por programa. Efectivamente, esto es muy frecuente cuando alguno de los dispositivos es un microcontrolador que no dispone de un controlador de bus I2C; en este caso se pueden utilizar líneas de E/S de propósito general para implantar las líneas del bus I2C y gestionar por programa todo el protocolo del bus. En estas circunstancias la carga

computacional de gestión del protocolo puede ser notable, con lo que disminuye el tiempo que se puede dedicar a otras tareas ajenas al control del bus I2C. Si se observa bien, un esclavo de este tipo ha de estar sondeando constantemente por programa el estado de las líneas SDA y SCL simplemente para poder detectar una condición de inicio y, en caso afirmativo, ver si se dirigen a él. Si el maestro que inicia una transacción lo hace a una velocidad elevada esto exige que el esclavo al que nos referimos tenga que estar monitorizando casi permanentemente el bus para que no se le pase la condición de inicio, cosa que empobrece el rendimiento de otras tareas que haya que realizar.

Pues bien, para permitir que este tipo de esclavos que implementan el protocolo I2C por programa puedan conseguir un rendimiento superior en otras tareas es por lo que la norma I2C contempla un modo especial de inicio de una transacción mediante el primer envío de un carácter de inicio. Si se observa, este carácter de inicio contiene siete ceros seguidos, que si se añaden al estado 0 en que quedó la línea SDA al producirse la condición de inicio, entonces se tiene que en la línea SDA se produce un estado al nivel 0 durante un tiempo entre siete y ocho veces superior al tiempo de bit de la transmisión. Por este motivo, un esclavo que monitorice por programa la condición de inicio podrá hacerlo de manera mucho más espaciada en el tiempo, y cuando detecte que en la línea SDA existe un 0 lógico entonces conmutará a un modo de gestión mucho más rápida para poder estar pendiente con detalle de la transacción que se acaba de iniciar. Por otra parte, el maestro que ha iniciado la transacción con el carácter de inicio deberá generar tras él el impulso SCL para el reconocimiento, pero no considerará como erróneo el que no se reciba; es más, ningún esclavo debe

reconocer un carácter de inicio. A continuación generará una condición de reinicio y comenzará la transacción con el formato normal que ya se conoce.

1.5 Arbitraje en las topologías Multimaestro

Como se sabe, el bus I2C admite configuraciones en las que pueda existir más de un dispositivo capaz de actuar como maestro. Cualquier bus de comunicación que contemple tal circunstancia ha de proveer algún tipo de mecanismo de arbitraje que solucione el problema de quién gana el gobierno del bus si más de un sistema intenta simultáneamente iniciar una transacción. Los buses más evolucionados definen niveles jerárquicos de manera que el dispositivo de mayor nivel tiene prioridad sobre los de menor; de esta manera es posible hacer configuraciones coherentes con el grado de relevancia de los dispositivos conectados al bus. Pero para conseguir esto es necesario el concurso de señales en número suficiente para determinar el nivel de jerarquía. Dado que el bus I2C tan sólo utiliza dos señales, la de datos y la de reloj, no existe la posibilidad de establecer un mecanismo coherente de jerarquía que pueda utilizarse a la hora del arbitraje. No obstante, esto no quiere decir que no se pueda definir tal método de arbitraje. En concreto, se recurre a una técnica ingeniosa que se apoya, una vez más, en el hecho de que las líneas del bus I2C son a colector o drenador abierto. El mecanismo de arbitraje consiste en los siguientes principios:

- Todo sistema maestro que pueda formar parte de una topología multimaestro ha de admitir la sincronización.

- Todo maestro que pretenda gobernar el bus podrá iniciar una transacción si parte de una condición de bus libre.

- Todo maestro que haya iniciado una transacción y compruebe que el nivel aplicado por él a la línea SDA no se refleja efectivamente en tal línea deberá entonces considerar que algún otro maestro contiende con él por el gobierno del bus, habiendo él perdido y el otro ganado.

- Todo maestro que compruebe que ha perdido el arbitraje, debe inmediatamente aislarse del bus y, de ser el caso, actuar como esclavo receptor por si el ganador del arbitraje se dirigiese a él.

Como puede verse, se saca provecho del hecho de que en las líneas SDA y SCL se tiene el producto lógico de las señales locales SDA y SCL de los excitadores de todos los nodos del bus. Este mecanismo de arbitraje es un tanto peculiar pero efectivo, aunque el resultado del arbitraje (quién gana) se deja a cuestiones puramente azarosas. Efectivamente, gana el arbitraje el primero que genera un bit de dato a 0 mientras todos los restantes generan un 1.

Aunque múltiples maestros pueden estar transaccionando a velocidades a priori diferentes, no existe posibilidad de que la emisión de datos se corrompa durante el arbitraje puesto que los maestros han de admitir el mecanismo de sincronización por el que los más rápidos adaptan sus relojes a la velocidad del más lento.

En la figura 1.12 puede verse un ejemplo en el que dos maestros pugnan por gobernar el bus. Como puede observarse, además de la sincronización en términos de adaptación del más rápido al más lento, el arbitraje se pierde en el momento en que un maestro intenta poner un 1 lógico en la línea SDA y se encuentra con que ésta se mantiene a baja. Si se medita lo que implica este mecanismo de arbitraje puede llegarse acertadamente a la conclusión de que múltiples maestro pueden gobernar simultáneamente el bus, y hasta el final, siempre que se dirijan al mismo esclavo y pretendan realizar absolutamente lo mismo con él. Por ejemplo, si tres maestros pretenden leer los contenidos de las mismas posiciones de memoria de una EEPROM, y coinciden al iniciar la transacción, se tendrá que los tres finalizarán con éxito el proceso de lectura, aunque se emplee para ello el reloj del sistema más lento.

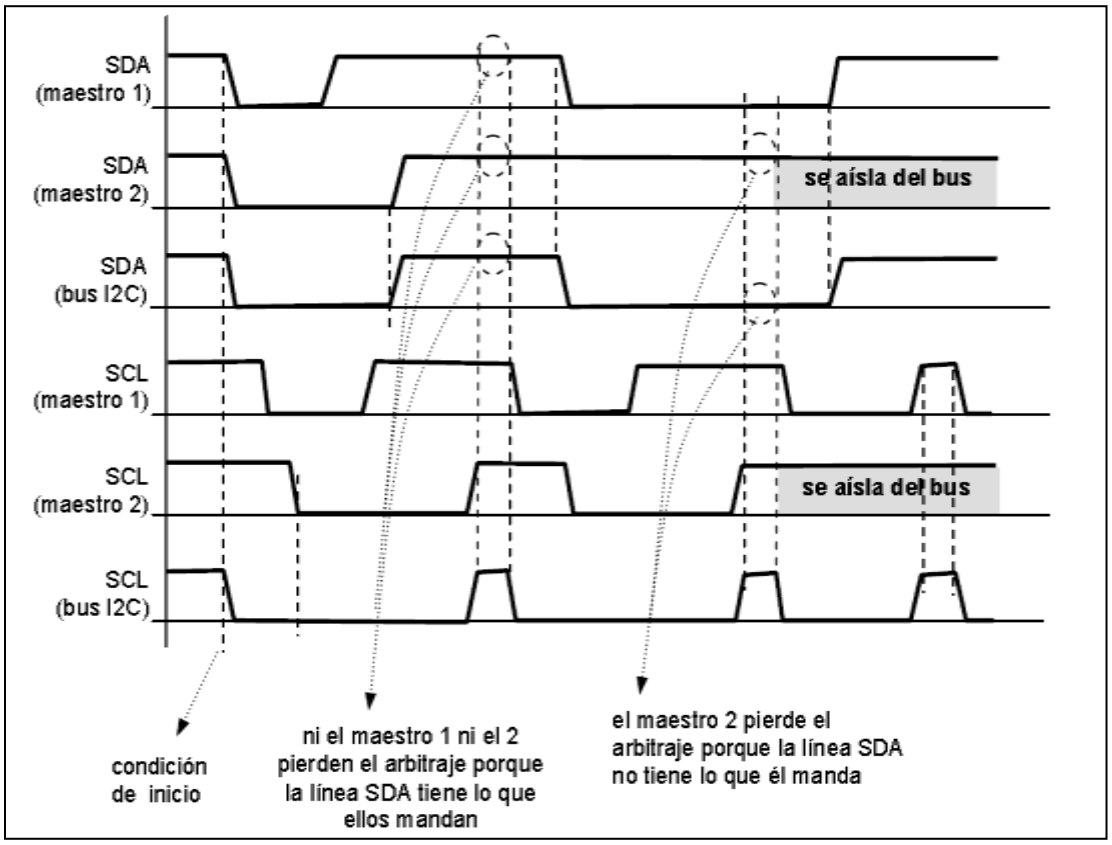


Figura 1.12 Ejemplo de arbitraje en el bus I2C

En el ejemplo de la figura 1.12 puede verse cómo los maestros han de verificar la coincidencia de su dato con el efectivamente presente en el bus, y esto se hace no cuando cada maestro activa su SCL sino cuando la señal SCL del bus está activa, puesto que no debe olvidarse que los maestros han de admitir el mecanismo de sincronización. Obsérvese que el mecanismo de sincronización da como resultado una señal de reloj que tiene la frecuencia del más lento, y el tiempo en alto del que presente más corto tiempo. En el ejemplo, el más lento es el maestro 2, y el tiempo en alto más corto es el del 1, aunque sus dos primeros impulsos tienen una fase activa notablemente más ancha debido al mecanismo de sincronización forzado por el maestro 2, hasta que éste pierde el arbitraje y se aísla entonces del bus.

1.6 Modo de Alta Velocidad

La norma I2C contempla tres modos de funcionamiento: normal, rápido y de alta velocidad. El primero admite tasas de hasta 100 kbps, el segundo –introducido en la revisión 1.0 de 1992– de hasta 400 kbps, y el tercero –introducido en la revisión 1.0 de 1998– de hasta 3’4 Mbps.

Existe una compatibilidad descendente, de tal manera que un dispositivo capaz de funcionar en modo de alta velocidad también lo hará en modo rápido, y uno en modo rápido también lo hará en modo normal. Actualmente, todo nuevo dispositivo I2C ha de ser capaz de funcionar al menos en modo rápido; no obstante, los circuitos capaces del modo de alta velocidad resultan una excelente solución para aquellas aplicaciones de altas prestaciones en las que se demande una elevada tasa de transferencia. Este tipo de circuitos necesita que la electrónica de excitación en los maestros posea unas

características especiales con el fin de acelerar las transiciones de baja a alta en la línea de reloj, puesto que de actuar tan sólo la resistencia de elevación sería imposible conseguir las elevadas tasas transferencia de hasta 3’4 megabits. No se olvide que la resistencia de elevación actuará en las transiciones 0-1 como vía de carga de la capacidad concentrada en el bus (hasta 400 pF según la norma), y por ello en la línea SCL se tendrá la típica forma exponencial de carga de un condensador, en el que la constante de tiempo t=RxC, limitará la máxima velocidad alcanzable.

Los aspectos topológicos así como de tipo eléctrico y temporal se comentarán más adelante, y nos centraremos ahora en comentar los aspectos funcionales y de protocolo que caracterizan al modo de alta velocidad. Por todo lo señalado, los circuitos I2C de alta velocidad denominan a sus señales SDAH y SCLH en lugar de SDA y SCL; no obstante, pueden existir circuitos que ofrezcan ambos tipos de señales, en cuyo caso las normales pueden utilizarse para otros fines si no se utilizan. Esta dualidad de señales resulta útil en topologías con velocidades mixtas.

Por lo que respecta a las características de protocolo, en primer lugar hay que decir que este modo no admite ni el arbitraje ni la sincronización bit a bit puesto que tal cuestión iría en contra del objetivo perseguido de conseguir la máxima velocidad posible de transferencia. Esto es, todo dispositivo maestro capaz de manejar el modo de alta velocidad deberá realizar transacciones sólo con circuitos también capaces de utilizar el modo de alta velocidad. Por supuesto, si un maestro es capaz de emplear la alta velocidad pero funciona en un modo inferior (rápido o normal) entonces podrá dirigirse a cualquier circuito capaz de manejar tales modos inferiores. El arbitraje se realiza en un modo que no es de alta velocidad, como se analizará más adelante.

En este momento puede resultar interesante comentar un aspecto de las topologías mixtas en las que existen circuitos de alta velocidad (Hs de aquí en adelante) junto a otros rápidos o normales. Para evitar cargar excesivamente el bus (a mayor número de circuitos mayor capacidad equivalente existirá en él) los circuitos normales y rápidos se pueden separar de los de alta velocidad mediante sendos puentes en SDA y SCL, que no son más que circuitos especiales diseñados para tal caso.

1.6.1 Formato de una transferencia en Modo de Alta Velocidad

Todo circuito Hs tiene la obligación de entrar en modo rápido a la puesta en alimentación. Un maestro Hs que quiera iniciar una transacción podrá hacerlo tal y como ya se sabe si va a emplear una velocidad rápida o normal, pero si desea emplear la alta velocidad entonces antes deberá realizar una petición de alta velocidad, que consiste en lo que sigue:

1. Generar una condición de inicio.

2. Emitir su código de maestro Hs, que es 00001XXX. Esto implica que no se admiten topologías con más de ocho maestros Hs. Realmente deberían ser siete, puesto que la norma recomienda reservar el código 00001000 para que pueda ser utilizado por equipos de análisis y diagnóstico de buses I2C.

3. Al código de maestro Hs nadie debe darle su reconocimiento.

En el transcurso de estas tres fases, que se realizan bien en modo normal o bien en modo rápido, se admite tanto el arbitraje como la sincronización. Por otra parte el código 00001XXX sirve de indicador, de que a continuación se va a comenzar una transacción a alta velocidad. El código concreto de un maestro Hs (desde 00001000 hasta 00001111) es de libre elección por parte del diseñador de un sistema, y ningún circuito que pueda comportarse como tal lo lleva pre-asignado por el comité I2C. El maestro Hs que, tras el no-reconocimiento, logre el gobierno del bus pondrá a alta su señal SCLH y a partir de este mismo instante tH se conmutará al modo Hs. En este momento el maestro Hs generará una condición de reinicio a la que le seguirá la dirección (7 o 10 bits) del esclavo con el que se desea realizar la transacción, con una mecánica de funcionamiento lógico enteramente similar a la ya conocida.

Dado que en el modo Hs se admite tan sólo la sincronización carácter a carácter, para hacerla posible el maestro ha de desactivar su excitador SCLH (es una fuente de corriente) en los instantes tras cada condición de reinicio y tras cada tiempo de reconocimiento (se dé o no se dé). De esta manera sólo queda operativa la resistencia de elevación y así es posible que un posible esclavo receptor pise efectivamente la señal SCLH sin temor a que se destruya su propio excitador SCLH local. El maestro Hs reactivará su excitador especial SCLH en el momento en que detecte que nadie le está pisando su SCLH local. Todo lo comentado puede observarse en la figura siguiente.

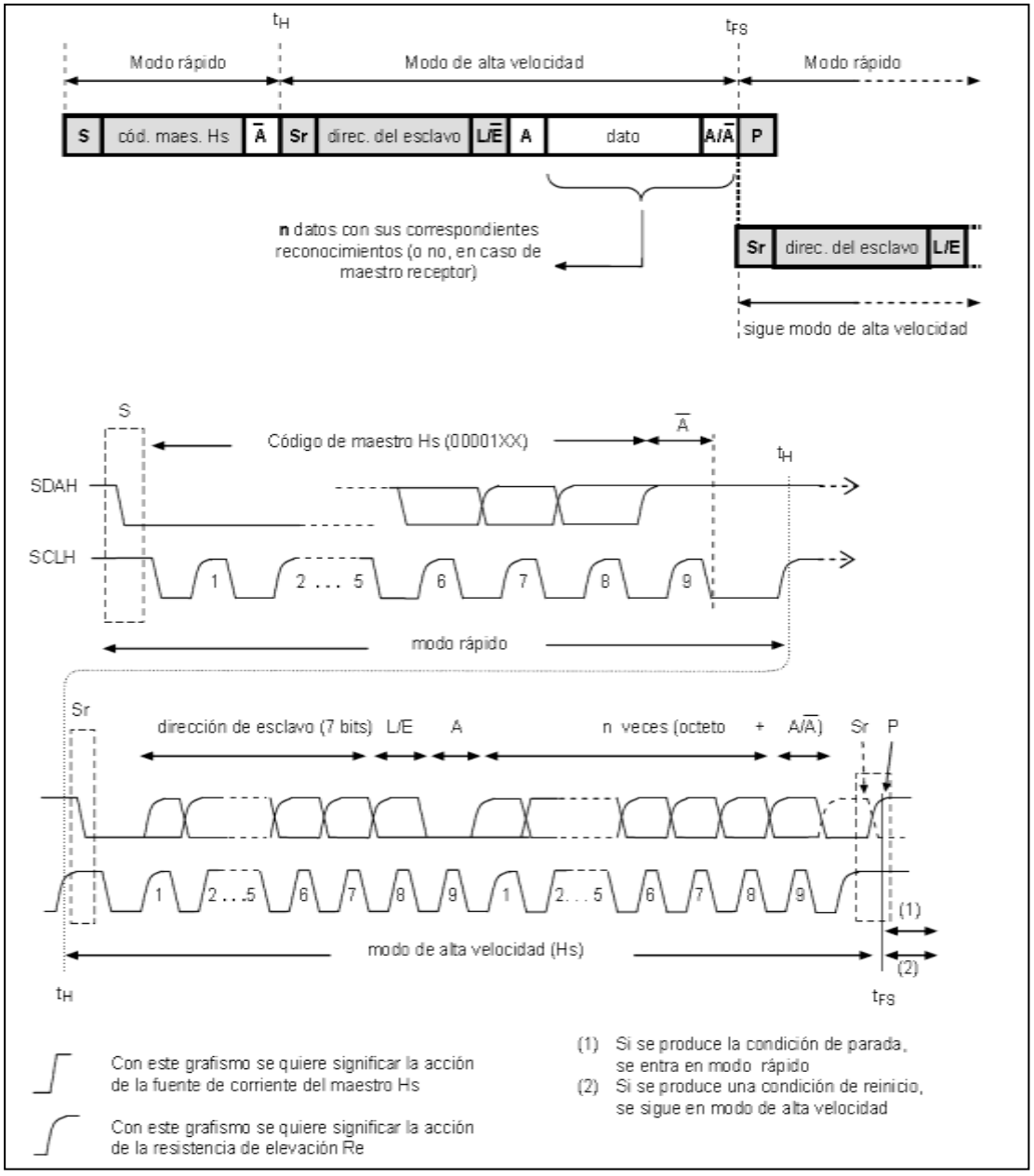


Figura 1.13 Formatos de una transacción en modo de alta velocidad (Hs)

El cronograma que se muestra en la figura 1.13 se ha dividido en dos partes; no obstante, el lector debe considerarlo como uno solo en el que las líneas del extremo derecho en la primera mitad (la superior) continúan por el extremo izquierdo de la segunda mitad (la inferior); el instante de nexo entre las dos mitades es el indicado tH, que es el instante en el que el maestro entra en el modo de alta velocidad.

1.7 Especificaciones Eléctricas y Temporales

En la figura 1.14 puede observarse el aspecto de la electrónica de excitación y de recepción de un dispositivo I2C de tecnología MOS, válida para los modos normal y rápido. Los transistores de excitación de las líneas SDA y SCL del bus I2C son a drenador abierto, y por ello las líneas SDA y SCL del bus deben tener conectadas a VDD sendas resistencias de elevación que marquen el estado lógico 1 cuando el respectivo transistor no conduzca. También puede verse cómo el estado de ambas líneas del bus puede ser sondeado mediante el correspondiente búfer de entrada (disparador de Schmitt). Esto, que puede parecer lógico para la línea de datos serie, SDA, si se desea poder actuar tanto como emisor como receptor, también es necesario para la señal de reloj SCL si se desea tener la capacidad de sincronización.

Además se puede apreciar, que el nivel que se desea aplicar a una línea del bus se aplica invertido a la puerta del transistor, de manera que si el dato es un 0, al cambiar a 1 permite que el transistor conduzca; como la resistencia drenador-surtidor es muy baja en estas condiciones, entonces el nivel que se tiene a la salida es prácticamente el del común de señal, es decir, un cero lógico. Y si el dato es un 1 lógico, al invertirse a 0 la tensión asociada hace que el transistor no conduzca, ofreciendo una elevadísima resistencia drenador-surtidor. Esta enorme impedancia de salida trae como efecto el práctico aislamiento eléctrico del excitador con respecto a la línea del bus. En este caso, el nivel existente en el bus es un 1 lógico al actuar la resistencia externa como una de elevación

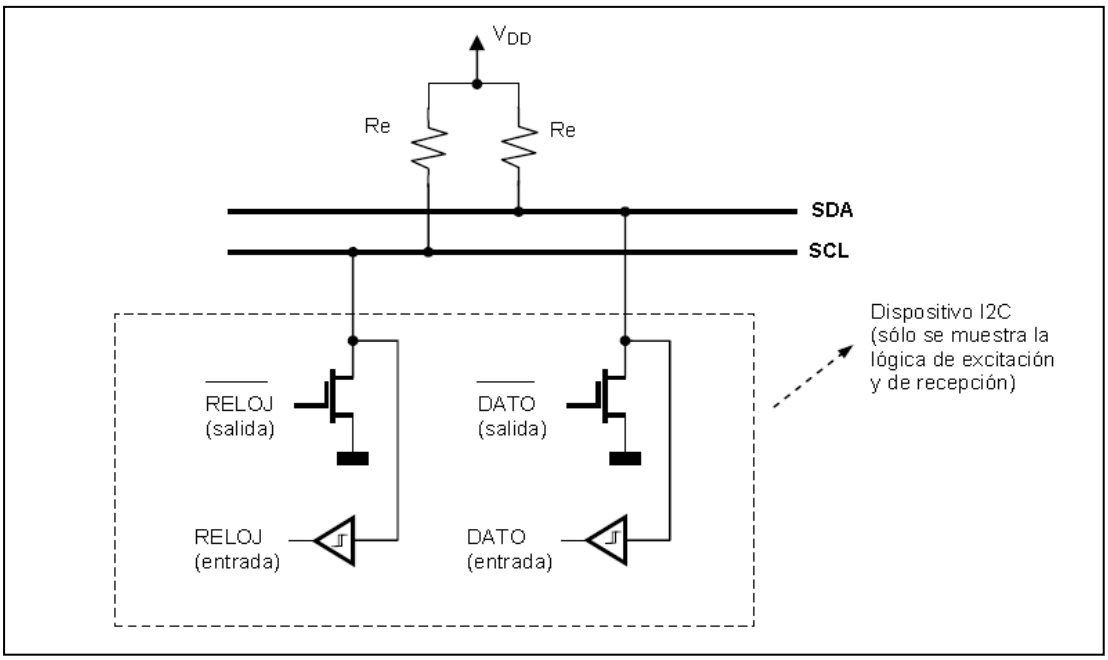


Figura 1.14 Etapa de excitación y recepción en un circuito I2C normal o rápido

Cuando el bus está libre los excitadores están inactivos, y por tanto son las resistencias de elevación las que marcan los niveles a alta en las líneas. Para los circuitos en modo rápido las resistencias de elevación pueden ser tales si las cargas conectadas al bus no exceden los 200 pF, pero si están entre este valor y los 400 pF entonces deben emplearse fuentes de corriente (3 mA máximo) o una red resistiva conmutada. En el modo de alta velocidad el umbral está en los 100 pF Por otra parte, entre cada línea del bus y su terminal de cada circuito I2C se pueden conectar una resistencia en serie si se desea proteger a los circuitos ante posibles picos de tensión que puedan producirse en las líneas. El valor de esta resistencia serie RS no debe exceder los 100 ohmios. En los circuitos que admiten la alta velocidad estas resistencias en serie además sirven para minimizar fenómenos como las oscilaciones espurias, reflexiones de señal y demás interferencias.

La electrónica de excitación incluye una circuitería cuya misión es controlar la velocidad de cambio en las líneas SDA y SCL, manteniéndola dentro de unos límites

concretos. Esta limitación de la tasa de cambio (slew rate en inglés) persigue minimizar la generación de interferencias electromagnéticas hacia los circuitos en las proximidades del bus; no se olvide que los cambios rápidos en una señal digital contienen un espectro de altas frecuencias capaces de inducir IEM. Cuando se comentó el modo de alta velocidad se dijo que estos circuitos incluyen un excitador especial –fuente de corriente – que tiene como misión conseguir aumentar la velocidad de cambio de la línea SCLH. En este modo las resistencias de elevación llegan a constituir un problema debido a la constante de tiempo asociada, que es el producto de la capacidad acumulada en una línea por el valor de la resistencia de elevación. De hecho el problema se produce esencialmente en los cambios de baja a alta; y esto es debido a que, como ya se ha dicho, cuando un excitador genera un 1 lógico realmente no lo hace puesto que no conduce y ofrece una elevada impedancia de salida.

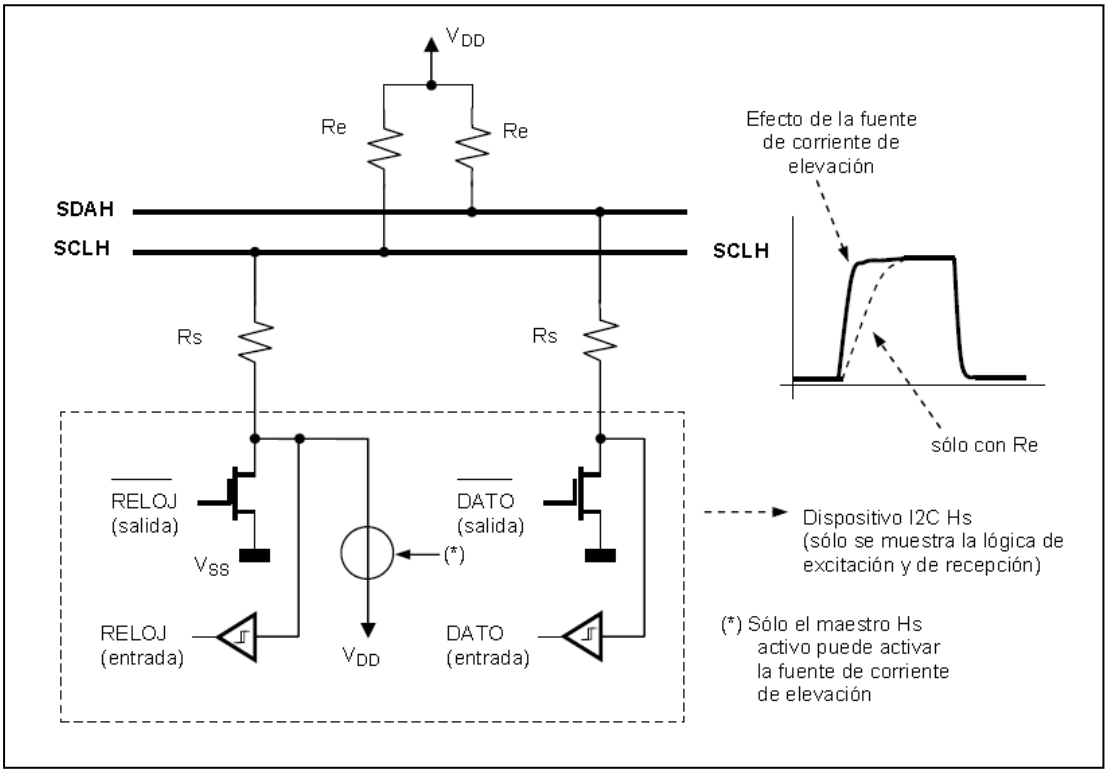


Figura 1.15 Etapa de excitación y recepción en un circuito maestro I2C de Alta Velocidad

Por tanto el nivel en la línea queda determinado por la forma en que se carga una capacidad. En los cambios de alta a baja se produce una rápida descarga de esta capacidad concentrada equivalente, a través del transistor del excitador, que en este caso posee una resistencia colector-emisor o drenador-surtidor muy reducida. Así, en los excitadores especiales SCLH se tiene una fuente de corriente que en las transiciones de baja a alta se encarga de inyectar una elevada corriente a la línea SCLH forzando una rápida carga de la capacidad equivalente en la línea. Esto se puede observar en la figura 1.15.

1.8 Topologías con Velocidades Mixtas

Como ya ha sido mencionado, es posible trabajar con topologías mixtas en las que existan circuitos normales, rápidos y de alta velocidad. En este caso es posible conseguir distintas velocidades de transferencia haciendo uso de un puente de interconexión entre la sección del bus al que se conectan los circuitos Hs y la que tiene los circuitos rápidos y normales. Existen circuitos maestros Hs que ya integran este puente; en la norma I2C versión 1.1 pueden obtenerse los detalles de la filosofía operativa de estos puentes. Aquí tan sólo se indicará en la tabla 1.2 las posibles velocidades que se pueden conseguir. Los maestros Hs en topologías mixtas utilizarán las señales SDAH, SCLH y SDA, SCL, manejadas por el puente, cada una de las cuales se conectarán a las secciones oportunas del bus I2C. Las líneas SDAH y SCLH siempre se utilizarán, independientemente del modo empleado de la transferencia, mientras que las SDA y SCL tan sólo serán utilizadas en los modos normal y rápido. Si un usuario utiliza un maestro Hs esta cuestión es totalmente transparente puesto que es gestionada por la capa física de la interfaz que integre tal dispositivo.

TRANSFERENCIA ENTRE. . .	CONFIGURACIÓN DEL BUS I2C			
	Hs + RÁPIDO +NORMAL	Hs + RÁPIDO	Hs+ NORMAL	RÁPIDO+ NORMAL
Hs ↔ Hs	0 ~ 3’4 Mbits/s	0 ~ 3’4 Mbits/s	0 ~ 3’4 Mbits/s	--
Hs ↔ Rápido	0 ~ 100 kbits/s	0 ~ 400 kbits/s	--	--
Hs ↔ Normal	0 ~ 100 kbits/s	--	0 ~ 100 kbits/s	--
Rápido ↔ Rápido	0 ~ 100 kbits/s	--	--	0 ~ 100 kbits/s
Rápido ↔ Normal	0 ~ 100 kbits/s	0 ~ 100 kbits/s	--	0 ~ 100 kbits/s
Normal ↔ Normal	0 ~ 100 kbits/s	--	0 ~ 100 kbits/s	0 ~ 100 kbits/s

Tabla 1.2 Velocidades que se obtienen en topologías mixtas

Los maestros Hs en topologías mixtas utilizarán las señales SDAH, SCLH y SDA, SCL, manejadas por el puente, cada una de las cuales se conectarán a las secciones oportunas del bus I2C. Las líneas SDAH y SCLH siempre se utilizarán, independientemente del modo empleado de la transferencia, mientras que las SDA y SCL tan sólo serán utilizadas en los modos normal y rápido. Si un usuario utiliza un maestro Hs esta cuestión es totalmente transparente puesto que es gestionada por la capa física de la interfaz que integre tal dispositivo.

1.9 Niveles de Tensión y de Corriente

Dado que la norma admite la conexión al bus de circuitos de muy diferentes tecnologías, se especifican de varias maneras los niveles de tensión de las señales. A continuación se indica un resumen de ello:

- Para aquellos circuitos que se alimentan a 5 voltios (sean TTL, CMOS, etcétera), se tiene lo siguiente:
  - Tensión mínima de entrada a alta: VIHmin=3 V
  - Tensión máxima de entrada a baja: VILmax=1’5 V

- Para aquellos circuitos que admiten amplios márgenes de alimentación se tiene que:
  - Tensión mínima de entrada a alta: VIHmin=70% de VDD
  - Tensión máxima de entrada a baja: VILmax=30% de VDD
- Por otra parte, ha de cumplirse que, independientemente de la tensión de alimentación VDD, la tensión de salida a baja VOL no ha de superar los 0’4 voltios para una corriente de salida de 3 mA.
- La corriente de entrada en SDA o SCL de un dispositivo I2C, para un nivel alto, no excederá los -10µA.

PARÁMETRO	SÍMBOLO	MODO NORMAL		MODO RÁPIDO		UNIDAD
		mínimo	máximo	mínimo	máximo	
Tensión de entrada para nivel BAJO: Niveles de entrada fijos Niveles relativos a VDD	VIL	-0’5 -0’5	1’5 30% VDD	no aplicable -0’5	no aplicable 30% VDD (1)	V
Tensión de entrada para nivel ALTO: Niveles de entrada fijos Niveles relativos a VDD	VIH	3’0 70% VDD	VDDmax+0’5 VDDmax+0’5	n. a. 70% VDD (1)	n. a. VDDmax+0’5	V
Histéresis de los disparadores de Schmitt en las entradas: VDD > 2 V VDD < 2 V	Vhys	n. a. n. a.	n. a. n. a.	5% VDD 10% VDD	--	V
Tensión de salida a nivel bajo (colector o drenador abierto) sumiendo 3 mA: VDD > 2 V VDD < 2 V	VOL1 VOL3	0 n. a.	0’4 n. a.	0 n. a.	0’4 20% VDD	V
Tiempo de bajada, desde VIHmin hasta VILmax, con una capacidad en el bus desde 10 pF hasta 400 pF	tof	--	250(3)	20+0’1Cb (2)	250(3)	ns
Anchura de los picos espurios que deben ser suprimidos por el filtro de las entradas	tSP	n. a.	n. a.	0	50	ns
Corriente de entrada en cada patilla de E/S con una tensión de entrada entre el 10 y el 90% de VDDmax	Ii	-10	10	-10(4)	10(4)	µA
Capacidad de entrada de patilla de E/S	Ci	--	10	--	10	pF

Tabla 1.3 Características de las etapas de E/S de SDA y SCL en un dispositivo I2C normal o rápido

(1) Aquellos dispositivos con una alimentación no estándar, que no se ajustan a los niveles pensados para un sistema I2C, deben relacionar sus niveles de entrada a la tensión VDD a la que se conectan las resistencias de elevación Re.

(2) Cb = capacidad de una línea del bus, en picofaradios (es la capacidad concentrada del cable y de las patillas de E/S)

(3) Este valor máximo de tof para las salidas de los excitadores es menor que el indicado tf en la tabla 5.5 para las líneas SDA ySCL (300 ns). El motivo es permitir la conexión como protección de resistencias en serie, Rs, entre la salida de cada excitador y las líneas SDA y SCL, sin exceder tal valor tf.

(4) En los dispositivos que admitan el modo rápido las patillas de E/S no deben bloquear las líneas SDA y SCL si VDD se apaga o suprime.

En la tabla 1.3, se indican los principales parámetros eléctricos en los modos normal, rápido y de alta velocidad, en lo que atañe a las etapas excitadoras y receptoras en un circuito I2C.

PARÁMETRO	SÍMBOLO	MODO Hs		UNIDAD
		mínimo	máximo	
Tensión de entrada para nivel BAJO:	VIL	-0’5	30% VDD (1)	V
Tensión de entrada para nivel ALTO:	VIH	70% VDD (1)	VDD+0’5(2)	V
Histéresis de los disparadores de Schmitt en las entradas:	Vhys	10% VDD	--	V
Tensión de salida a nivel BAJO (colector o drenador abierto) sumiendo 3 mA: VDD > 2 V VDD < 2 V	VOL	0 0	0’4 20% VDD	V
Resistencia, en estado activo, de la puerta de transferencia, para ambos sentidos de la corriente a nivel VOL entre SDA y SDAH o SCL y SCLH, a 3 mA	RonL	--	50	Ω
Resistencia, en estado activo, de la puerta de transferencia, entre SDA y SDAH (o SCL y SCLH) si ambos están al nivel VDD	RonH (2)	50	--	KΩ
Corriente de elevación, de la fuente de corriente de SCLH. Es válido para niveles entre el 30 y el 70% de VDD	ICS	3	12	mA
Tiempo de subida, en la salida (con la fuente de corriente activa), y Tiempo de bajada, para SCLH con una capacidad de carga de entre 10 y 100 pF	trCL tfCL	10	40	ns
Tiempo de subida, en la salida (con la fuente de corriente activa), y Tiempo de bajada, para SCLH con una fuente de corriente externa de 3 mA y capacidad de carga de hasta 400 pF	trCL (3) tfCL (3)	20	80	ns
Tiempo de bajada, en la salida SDAH, con una capacidad de la carga en el bus de 10 pF hasta 100 pF	tfDA	10	80	ns
Tiempo de bajada, en la salida SDAH, con una capacidad de la carga en el bus de 400 pF	tfDA (3)	20	160	ns
Anchura de los picos espurios en SDAH y SCLH que deben ser suprimidos por el filtro de las entradas	tSP	0	10	ns
Corriente de entrada en cada patilla de E/S con una tensión de entrada entre el 10 y el 90% de VDD	Ii	--	10	μA
Capacidad de entrada de patilla de E/S	Ci	--	10	pF

Tabla 1.4 Características de las etapas de E/S de SDA(H) y SCL(H) en un dispositivo I2C de alta velocidad

(1) Aquellos dispositivos con una alimentación no estándar, que no se ajustan a los niveles pensados para un sistema I2C, deben relacionar sus niveles de entrada a la tensión VDD a la que se conectan las resistencias de elevación Re.

(2) Los dispositivos que ofrezcan la función de ajuste de niveles deben admitir una tensión máxima de entrada de 5’5 V en SDA y SCL.

(3) El valor de los tiempos de subida y de bajada debe interpolarse linealmente para capacidades de carga entre los 100 y 400 pF.

(4) Las etapas SDAH y SCLH de un dispositivo esclavo Hs deben poseer salidas flotantes si su alimentación se ha suprimido o apagado. Este requisito no es obligatorio para las etapas de E/S SDAH y SCLH en un dispositivo maestro Hs; ello es debido al circuito fuente de corriente de salida, que normalmente posee un diodo rectificador conectado a VDD. Esto significa que la fuente de alimentación de un dispositivo maestro en modo de alta velocidad, Hs, no puede apagarse sin afectar a las líneas SDAH y SCLH.

1.10 Especificación de la temporización

A continuación se indicarán los parámetros temporales definidos por la norma I2C. En la tabla 1.5 para los circuitos en modo normal o rápido, y en la tabla 1.6 para los de alta velocidad.

PARÁMETRO	SÍMBOLO	MODO NORMAL		MODO RÁPIDO		UNIDAD
		mínimo	máximo	mínimo	máximo	
Frecuencia del reloj SCL	fSCL	0	100	0	400	khz
Tiempo de retención en la condición de(RE-) INICIO. Tras este instante se genera el primer impulso de reloj	tHD; STA	4’0	--	0’6	--	µs
Periodo a BAJA de la señal de reloj SCL	tLOW	4’7	--	1’3	--	µs
Periodo a ALTA de la señal de reloj SCL	tHIGH	4’0	--	0’6	--	µs
Tiempo de asentamiento para una condición de reinicio	tSU;STA	4’7	--	0’6	--	µs
Tiempo de retención de un dato: - Para maestros compatibles CBUS - Para dispositivos I2C	tHD;DAT	5’0 0(2)	3’45(3	---- 0(2	--- 0’9(3)	µs
Tiempo de asentamiento de un dato	tSU;DAT	250	--	100(4)	--	ns
Tiempo de subida de SDA y de SCL	tr	--	1000	20+0’1Cb (5)	300	ns
Tiempo de bajada de SDA y de SCL	tf	--	300	20+0’1Cb (5)	300	ns
Tiempo de asentamiento para una condición de PARADA	tSU;STO	4’0	--	0’6	--	µs
Tiempo de bus libre entre una condición de PARADA y una de INICIO	tBUF	4’7	--	1’3	--	µs
Capacidad de la carga en cada la línea	Cb		400		400	pF
Margen de ruido a nivel BAJO para cada dispositivo conectado el bus (incluyendo histéresis)	VnL	10% VDD	--	10% VDD	--	V
Margen de ruido a nivel ALTO para cada dispositivo conectado el bus (incluyendo histéresis)	VnH	20% VDD	--	20% VDD	--	V

Tabla 1.5 Parámetros de temporización en los modos normal y rápido [(1)]

(1) Todos los valores están referidos a los niveles VIHmin y VILmax. (Véase la tabla 1.3)

(2) Un dispositivo debe procurar un tiempo de retención de al menos 300 ns para la señal SDA (con respecto a la señal SCL en su valor VIHmin) para así cubrir la región no definida en el transcurso del cambio a baja de SCL.

(3) El valor máximo de THD;DAT sólo debe satisfacerse si el dispositivo no alarga la fase a BAJA (tLOW) de la señal SCL.

(4) Un dispositivo en modo rápido I2C puede utilizarse en un sistema I2C en modo normal, pero ha de cumplirse el requisito de que tSU;DAT≥250 ns. Automáticamente este será el caso si el dispositivo no alarga la fase a BAJA de la señal SCL. Si el dispositivo sí que lo hace entonces deberá lanzar el siguiente bit por la línea SDA un tiempo tr(max)+tSU;DAT=1000+250=1250 ns antes de que la línea SCL sea relajada (valores acordes a la especificación del modo normal).

(5) Cb=capacidad total de una línea del bus, en picofaradios. En topologías mixtas se admiten tiempos de bajada más rápidos, conformes a la tabla 1.6.

En los cronogramas mostrados en las figuras 1.16 (modos normal y rápido) y 1.17 (modo de alta velocidad) se puede observar el significado de cada uno de estos parámetros.

1.10.1 Modos Rápido/Normal

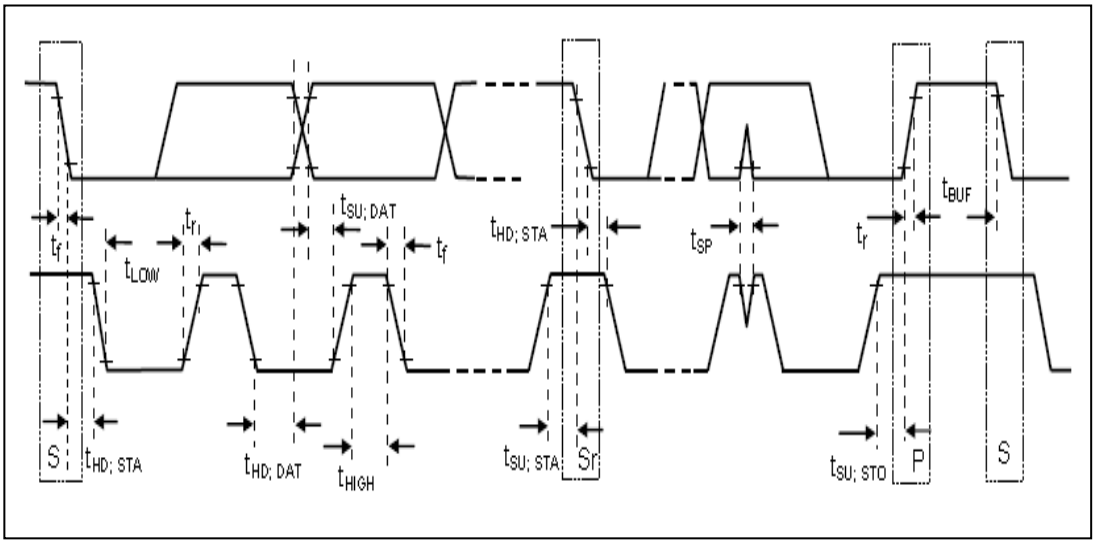


Figura 1.16 Parámetros de temporización en modo normal y rápido

1.10.2 Modo de Alta Velocidad (Hs)

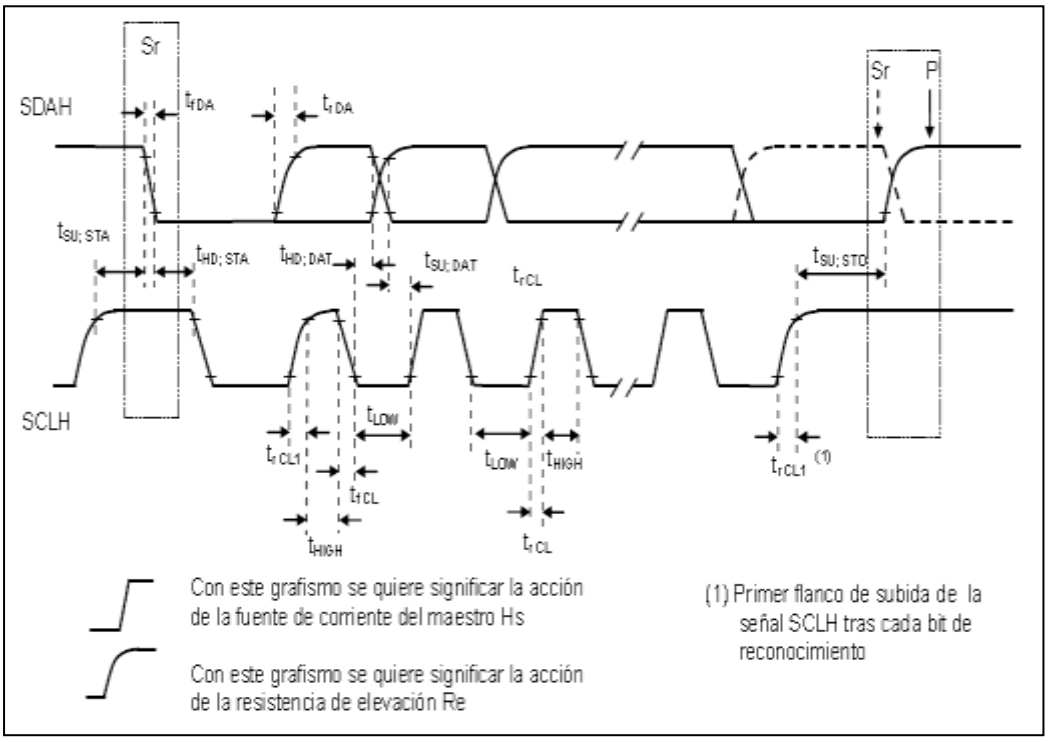


Figura 1.17 Parámetros de temporización en modo de Alta Velocidad

PARÁMETRO	SÍMBOLO	Cb = 100 pF máximo		Cb = 400 pF (2)		UNIDAD
		mínimo	máximo	mínimo	máximo	
Frecuencia del reloj SCLH	fSCLH	0	3’4	0	1’7	Mhz
Tiempo de asentamiento en la condición de (RE-) INICIO	tSU;STA	160	--	160	--	ns
Tiempo de retención en la condición de (RE-) INICIO.	tHD;STA	160	--	160	--	ns
Periodo a BAJA de la señal de reloj SCLH	tLOW	160	--	320	--	ns
Periodo a ALTA de la señal de reloj SCLH	tHIGH	60	--	120	--	ns
Tiempo de asentamiento de un dato	tSU;DAT	10	--	10	--	ns
Tiempo de retención de un dato:	tHD;DAT	0(3)	70	0(3)	150	ns
Tiempo de subida de SCLH	trCL	10	40	20	80	ns
Tiempo de subida de SCLH tras una condición de REINICIO y tras un bit de reconocimiento	trCL1	10	80	20	160	ns
Tiempo de bajada de SCLH	tfCL	10	40	20	80	ns
Tiempo de subida de SDAH	trDA	10	80	20	160	ns
Tiempo de bajada de SDAH	tfDA	10	80	20	160	ns
Tiempo de asentamiento para una condición de PARADA	tSU;STO	160	--	160	--	ns
Capacidad de la carga en SDAH o SCLH	Cb (2)	--	100		400	pF
Capacidad de la carga en SDAH+SDA o SCLH+SCL	Cb	--	400		400	pF
Margen de ruido a nivel BAJO para cada dispositivo conectado el bus (incluyendo histéresis)	VnL	10% VDD	--	10% VDD	--	V
Margen de ruido a nivel ALTO para cada dispositivo conectado el bus (incluyendo histéresis)	VnH	20% VDD	--	20% VDD	--	V

Tabla 1.6 Parámetros de temporización en el modo de alta velocidad (1)

(1) Todos los valores están referidos a los niveles VIHmin y VILmax. (véase la tabla 1.4)

(2) Los valores de temporización han de interpolarse linealmente para cargas en la línea, Cb, comprendidas entre los 100 y los 400 pF.

(3) Un dispositivo de proporcionar internamente y tiempo de retención de dato para cubrir la zona indefinida del cambio a baja de SCLH entre los instantes VIH y VIL. Para minimizar este problema un dispositivo ha de procurar tener este umbral lo más pequeño posible.

Capítulo 2

Fundamento Teórico

2.1 Introducción

El estudio de los microcontroladores AVR-ATMEL, así como el desarrollo de este proyecto, no consiste solo en dominar su arquitectura interna sino también en conocer programas auxiliares que facilitan el diseño de los sistemas donde intervienen.

Entre los distintos programas establecidos para el desarrollo de este tipo de proyectos, se ha hecho uso del paquete PROTEUS VSM de Labcenter Electronics y el compilador AVR-Studio de ATMEL. El Programa PROTEUS VSM es una herramienta para la verificación vía software que permite comprobar, prácticamente en cualquier diseño, la eficacia del programa desarrollado. Su combinación de simulación de código de programación y simulación mixta SPICE permite verificaciones analógico-digitales de sistemas basados en microcontroladores.

Por otra parte, se tiene el compilador AVR-Studio de ATMEL, herramienta que nos permite conocer y desarrollar programas en lenguaje ensamblador, además junto con el paquete de software libre WINAVR nos permite la compilación de proyectos en lenguaje C/C++.

2.2 Herramientas de Diseño

2.2.1 Proteus

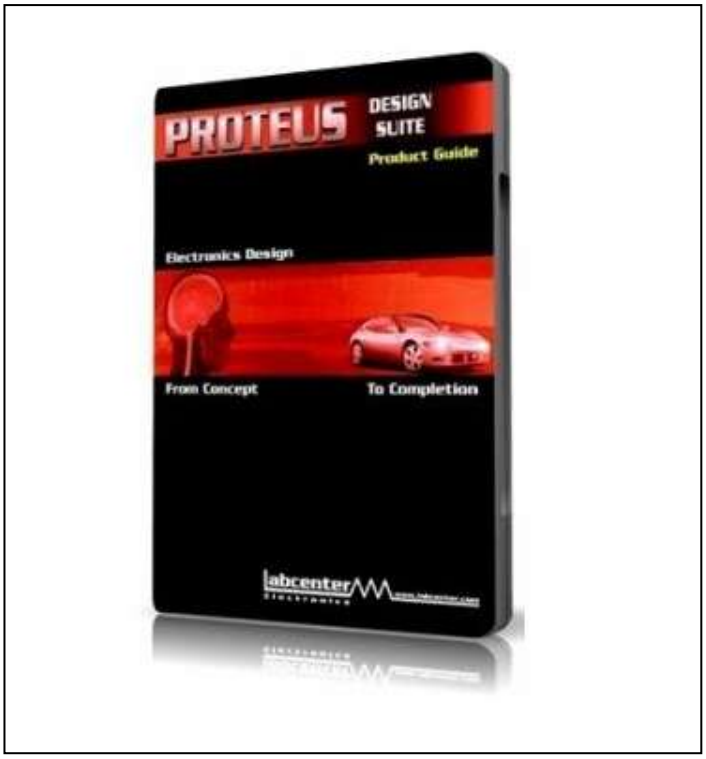


Figura 2.1 Paquete de Diseño Proteus de Labcenter Electronics

La herramienta de diseño PROTEUS VSM de Labcenter Electronics, ofrece la posibilidad de simular código microcontrolador de alto y bajo nivel y, simultáneamente con la simulación en modo mixto de SPICE. Esto permite el diseño tanto a nivel de hardware como software y realizar la simulación en un mismo y único entorno. Para ello, se suministran tres potentes subentornos de desarrollo:

- ISIS, para el diseño a nivel gráfico.
- VSM (Virtual System Modelling), para la simulación.
- ARES, para el diseño de placa a nivel físico.

2.2.2 AVR Studio

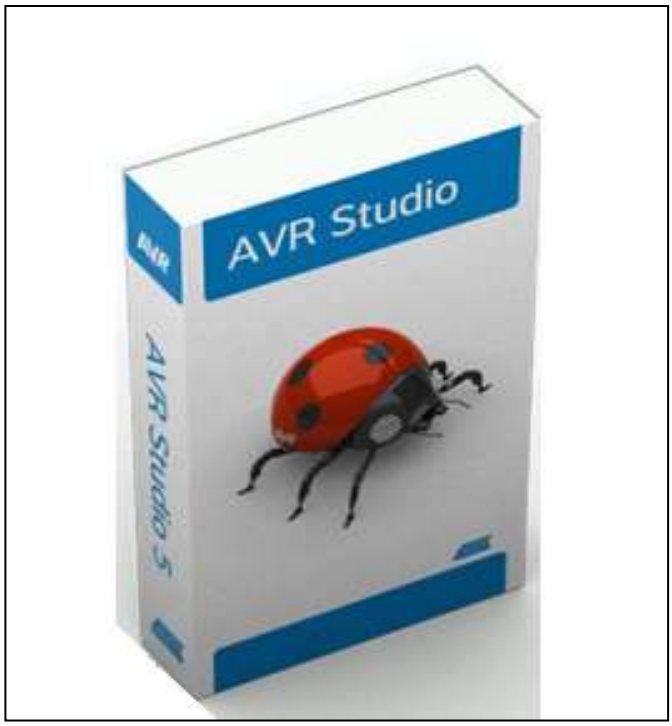


Figura 2.2 Paquete de Diseño AVR Studio de ATMEL

AVR-Studio es un compilador desarrollado por ATMEL. Incluye herramientas de compilación en ensamblador, y trabaja en conjunto con WinAVR para proveer el compilador de C/C++, AVR-GCC. Sin embargo, la última versión incluye soporte directo del compilador AVR-GCC, sin necesidad de instalar adicionalmente WinAVR. AVR Studio soporta TODOS los microcontroladores de ATMEL de la familia AVR, así como las herramientas hardware que ellos ofrecen (programadores, depuradores, tarjetas de desarrollo, etc). AVR Studio se encuentra disponible exclusivamente para sistemas operativos Windows, sin embargo existen casos en los

que han podido instalar exitosamente AVR Studio en Linux usando la herramienta Wine. AVR Studio es además completamente gratuito y no impone restricciones en cuanto a tamaño de código generado.

Figura 2.3 Herramienta de Software Libre para compilación de proyectos en C/C++

2.3 Descripción del Hardware

2.3.1 Introducción a los Microcontroladores

Un microcontrolador es un chip o circuito integrado que contiene todos los elementos de una CPU (Procesador, RAM, ROM, E/S). Estos dispositivos nacieron a finales de la década del 70' para brindar una solución a los caros y complejos sistemas basados en lógica discreta. Cotidianamente observamos cientos de aplicaciones en donde se utilizan microcontroladores tales como, hornos eléctricos digitales, automóviles, PLC's en la industria, etc.

En el presente trabajo se mencionan principalmente los dispositivos que pertenecen a la familia AVR de la empresa ATMEL, microcontroladores muy utilizados dentro del ámbito aficionado y también dentro de la industria. Este tipo de dispositivos son del tipo RISC (conjunto de instrucciones reducido). Lo cual quiere decir que su conjunto de instrucciones es muy reducido en el orden de 30 a 200 instrucciones que puede ejecutar, salvo alguna de ellas, en el orden de 1 ciclo de máquina.

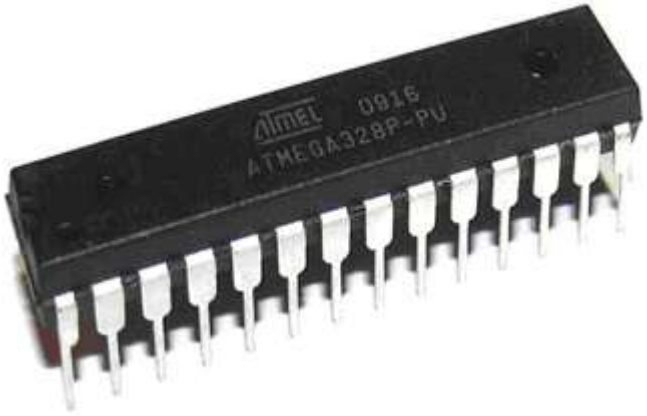


Figura 2.4 Microcontrolador ATmega328P de ATMEL

2.3.2 Estructura General de un microcontrolador

A continuación observamos los componentes básicos de cualquier microcontrolador. Los componentes que conforman cada dispositivo suelen variar según el fabricante y la arquitectura que posea.

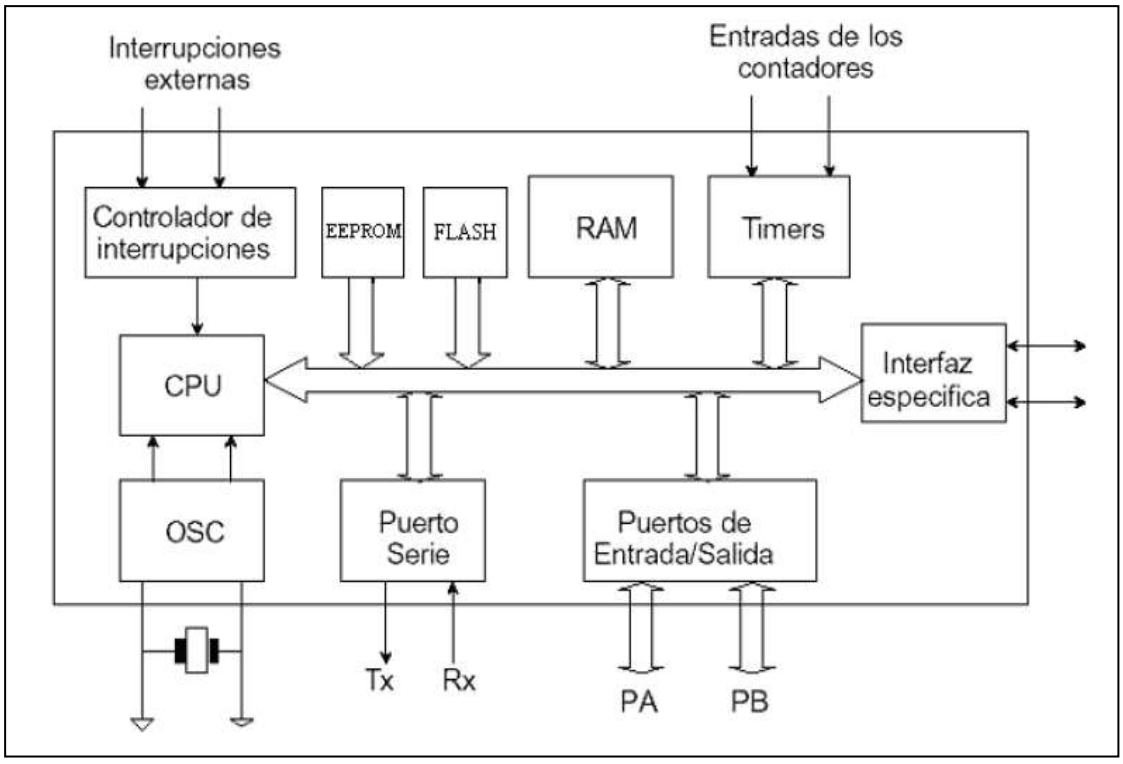


Figura 2.5 Estructura general de un microcontrolador

2.4 MCU AVR ATMega 32 [1]

2.4.1 Características Generales

Microcontrolador AVR de 8 bits de alto rendimiento y bajo consumo. Arquitectura Avanzada RISC

- Conjunto de 131 instrucciones. La mayoría de un solo ciclo de reloj de ejecución.
- 32 registros de trabajo de 8 bits de propósito general.
- Funcionamiento estático total.
- Capacidad de procesamiento de unos 20 MIPS a 20 MHz.
- Multiplicador por hardware de 2 ciclos

Memorias de programa y de datos no volátiles de alta duración

- 16/32/44 K bytes de FLASH auto programable en sistema
- 512B/1K/2K bytes de EEPROM
- 1/2/4K bytes de SRAM Interna
- Ciclos de escritura/borrado: 10.000 en Flash / 100.000 en EEPROM
- Retención de Datos: 20 años a 85ºC / 100 años a 25ºC
- Sección opcional de código Boot con bits de bloqueo independientes.
    - Programación en el sistema. A través del programa de Arranque presente en el chip.
    - Operación Real de lectura durante la escritura.

- Bloqueo programable para seguridad del software.

Interfaz JTAG (conforme el Standard IEEE 1149.1)

- Exploración limitada de acuerdo a Capacidad acorde al estándar JTAG
- Soporte Extendido para depuración dentro del chip
- Programación de FLASH, EEPROM, fusibles y bits de bloqueo a través de la interfaz JTAG.

Características de los periféricos

- Dos Temporizadores/Contadores de 8 bits con Preescalamiento separado y modo de comparación.
- Un Temporizador/Contador de 16 bits con Preescalamiento separado, modo de Comparación y modo de Captura.
- Contador en Tiempo Real con Oscilador separado
- 4 Canales para PWM
- 8 canales y 10bits para ADC
    - 8 Canales Unipolares
    - 7 Canales diferenciales presentes en Encapsulado TQFP
    - Canales diferenciales con Ganancia Programable a 1x, 10x o 200x.
- Byte orientado a interfaz serie de Dos-Hilos
- Puerto USART Programable
- Interfaz Serie SPI Maestro/Esclavo

- Temporizador Watchdog Programable con oscilador independiente, presente en el mismo Chip.
- Comparador Analógico dentro del mismo Chip
- Interrupción y encendido por cambio de estado en pines

Características especiales del microcontrolador

- Reset al Encendido y Detección Brown-Out Programable..
- Oscilador RC interno calibrado.
- Fuentes de interrupción externas e internas.
- 6 modos de descanso: Idle, Reducción de Ruido ADC, Power-Save, Power-Down, Standby y Standby extendido.

Encapsulados para Entradas/Salidas (E/S)

- 32 líneas de E/S programables.
- PDIP de 40 pines, TQFP y QFN/MLF de 44 pines.

Voltaje de Operación

- 2.7 – 5.5V

Velocidad de Funcionamiento

- 0 – 16MHz @ 1.8 – 5.5V - 10MHz @ 2.7 –

Consumo de energía a 1MHz, 3V, 25ºC

- Activo: 0.6mA
- Modo Idle: 0.2uA
- Modo Power-down < 0.1µA

2.4.2 Distribución de Pines

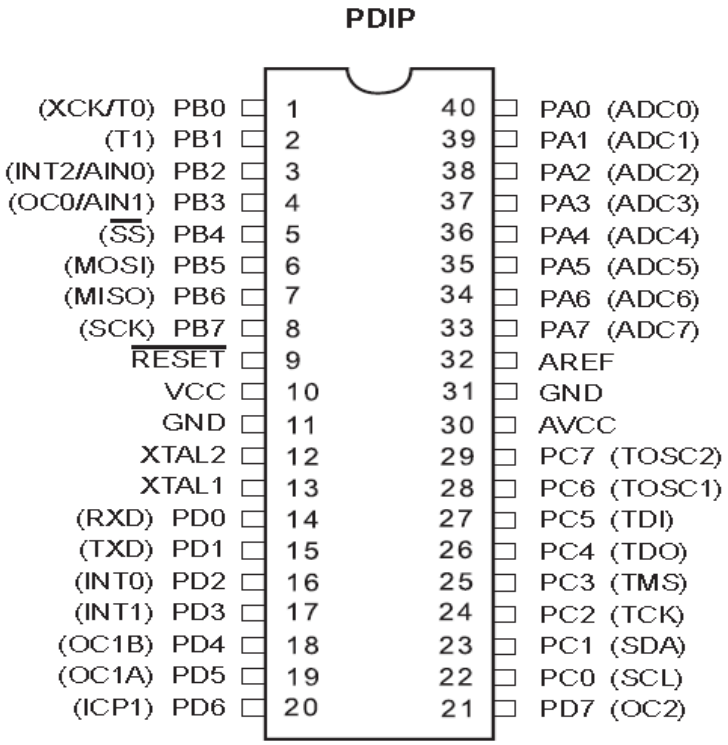


Figura 2.6 Pines correspondientes al MCU ATmega32

2.4.3 Revisión Global

El ATMEL AVR ATmega32A es un microcontrolador CMOS de bajo consumo basado en la arquitectura RISC mejorada. El ATmega32A consigue una tasa de transferencia de información alrededor de 1 MIPS por MHz admitido por el sistema, permitiendo al diseñador del sistema optimizar el consumo de energía versus la velocidad de procesamiento.

2.4.4 Diagrama de Bloques

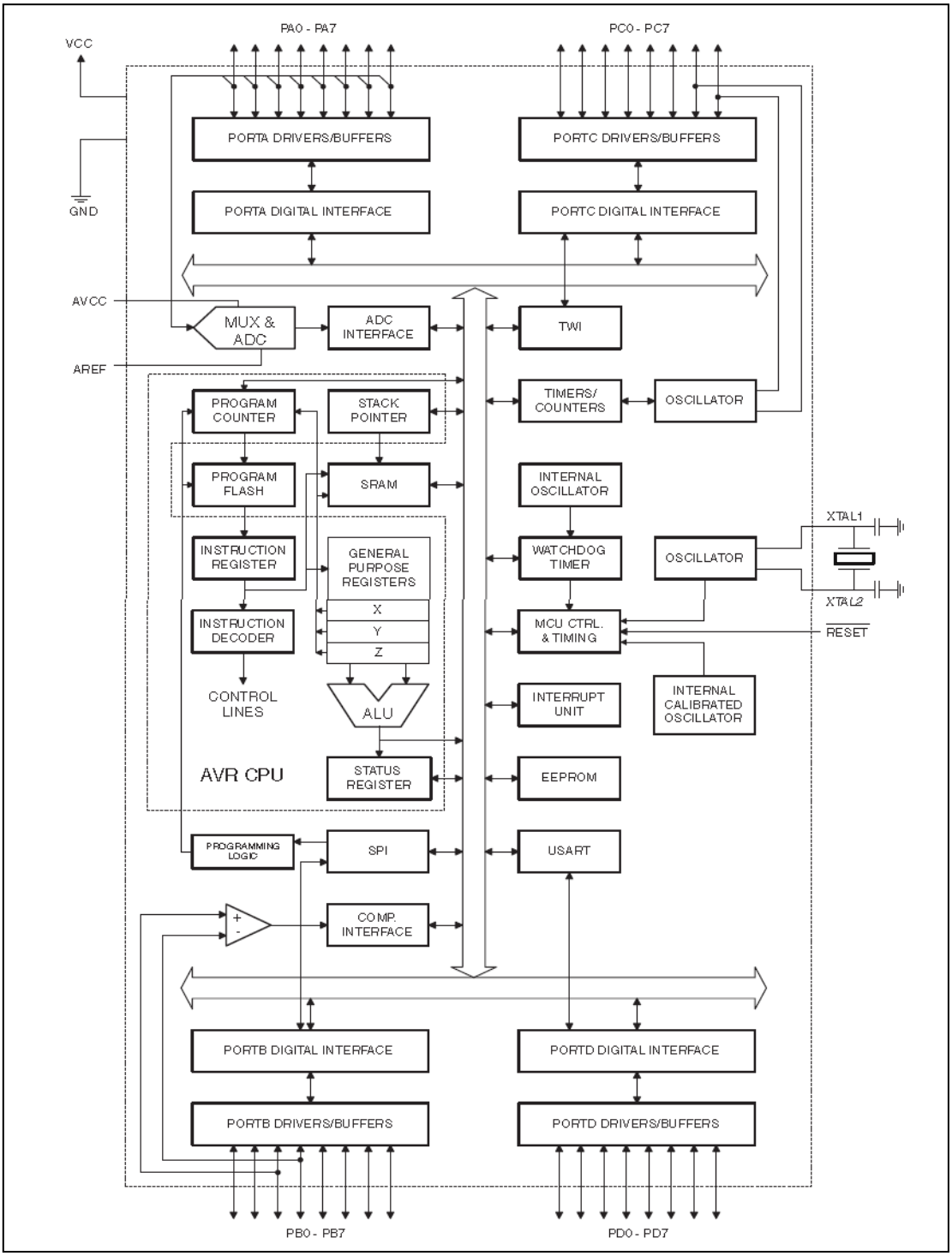


Figura 2.7 Diagrama de Bloques del ATmega32A

El núcleo (Core) AVR combina un conjunto de instrucciones RISC con 32 registros para uso de propósito general. Todos los 32 registros están directamente relacionados

con la Unidad Aritmética Lógica (ALU), admitiendo dos registros independientes al ejecutarse una instrucción en un ciclo de máquina. El resultado de esta arquitectura es más eficiente, se consigue un caudal de flujo y transferencia hasta diez veces más rápido que microcontroladores CISC convencionales.

El ATmega32A provee las siguientes características: 32K bytes en el sistema de Flash Programable con capacidad de lectura y escritura de 1024bytes en la EEPROM, 2K bytes en la SRAM, 32 pines de E/S para propósito general, 32 registros de propósito general, tres Temporizadores/Contadores flexibles con modo de Comparación y PWM, 2 USARTs, un byte orientado a la Interfaz Serial de 2 hilos, 8 canales ADC de 10 bits con opción de entrada Diferencial con ganancia programable, Temporizador Watchdog programable con oscilador interno, un Puerto serial SPI, Interfase de prueba JTAG, también usado para acceder al sistema de Arranque presente en el Chip y seis modos de programación seleccionable para ahorro de energía. El modo Idle detiene al CPU mientras permite a la SRAM, Temporizador/Contador, Puerto SPI y un sistema de interrupción para funcionamiento continuo.

El Modo Power-down guarda el contenido de los registros pero paraliza al oscilador, desactiva todas las otras funciones de chip hasta la próxima interrupción o mediante reseteo por hardware. En el Modo Power-save, el reloj asincrónico continúa corriendo, permitiendo tener actualizado al reloj mientras el resto de dispositivos están descansando. El Modo de Reducción del Ruido del ADC detiene al CPU y a todos los módulos de E/S excepto al Reloj Asincrónico y al ADC, para minimizar el ruido durante la conversión. En el Modo Standby, el oscilador Cristal/Resonador está corriendo mientras el resto de dispositivos están descansando. Estos permiten

comenzar una rápida combinación con el consumo de baja energía. En el Modo de Standby extendido, corre el Oscilador principal y el Reloj Asincrónico. Este elemento es hecho usando tecnología de alta densidad de memoria no volátil de ATMEL. El chip interno ISP de la FLASH permite a la memoria de programa ser reprogramada a través del puerto interno ISP mediante un programador convencional no volátil o mediante un programa interno en el dispositivo AVR. El programa de inicialización puede usar cualquier interfaz para descargar el programa de aplicación en la memoria flash. El programa en la sección Flash de Arranque es actualizado mientras continúa corriendo la sección de aplicaciones de la Flash, proporcionando una escritura/lectura verdadera de operación. Para combinar un CPU RISC de 8 bits en un sistema de Flash Auto-programable en un chip monolítico, el ATmega32A es un poderoso microcontrolador que provee una alta flexibilidad y solución de costos efectivos para cualquier aplicación de control.

El ATmega32A AVR es soportado con un juego completo de programas y herramientas de desarrollo del sistema incluyendo: compiladores de C, ensambladores de macro, depurador / simuladores de programa, emuladores de circuitos y equipos de evaluación.

2.4.5 Comparación entre el ATmega16, ATmega32 and ATmega64

Dispositivo	Flash	EEPROM	RAM
ATmega16	16 Kbyte	512 bytes	1 Kbyte
ATmega32	32 Kbyte	1 Kbyte	2 Kbyte
ATmega32	64 Kbyte	2 Kbyte	4 Kbyte

Tabla 2.1 Diferencias entre el ATmega16, ATmega32, ATmega64

2.4.6 Descripción de Pines

- VCC

  Alimentación de Voltaje Digital

- GND

  Tierra

- Puerto A (PA7:PA0)

  El puerto A sirve como entradas analógicas para el conversor Análogo Digital. El puerto A también sirve como un puerto bidireccional de 8 bits con resistencias internas de pull up (seleccionables para cada bit). Los buffers de salida del puerto A tienen características simétricas controladas con fuentes de alta capacidad. Los pines del puerto A están en tri-estado cuando las condiciones de reset están activadas o cuando el reloj no este corriendo. El puerto A también sirve para varias funciones especiales del ATmega164P/324P/644P como la Conversión Análoga Digital.

- Port B (PB7:PB0)

  El puerto B es un puerto bidireccional de 8 bits de E/S con resistencias internas de pull up. Las salidas de los buffers del puerto B tienen características simétricas controladas con fuentes de alta capacidad. Los pines del puesto B están en tri-estado cuando las condiciones de reset están activadas o cuando el reloj no esté corriendo. El puerto B también sirve para varias funciones

especiales del ATmega164P/324P/644P como se menciona en las páginas iniciales.

- Port C (PC7:PC0)

  El puerto C es un puerto bidireccional de 8 bits de E/S con resistencias internas de pull up (seleccionadas por cada bit). Las salidas de los buffers del puerto C tienen características simétricas controladas con fuentes de alta capacidad. Los pines del puesto C están en tri-estado cuando las condiciones de reset están activadas siempre y cuando el reloj no este corriendo. El puerto C también sirve para las funciones de Interfaz del JTAG, con funciones especiales del ATmega164P/324P/644P como se menciona en las páginas iniciales.

- Port D (PD7:PD0)

  El Puerto D es un puerto bidireccional de entradas y salidas con resistencias internas de pull up (seleccionadas por cada bit). Las salidas de los buffers del puerto D tienen características simétricas controladas con sumideros de fuentes de alta capacidad. Los pines del Puerto D están en tri-estado cuando llega una condición de reset activa, siempre y cuando el reloj no esté corriendo. El puerto D también sirve para varias funciones especiales del ATmega164P/324P/644P como se menciona en las páginas iniciales.

- RESET

  Entrada del Reset. Un pulso de nivel bajo en este pin por períodos de pulso mínimo genera un reset, siempre y cuando el reloj no esté corriendo. La longitud del pulso mínimo está especificada en las Características y Sistemas

de Reset (Páginas 331 del Data Sheet). Pulsos cortos no son garantizados para generar un reset.

- XTAL1

  Entrada para el amplificador del oscilador invertido y entrada para el circuito de operación del reloj interno.

- XTAL2

  Salida desde el Oscilador amplificador invertido.

- AVCC

  AVCC es la alimentación de voltaje para el pin del Puerto F y el Conversor Análogo a Digital. Este debe ser conectado externamente a VCC, siempre y cuando el ADC no sea usado. Si el ADC es usado, este deberá ser conectado a VCC a través de un filtro paso bajo.

- AREF

  Está es la referencia para el pin de la conversión Análoga a Digital.

2.4.7 Retención de Datos

La fiabilidad de la calificación de resultados muestra que la velocidad de falla de un proyecto es mucho menor que 1 PPM en 20 años a 85°C ó 100 años a 25°C.

2.4.8 Núcleo de la CPU AVR (CPU CORE)

2.4.8.1 Visión General

Esta sección discute la arquitectura general del AVR. La principal función del núcleo de la CPU AVR es asegurar la correcta ejecución del programa. La CPU debe ser capaz de acceder a la memoria, llevar a cabo cálculos, control de periféricos y atención de interrupciones.

Para maximizar el rendimiento y el paralelismo, el AVR usa una arquitectura de Hardware con memorias y buses separados para programa y datos.

Las instrucciones en la memoria de programa son ejecutadas con un simple nivel de colas. Mientras una instrucción es ejecutada, la siguiente instrucción es ejecutada desde la memoria de programa. Este concepto permite que las instrucciones sean ejecutadas en cada ciclo de máquina. La memoria de programa está en la memoria Flash re-programable.

El Archivo del Registro (Register File) de rápido acceso contiene 32 registros de propósito general de 8 bits trabajando en un simple ciclo de reloj. Esto permite una operación de ciclo simple en la Unidad Aritmética lógica. En una operación típica de la ALU, dos operandos están fuera del Archivo de Registro, la operación es ejecutada, y el resultado es guardado en el Archivo de Registro en un ciclo de máquina.

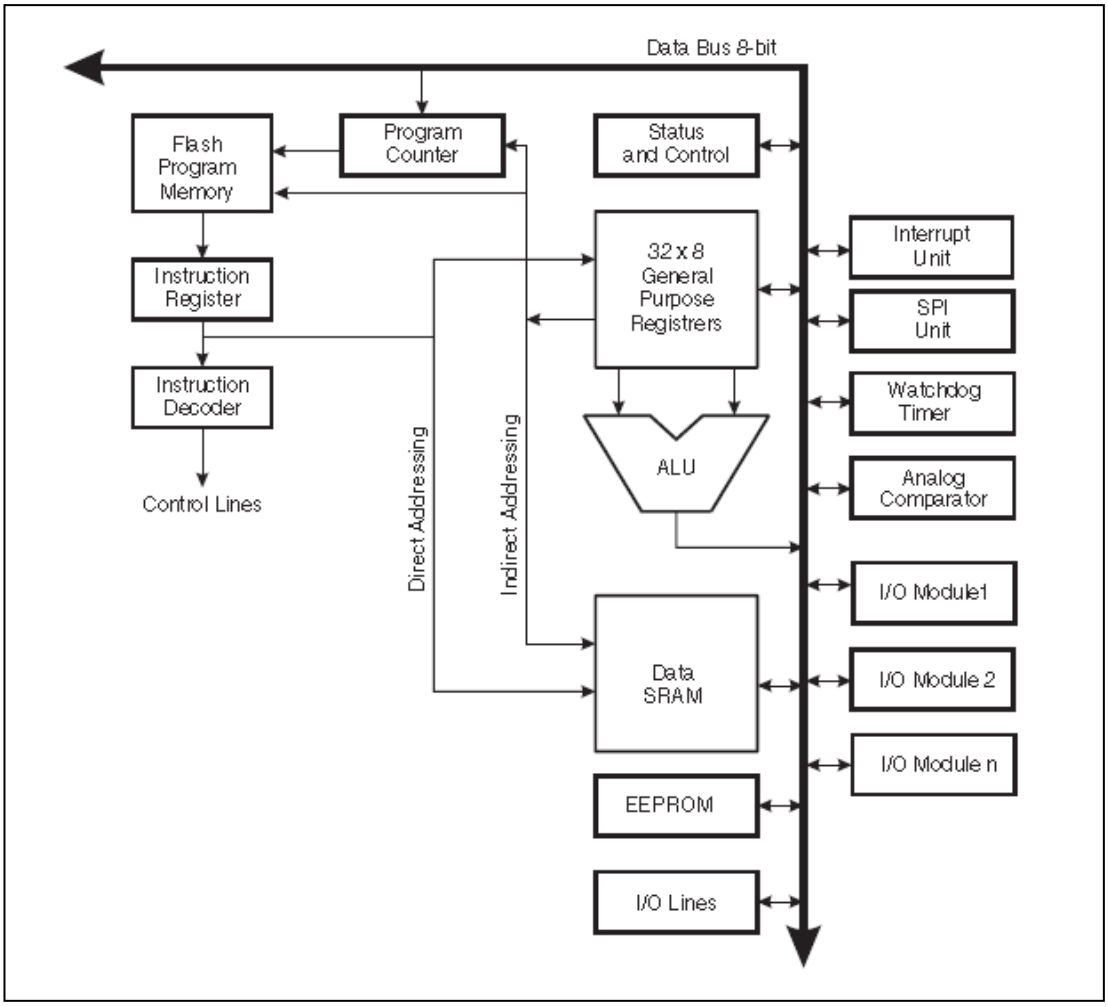


Figura 2.8 Diagrama de Bloques de la Arquitectura AVR

Seis de los 32 registros pueden ser utilizados como tres registros punteros de 16 bits de dirección, para direccionar los Datos y permitir los cálculos de direcciones diferentes. Uno de estos tres punteros puede ser usado como un puntero de direcciones para tablas en la memoria de programa de la Flash. Estos registros de función adicionales son los registros X, Y y Z de 16 bits, descritos después en esta sección. La ALU soporta operaciones lógicas y aritméticas entre registros o entre constantes y registros. Simples operaciones de registros pueden ser ejecutadas en la ALU. Después de una operación aritmética, el registro de estado es actualizado para reflejar información acerca de los resultados de la operación. El flujo del programa es provisto

de un salto condicional e incondicional y llamado de interrupciones, capaz de direccionar espacios de direcciones completamente.

La mayoría de instrucciones del AVR tienen un formato simple de una palabra de 16 bits. Toda dirección en memoria de programa contiene una instrucción de 16 o 32 bits. El espacio de la memoria de programa flash está dividido en dos secciones, la sección Baja del programa y la sección de aplicación de programa. Ambas secciones presentan bits de bloqueo de escritura y protección de lectura/escritura. La instrucción SMP que se escribe en la Sección de la memoria Flash debe residir en la sección Baja del programa. Durante los llamados de interrupción y subrutinas, la dirección de retorno del Contador de Programa (CP) es almacenado en la pila (stack). La pila (stack) está localizada efectivamente en la SRAM (RAM estática) de datos y consecuentemente el tamaño de la pila está limitado solo por el tamaño total de la SRAM (RAM estática) y su uso.

Todos los programas a usarse deben inicializar en el SP (Stack Pointer) en la rutina del Reset (antes de que sea ejecutada una interrupción o una subrutina). El Puntero de Pila (Stack pointer SP) es la lectura/escritura accesible en el espacio de E/S. La RAM estática de datos puede ser fácilmente penetrada a través de los cinco diferentes modos de direccionamiento soportados en la arquitectura de AVR.

El espacio de memoria en la arquitectura de los mapas de memoria de los AVR son todos lineales y regulares. Un módulo de interrupción flexible tiene sus registros de control en los espacios de E/S con una Habilitación de Interrupción Global en el Registro de Estado. Todas las interrupciones tienen separado un vector de

interrupciones en la tabla del vector de interrupciones. Las interrupciones tienen prioridad de conformidad con su vector de interrupciones. La dirección más baja del vector de interrupciones tiene alta prioridad.

El espacio de memoria de E/S contiene 64 direcciones para las funciones periféricas de la CPU, el Registro de Control SPI y otras funciones de Entrada y Salida. La memoria de Entrada y Salida puede ser accedida directamente o como localidades de espacio de datos siguiendo estos Archivos de Registro: 0x20 - 0x5F. En suma, el ATmega164P/324P/644P tiene espacios extendidos de entrada y salida desde la dirección 0x60 - 0xFF en la SRAM donde solo las instrucciones ST/STS/STD y LD/LDS/LDD pueden ser usadas.

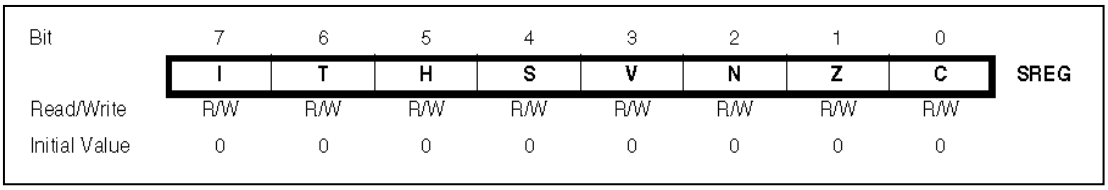
2.4.9 Registro de Estado

El registro de estado contiene información acerca de los resultados de las instrucciones aritméticas más recientes ejecutadas. Está información puede ser usada para alterar el flujo del programa en el funcionamiento de operaciones condicionales. Note que el Registro de Estado es actualizado después de todas las operaciones de la ALU, como especificaciones en el Set de Instrucciones de Referencia.

En algunos casos esto retira la necesidad de usar comparación de instrucciones dedicadas, resultando un código más rápido y compacto. El Registro de estado no es almacenado automáticamente cuando entra una rutina de interrupción y se restituye cuando regresa de una interrupción.

2.4.9.1 Registro de Estado: SREG-AVR

El registro de Estado AVR – SREG, está definido por:



Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

El bit de habilitación de las interrupciones globales debe estar en uno para habilitar las interrupciones. La interrupción individual permite que el control sea llevado a cabo en registros de control distintos. Si el registro de Habilitación de interrupciones globales es borrado, ninguna de las interrupciones están activadas independiente de la configuración de una interrupción individual. El bit I es limpiado por hardware después de que una interrupción ha ocurrido, y es puesto en uno por la instrucción de RETI para habilitar interrupciones siguientes. El bit I también puede ser puesto en uno y borrado por la aplicación con las instrucciones SEI y CLI.

Bit 6 – T: Bit Copia de almacenamiento

El bit de instrucción de copia BLD (cargar bit) y BST (almacenar bit) usa el bit T como una fuente o destino para la operación del bit. Un bit desde un registro en el Archivo de Registro pueden ser copiados en el bit T mediante la instrucción SBT y un bit en T puede ser copiado dentro de un bit en un registro en el Archivo de registros mediante la instrucción BLD.

- Bit 5 – H: Half Carry Flag

  Half Carry es útil en la aritmética BCD.

- Bit 4 – S: Bit de Signo, $S = N \oplus V$

  El Bit S es una OR exclusiva entre la bandera negativa N y la bandera de desbordamiento V en Complemento a Dos.

- Bit 3 – V: Bandera de Desbordamiento V en Complemento a Dos

  La bandera de desbordamiento en Complemento a Dos soporta el complemento a dos.

- Bit 2 – N: Bandera Negativa

  La Bandera Negativa N indica un resultado negativo en una operación aritmética o lógica.

- Bit 1 – Z: Bandera del Cero

  La bandera del cero indica si un resultado es cero en una operación aritmética o lógica.

- Bit 0 – C: Bandera de Acarreo

  El Bit C indica la existencia de acarreo en una operación aritmética o lógica.

2.4.10 Archivo de Registros de Propósito General

El Archivo de Registro es optimizado por el Juego de instrucciones RISC del AVR. Para lograr la actuación y flexibilidad requerida, los siguientes esquemas de entrada/salida son soportados por el Archivo de Registro:

- Un operador de salida de 8 bits y una entrada resultante de 8 bits
- Dos operadores de salida de 8 bits y una entrada resultante de 8 bits
- Dos operadores de salida de 8 bits y una entrada resultante de 16 bits
- Un operador de salida de 16 bits y una entrada resultante de 16 bits

La siguiente Figura muestra la estructura de los 32 Registros de Propósito General que trabajan en la CPU.

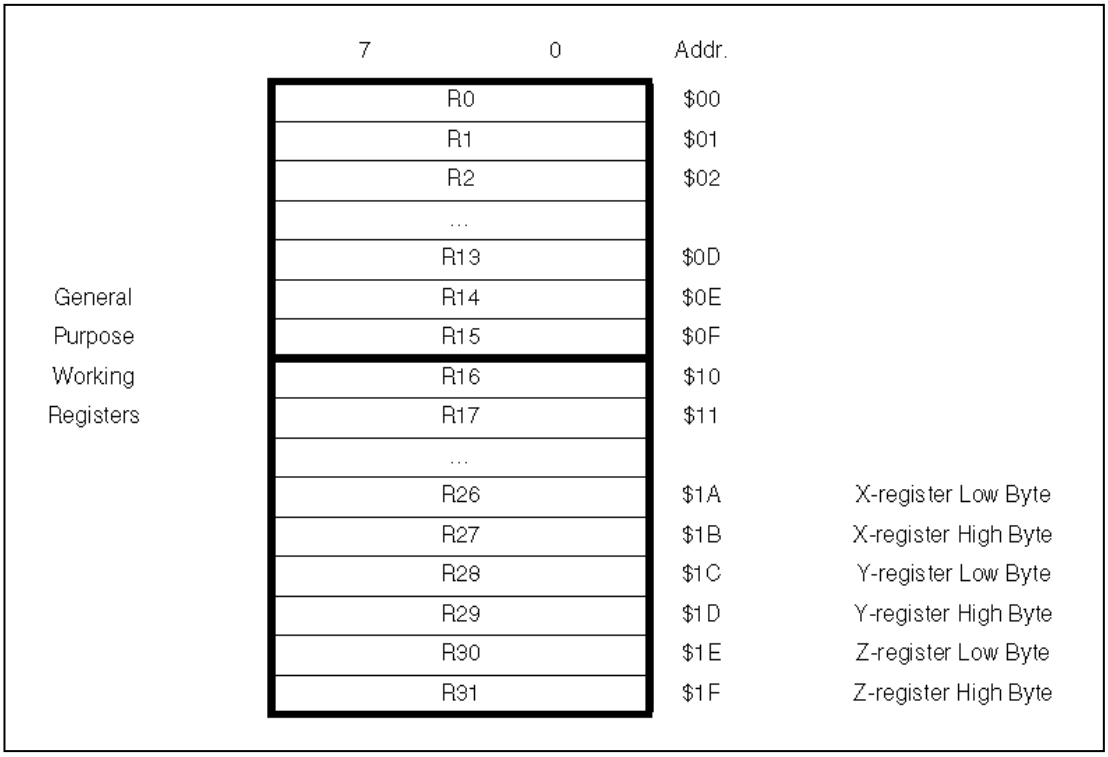


Figura 2.9 AVR-CPU Registro de Trabajo de Propósito General

La mayoría de las operaciones de instrucciones en el Archivo de Registro tiene acceso directo a todos los registros, y la mayoría de ellos son instrucciones de un ciclo. Como se muestra en la figura anterior a cada registro en la memoria de datos se le asigna una

dirección, mapeados estos directamente dentro de las 32 localidades para el uso del espacio de Datos.

Aunque físicamente no son implementados en las localidades de la SRAM, esta memoria provee gran organización flexible para el acceso a estos registros, como los registros punteros X, Y y Z que se usan como índices de cualquier archivo de registro.

2.4.10.1 Registros: X, Y y Z

Los registros R26… R31 tienen algunas funciones adicionales para uso de propósito general. Estos registros son punteros de 16 bits de dirección para direccionar indirectamente al espacio de datos. Los tres registros de direccionamiento indirecto X, Y y Z están definidos como se muestra en la figura siguiente.

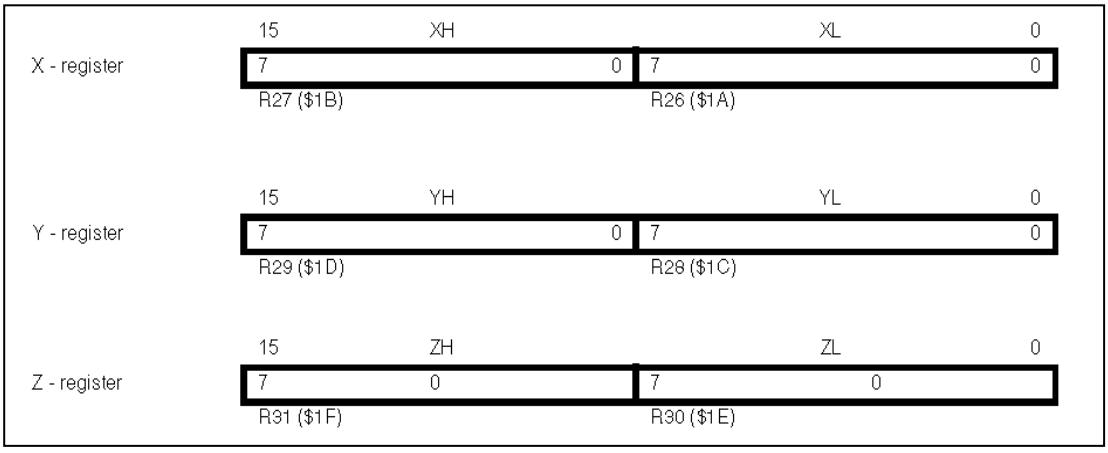


Figura 2.10 AVR-CPU Registros X, Y y Z.

En los modos de direccionamiento directo estos registros de dirección tienen desplazamientos fijos, incrementos y decrementos automáticos.

2.4.11 Puntero de Pila (Stack Pointer)

La Pila es utilizada principalmente para almacenar información temporal, para almacenar variables locales y para almacenar direcciones de retorno después de una interrupción o llamado de subrutinas. Los registros del Puntero de Pila siempre apuntan a la parte superior de la pila. Note que la pila es implementada como un crecimiento desde la localidad más alta a la localidad mas baja de la memoria. Esto implica que un comando PUSH decrementa el Puntero de Pila. El puntero de pila para la SRAM es el área donde las interrupciones y subrutinas son localizadas. Estos espacios de pila en los datos de la SRAM deben ser definidos por el programa antes de que cualquier llamado a subrutina sea ejecutado o una interrupción sea habilitada.

El puntero de pila debe estar por encima de la localidad 0x0100. El valor inicial del puntero de pila es la última dirección de la SRAM interna. El puntero de pila es decrementado a uno cuando los datos presionan en la pila con la instrucción PUSH, y este es decrementado por tres cuando los datos llenan la pila con un regreso desde una subrutina RET o de regreso desde una interrupción RETI.

Instrucción	Puntero de Pila	Descripción
PUSH	Decrementado en 1	El dato es colocado en la pila
CALL ICALL RCALL	Decrementado en 2	Dirección de retorno es colocada en la pila con un llamado a subrutina o interrupción.
POP	Incrementado en 1	El dato es extraído desde la pila.
RET RETI	Incrementado en 2	Dirección de retorno es extraída desde la pila con retorno desde subrutina o interrupción.

Tabla 2.2 Instrucciones del Puntero de Pila

El Puntero de Pila del AVR es implementado como un registro de 8 bits en el espacio de E/S. El número de bits actualmente usados es dependientemente implementado. Cabe notar que el espacio de datos en algunas implementaciones de la arquitectura AVR es tan pequeña que solo necesita el SPL. En este caso, el registro SPH no esta presente.

2.4.11.1 SPH y SPL – Puntero de Pila Alto y Puntero de Pila Bajo

Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Note que estos son los valores iniciales respectivamente para el ATmega16/32/64.

2.4.12 Manipulación del Reset y la Interrupción

El AVR provee diferentes Fuentes de interrupción. Cada una de estas fuentes de interrupción y el Vector Separador de Reset poseen un Vector separado de programa en el espacio de la memoria de programa. Todas las interrupciones son bits habilitados de forma individual los cuales deben ser escritos a uno lógico junto con el bit Habilitador de Interrupciones Globales en el registro de Estado.

Dependiendo del valor del contador del programa, las interrupciones deben ser deshabilitadas automáticamente cuando los bits de Bloqueo de Arranque (BLB02 o BLB12) son programados. Esta característica mejora la seguridad de software. La

dirección más baja en el espacio de la memoria de programa está definida por defecto como vectores de Reset e Interrupción. Existe una lista completa de vectores que se muestra en el DataSheet del microcontrolador. Esta lista también determina los niveles de prioridad de las diferentes interrupciones. El nivel de prioridad es desde la dirección más baja a la más alta. El RESET tiene la prioridad más alta y la siguiente es el pedido de interrupción externa (INTO).

Los vectores de interrupción pueden ser movidos al inicio de la sección de la parte baja de la Flash por la configuración del bit IVSEL en el MCU del registro de Control (MCUCR). El vector de Reset también puede ser movido al inicio de la sección de la parte mas baja por la programación del fusible BOOTRST. Cuando ocurre una Interrupción, el bit 'I' del Habilitador de Interrupciones Globales es limpiado y todas las interrupciones son deshabilitadas. El software del usuario puede escribir un uno lógico en el bit 'I' para habilitar de nuevo las interrupciones. Todas las interrupciones habilitadas pueden entonces interrumpir las rutinas de interrupciones. El bit 'I' es puesto a uno automáticamente cuando regresa de una interrupción al ser ejecutada la instrucción RETI.

Existen básicamente dos tipos de interrupciones. El primer tipo es disparado por un evento que configura la bandera de interrupción. Para estas interrupciones, el Contador de Programa es direccionado para el actual Vector de Interrupción a medida que se ejecuta una rutina de interrupción manual y limpiada por hardware la correspondiente bandera de interrupción. La bandera de interrupción puede también ser limpiada por un uno lógico para la posición de la bandera del bit. Si una condición de interrupción ocurre mientras se habilita el bit de interrupción correspondiente este

es limpiado, la bandera de interrupción será puesto a uno y la interrupción recordará ser siempre habilitada o a su vez la bandera puede ser limpiada por software. De igual forma, si una o más condiciones de interrupción ocurren mientras el bit de interrupción es limpiado, la correspondiente bandera de interrupción será puesta a uno y si la Bandera de Interrupción Global es habilitada mientras el bit de interrupción es puesto a uno, entonces la interrupción será ejecutada de acuerdo al orden de prioridad.

El segundo tipo de interrupción se activará siempre y cuando la condición de interrupción esté presente. Estas interrupciones no necesariamente tienen banderas de interrupción. Si la condición de interrupción desaparece antes de que la interrupción sea habilitada, la interrupción no será disparada. Cuando existe una interrupción en el AVR, esta siempre regresará al programa principal y ejecutará una o más interrupciones antes de cualquier pedido de interrupción.

Registro	Valor de Reset (hex)
PC	000000
R0-R31	00
DDRx	00
PORTx	00

Tabla 2.3 Valores de Reset para algunos Registros AVR

Note que el Registro de Estado no es almacenado automáticamente cuando entra una rutina de interrupción o cuando regresa de la rutina de interrupción. Esto deber ser hecho manualmente por software. Cuando usa la instrucción CLI para deshabilitar interrupciones, la interrupción será deshabilitada inmediatamente. Ninguna interrupción será ejecutada después de la interrupción CLI, Incluso si ocurre simultáneamente con la instrucción CLI.

2.4.13 Tiempo de respuesta de la Interrupción

La respuesta para la ejecución de interrupción de todas las interrupciones habilitadas del AVR es de mínimo cuatro ciclos de reloj. Después de cuatro ciclos de reloj la dirección del vector de programa para el manejo de la rutina de interrupción actual se ejecuta. Durante este periodo de cuatro ciclos, el Contador de Programa es guardado en el Stack (Pila). El vector es normalmente un salto para la rutina de interrupción, y este salto toma tres ciclos de reloj. Si una interrupción ocurre durante la ejecución de una instrucción de múltiples ciclos, esta instrucción es completada antes de que la interrupción sea atendida. Si una interrupción ocurre mientras el MCU está en modo de descanso, el tiempo de respuesta para la ejecución de interrupciones es incrementado en cuatro ciclos de reloj. Este incremento viene añadido desde el momento en que se selecciona el Modo de descanso (sleep). Un retorno desde una rutina de interrupción manual toma cuatro ciclos de reloj. Durante estos cuatro ciclos de reloj, el Contador de Programa (dos bytes) es extraído desde el Snack o pila, el puntero de Pila es incrementado en dos y el bit ‘I’ en la configuración del SREG es puesto a uno.

2.4.14 Sistema y Opciones de Reloj

2.4.14.1 Sistemas de Reloj y sus Distribuciones

La Figura 2.11 presenta los principales sistemas de reloj en el AVR y su distribución. Todos los relojes no necesariamente deben estar activos en un momento determinado. Con el fin de reducir el consumo de energía, los relojes de los módulos que no se

utilizan se pueden detener mediante el uso de distintos modos de espera. Los sistemas de reloj se detallan a continuación.

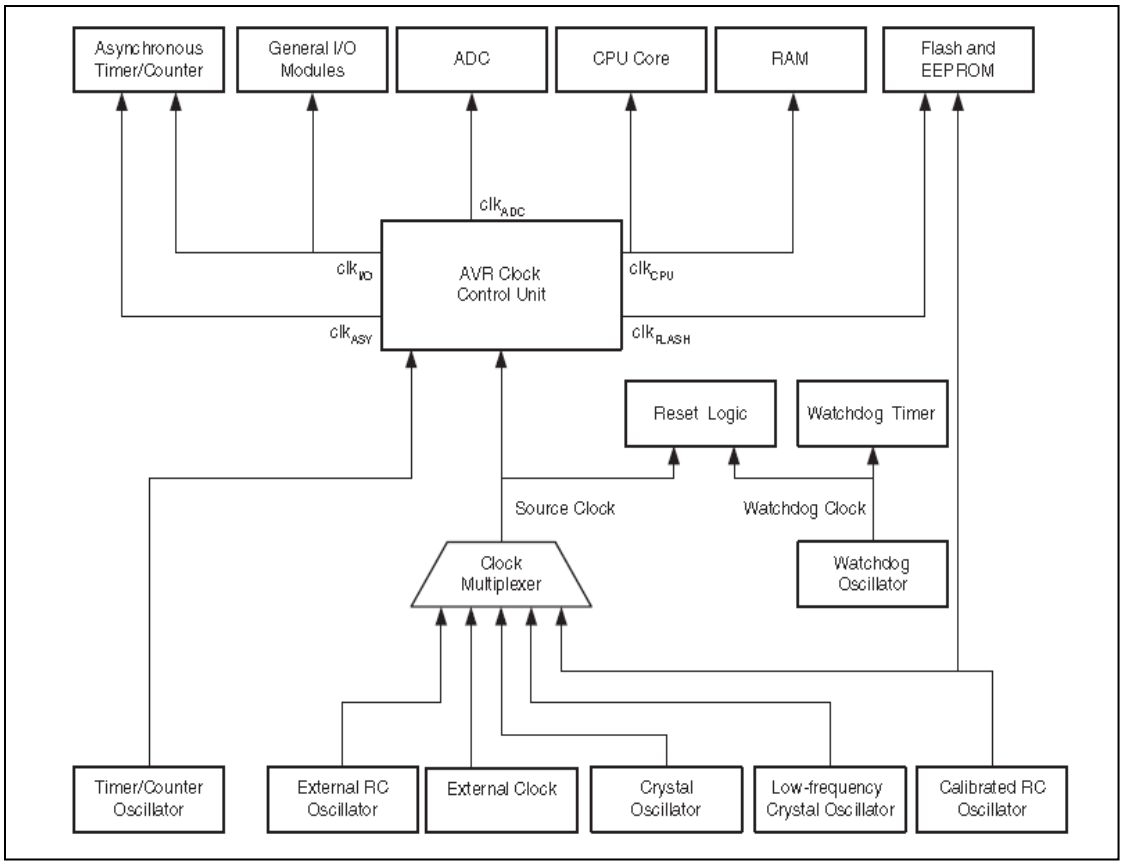


Figura 2.11 Distribución del Sistema de Reloj

2.4.14.2 Reloj del CPU – clk CPU

El reloj del CPU es dirigido a las diferentes partes del sistema concerniente a la operación del núcleo AVR. Ejemplos de estos módulos son el Archivo de Registro de Propósito General, el Registro de Estado y la Memoria de datos que mantiene el puntero de pila. Detener el reloj de la CPU inhibe el núcleo para la realización de operaciones generales y cálculos.

2.4.14.3 Reloj de E/S – clk I/O

El reloj de E/S es utilizado por la mayoría de los módulos de E/S, al igual que los contadores/temporizadores, SPI y USART. El reloj de E/S también es utilizado por el módulo de interrupción externa, pero cabe notar que algunas interrupciones externas son detectadas por lógica sincrónica, permitiendo a estas interrupciones ser detectados, incluso si el reloj de E/S ha sido detenido.

También se debe notar que la dirección de reconocimiento en el módulo TWI se lleva a cabo de forma asíncrona cuando el reloj de E/S es detenido, permitiendo la recepción de la dirección TWI en todos los modos de descanso (Sleep Modes).

2.4.14.4 Reloj de Flash – clk FLASH

El reloj de la Flash controla la operación de la interfase Flash. El reloj de la Flash es usualmente activado simultáneamente con el reloj del CPU.

2.4.14.5 Reloj de Temporizador Asincrónico – clk ASY

El Reloj de Tiempo Asincrónico permite al Temporizador/Contador asíncrono ser registrado directamente por un reloj de cristal externo de 32 kHz. El reloj dominante dedicado permite usar este Temporizador/Contador como un contador en tiempo real aún cuando el dispositivo se encuentre en modo de Descanso.

2.4.14.6 Reloj del ADC – clk ADC

El ADC es proporcionado con un reloj dominante dedicado. Esto permite detener el CPU y los relojes de E/S. lo cual reduce el ruido generado por los circuitos digitales. A su vez da una mayor presición a los resultados de la conversión ADC.

2.4.15 Fuentes de Reloj

El dispositivo tiene las siguientes opciones de fuente de reloj, seleccionable por los bits fusibles de la Flash, como se muestra a continuación. El reloj de la fuente seleccionada se introduce en el generador del reloj AVR, y dirigido a los distintos módulos correspondientes.

Dispositivo de Reloj Opcional	**CKSEL 3:0**
Cristal Externo/Resonador Cerámico	1111 – 1010
Cristal Externo de Baja Frecuencia	1001
Oscilador RC Externo	1000 – 0101
Oscilador interno RC Calibrado	0100 – 0001
Reloj Externo	0000

Tabla 2.4 Dispositivo de Reloj y sus Opciones de Selección

Cuando el CPU es habilitado desde un modo “Power-down” o “Power-save”, la fuente de reloj seleccionada es utilizada para medir el inicio, asegurando una operación de Oscilador estable antes del inicio de la ejecución de instrucciones. Cuando el CPU inicia desde un Reset, existe un retardo adicional permitiendo al encender buscar un nivel estable antes de comenzar la operación normal. El Oscilador Watchdog es empleado para temporizar parte de este tiempo de inicio. El número de ciclos del Oscilador WDT utilizados en cada caso se muestra en la siguiente tabla.

Tipo de Time*Out(Vcc=5V)	Tipo de Time*Out(Vcc=3V)	Número de Ciclos
4.1 ms	4.3 ms	4K (4,096)
65 ms	69 ms	64K (65,536)

Tabla 2.5 Número de ciclos del Oscilador WDT

2.4.16 Fuentes de Reloj por Defecto

El dispositivo vendido se suministra con CKSEL = "0001" y SUT = "10". El reloj de configuración por defecto es por lo tanto un oscilador interno RC de 1MHz con mayor tiempo de inicio. Esta configuración predeterminada asegura que todos los usuarios pueden hacer el ajuste deseado a la fuente de reloj usando un sistema interno o un programador paralelo.

2.4.17 Conexiones de la Fuente de Reloj

Los pines XTAL1 y XTAL2 son entradas y salidas, respectivamente, de un amplificador invertido el cual puede ser configurado como oscilador On-Chip, como se muestra en la Figura 2.12. Tanto un Cristal de Cuarzo o un resonador cerámico puede ser empleado. Además C1 y C2 deberían ser siempre iguales para ambos Cristal o Resonador. El valor óptimo de los capacitores depende del cristal y del resonador utilizado, la cantidad de pérdida de Capacitancia, y del ruido electromagnético del ambiente Para resonadores cerámicos, el valor del capacitor viene dado por el fabricante.

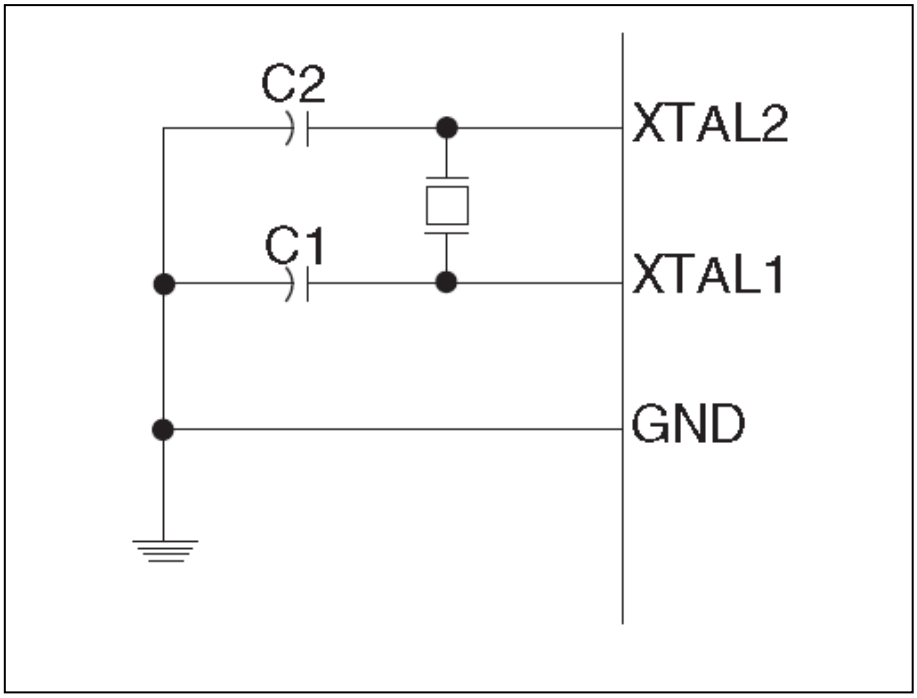


Figura 2.12 Conexión del Cristal Oscilador

El Oscilador puede operar en tres modos diferentes, cada uno optimizado para un rango de frecuencias específico. El modo de operación es seleccionado por los bits fusibles CKSEL 3:1, como se muestra en la siguiente tabla.

CKOPT	CKSEL 3:1	Rango de Frecuencia (MHz)	Rango recomendado para capacitores C1 y C2 en uso con cristales (pF)
1	101	0.4 – 0.9	--
1	110	0.9 – 3.0	12 - 22
1	111	3.0 – 8.0	12 – 22
1	101,110,111	1.0 ≤	12 – 22

Tabla 2.6 Modos de Operación del Cristal Oscilador

El fusible CKSEL0 junto con los fusibles SUT 1:0 permiten seleccionar el tiempo de inicio como se muestra a continuación.

CKSEL0	SUT 1:0	Tiempo de inicio desde el modo Power-down y Power-save	Retardo Adicional desde el Reset (Vcc=5V)	Uso Recomendado
0	00	258 CK	4.1 ms	Resonador Cerámico, encendido rápido
0	01	258 CK	65 ms	Resonador Cerámico, encendido lento
0	10	1K CK	--	Resonador Cerámico, BOD habilitado
0	11	1K CK	4.1 ms	Resonador Cerámico, encendido rápido
1	00	1K CK	65 ms	Resonador Cerámico, encendido lento
1	01	16K CK	--	Cristal Oscilador, BOD habilitado
1	10	16K CK	4.1 ms	Cristal Oscilador, encendido rápido
1	11	16K CK	65 ms	Cristal Oscilador, encendido lento

Tabla 2.7 Tiempo de inicio para selección de Cristal Oscilador

2.4.18 Calibrar el Oscilador Interno RC

Por defecto, el Oscilador Interno RC proporciona un reloj aproximado de 8MHz. Junto con el voltaje y temperatura nominal, este reloj puede ser calibrado con mucha precisión por el usuario.

La calibración del Oscilador RC interno provee un arreglo de 1.0, 2.0, 4.0 u 8.0 MHz para el reloj. Todas las frecuencias son valores nominales a 5V y 25°C. Este reloj puede ser seleccionado como el reloj del sistema por la programación de los fusibles

CKSEL. En caso de ser seleccionado este trabajará sin componentes externos. El fusible CKOPT puede ser siempre programado utilizando esta opción de reloj. Durante el Reset, por hardware se carga el byte de calibración para 1.0MHz dentro del registro OSCCAL, siendo calibrado automáticamente el Oscilador RC. A 5V, 25°C y 1.0MHz de frecuencia de Oscilación seleccionada, esta calibración da una frecuencia con una tolerancia de ±3% de la frecuencia nominal. Cuando el oscilador es utilizado como reloj del chip, el Oscilador Watchdog podrá ser empleado para el temporizador WDT y para el Reset.

CKSEL 3:0	Frecuencia Nominal (MHz)
0001	1.0
0010	2.0
0011	4.0
0100	8.0

Tabla 2.8 Modos de Operación para la calibración del Oscilador interno RC

Cuando este Oscilador es seleccionado, el tiempo de inicio es determinado por los fusibles “SUT” como se muestra en la siguiente tabla. XTAL1 y XTAL2 podrían estar desconectados.

SUT 1:0	Tiempo de inicio desde el modo Power-down y Power-save	Retardo Adicional desde el Reset (Vcc=5V)	Uso Recomendado
00	6 CK	--	BOD habilitado
01	6 CK	4.1 ms	Encendido rápido
10	6 CK	65 ms	Encendido Lento
11	Reservado		

Tabla 2.9 Tiempo de inicio para calibración de Oscilador interno RC

2.5 Reset y Control del Sistema

2.5.1 Reset del AVR

Durante el reset, todos los registros de E/S retornan a su valor inicial, y el programa empieza a ejecutarse desde el Vector de Reset. La instrucción localizada en el Vector de Reset debe ser un JMP – Salto Absoluto – instrucción para reiniciar la rutina manualmente. Si el programa nunca habilita una fuente de interrupciones, el Vector de interrupciones no es usado, y el código de programa puede estar ubicado en esta localidad.

Esto también ocurre en caso de que, el Vector de Reset, está en la sección de aplicaciones mientras el Vector de Interrupciones está en la sección baja del programa (boot), o viceversa. El diagrama del circuito se muestra en la figura 3.8 la cual muestra el reset lógico.

Los puertos de E/S del AVR son reiniciados inmediatamente a su estado inicial cuando una fuente de reset está activa. Esto no requiere que fuente alguna de reloj esté corriendo. Después de resetear todas las fuentes que se encuentran inactivas, se efectúa un conteo de retardo, alargándose el reset interno.

Lo cual permite lograr una estabilidad antes de la operación normal de inicio. El periodo de tiempo para el conteo del retardo está definido por el usuario a través de los fusibles SUT y CKSEL.

2.5.2 Fuentes de Reset

El ATmega32A presenta cinco Fuentes de Reset:

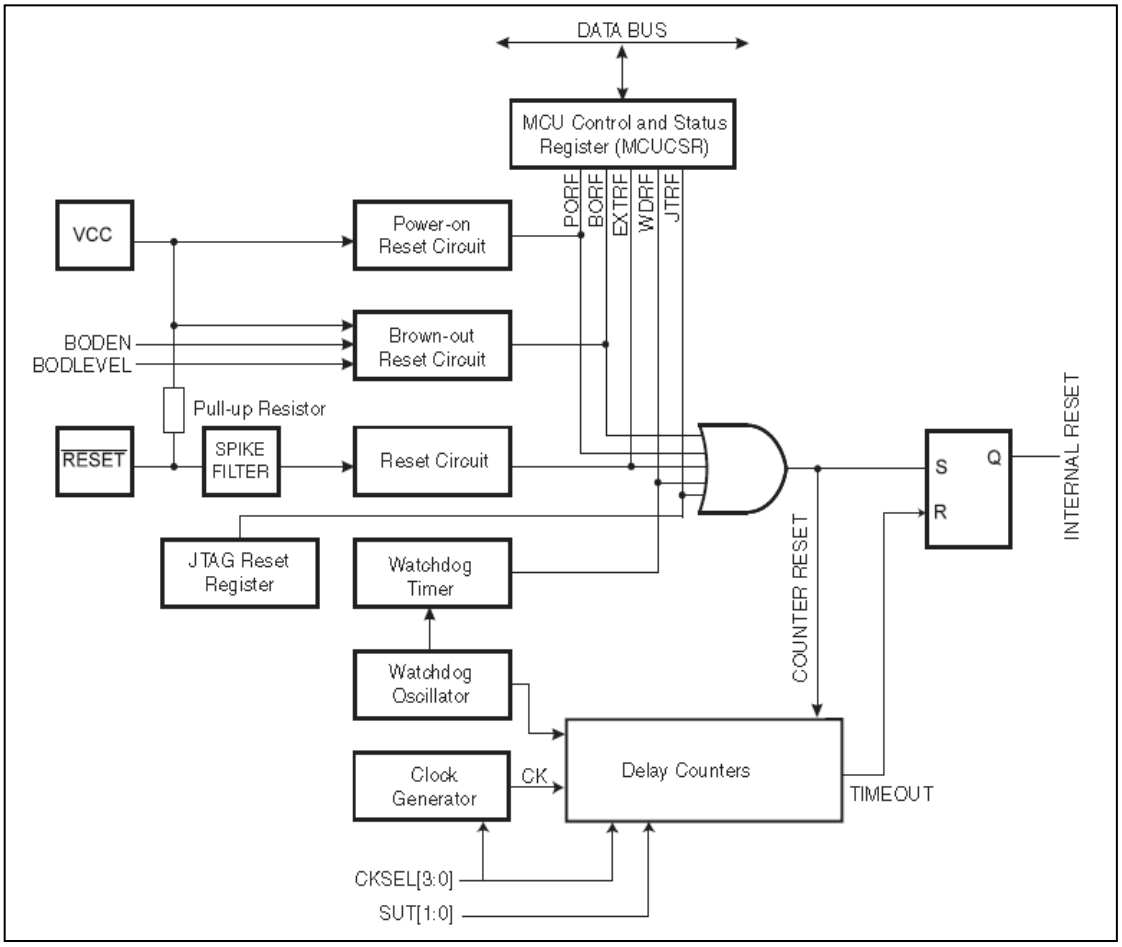


Figura 2.13 Bloques del Reset Lógico

- Reset de Encendido. El MCU es reseteado cuando el voltaje de alimentación está por debajo de cierto umbral de Voltaje de Encendido (VPOT).

- Reset Externo. El MCU es reseteado cuando un nivel bajo se presenta en el pin de RESET, el cual es más largo que la duración del mínimo pulso.

- Reset del Watchdog. El MCU es reseteado cuando el período de Tiempo del Watchdog termina y el mismo se encuentra habilitado.

- Reset de Apagado. El MCU es reseteado cuando el voltaje de Alimentación Vcc, está por debajo del umbral de Reset de Apagado (VBOT) y a la vez el detector de Apagado se encuentra habilitado.

- Reset del JTAG. El MCU es reseteado si existe un uno lógico en el registro del Reset. El sistema JTAG se encarga de ubicar este uno en el registro del Reset.

2.5.2.1 Reset de Encendido

Un pulso de Reset de Encendido (POR) es generado por un circuito de detección interno del chip. El circuito del POR está activado cuando el VCC está por debajo del nivel de detección. El circuito del POR puede ser empleado para disparar el reset, así como para detectar una falla en el voltaje de alimentación.

El circuito de Reset de Encendido asegura que el dispositivo sea reseteado desde el arranque. Al buscar alcanzar el voltaje umbral de Encendido se invoca un contador de retardo, el cual determina cuanto tiempo el dispositivo debe estar en Reset después del aumento del nivel de Vcc. La señal de Reset es activada nuevamente, sin ningún retardo, cuando el nivel de Vcc se decrementa por debajo del nivel de detección asignado.

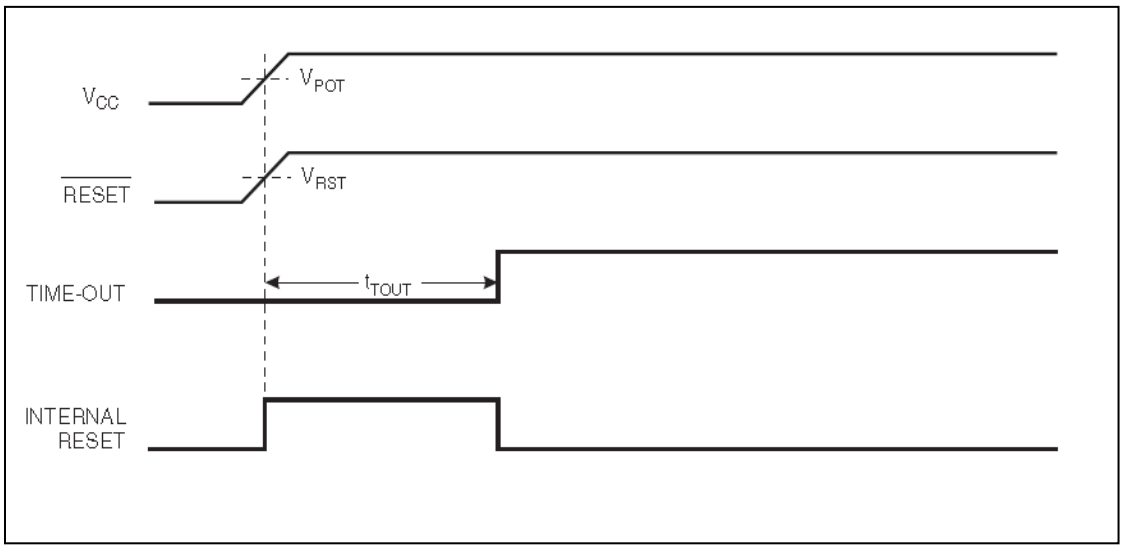


Figura 2.14 Inicio del MCU el reset tiende al nivel de Vcc

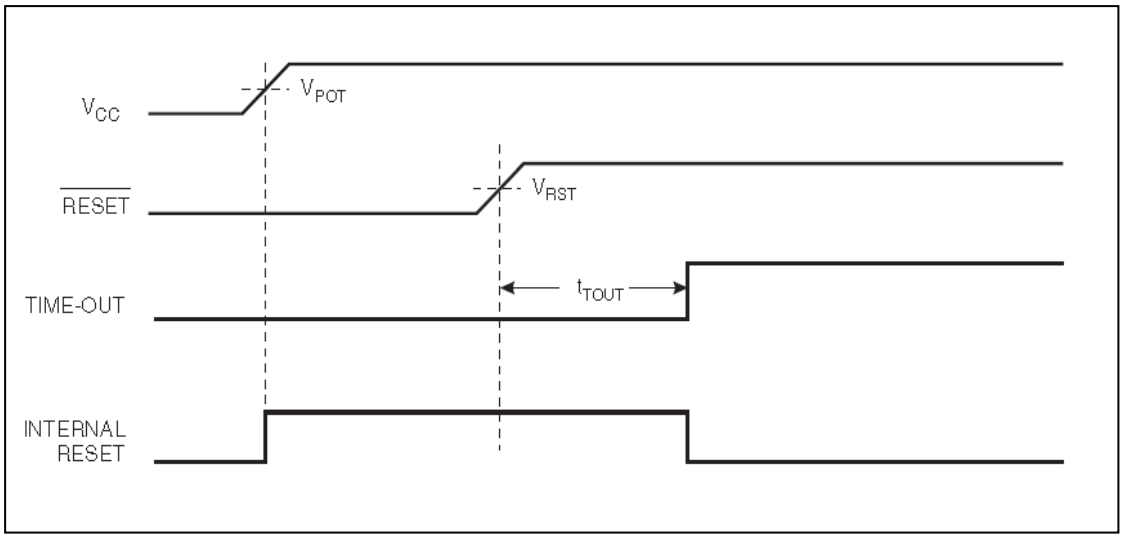


Figura 2.15 Inicio del MCU el reset es Extendido externamente

2.5.2.2 Reset Externo

Un reset externo es generado por un nivel bajo en el pin de RESET. Los pulsos de Reset más largos que el mínimo ancho del pulso generan un reset, siempre que el reloj no esté corriendo. Pulsos cortos no son garantía para generar un reset. Cuando la señal aplicada alcanza el voltaje umbral de Reset – VRST – en su flanco positivo, el

contador de retardos inicia al MCU después de que el periodo de tiempo –tout– ha finalizado.

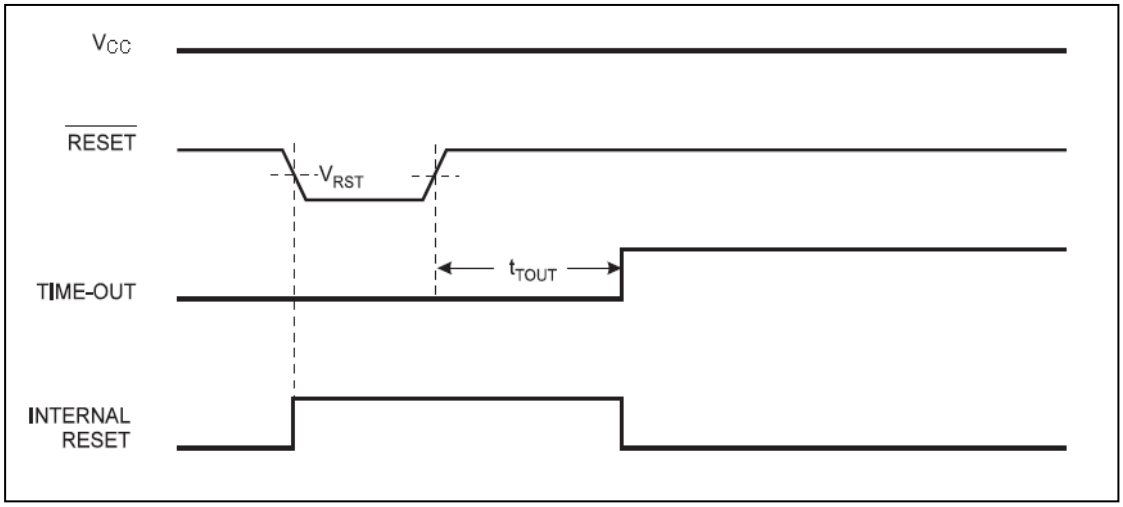


Figura 2.16 Reset Externo en ejecución

2.5.2.3 Detección de Apagado

El ATmega32A tiene en su interior un circuito detector de Apagado (Brown-Out Detector) para monitorear el nivel de Vcc durante su operación mediante la comparación con un cierto nivel de disparo.

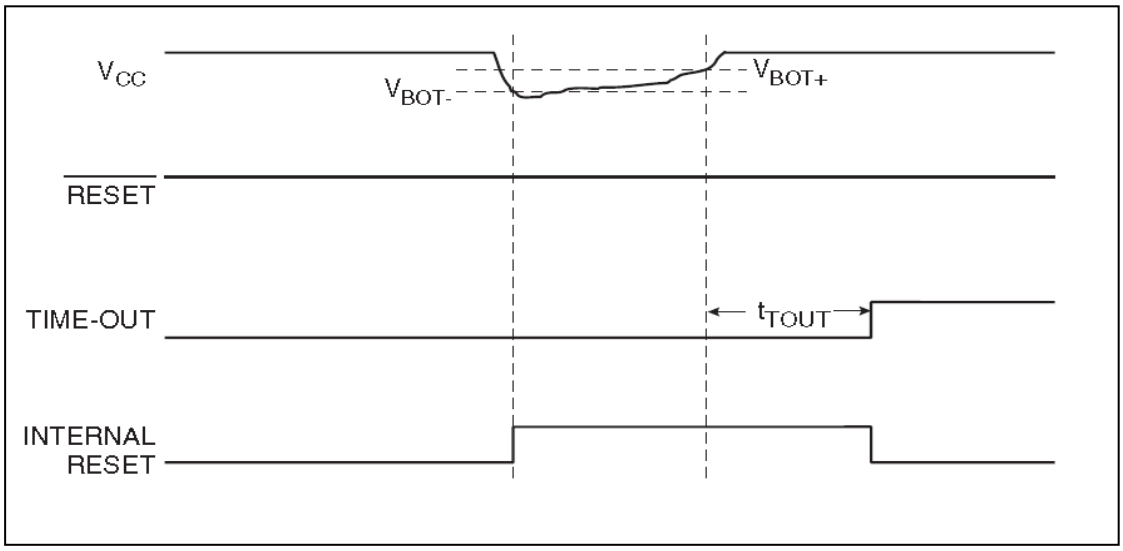


Figura 2.17 Brown-Out Reset en ejecución

El nivel de disparo para el detector de Apagado puede ser seleccionado por el fusible BODLEVEL., el cual será de 2.7V (BODLEVEL no programado), o 4.0V (BODLEVEL programado). El nivel de disparo presenta una histéresis para asegurar la detección del Apagado. La histéresis en el nivel de detección puede ser interpretada como VBOT+ = VBOT + VHYST/2 y VBOT- = VBOT - VHYST/2. El circuito BOD puede ser habilitado/deshabilitado por el fusible BODEN. Cuando el BOD está habilitado (BODEN programado), y Vcc decrementa hasta un valor por debajo del nivel de disparo, el Reset de Apagado es inmediatamente activado. Cuando Vcc se incrementa sobre el nivel de disparo, el contador de retardo inicia el MCU después de que ha finalizado el período de tiempo –tout–.

2.5.2.4 Reset del Watchdog

Cuando finaliza el tiempo del Watchdog, este genera un pequeño pulso de reset con un ciclo de duración de un CK. En el flanco de bajada de este pulso, el temporizador de retardo empieza el conteo del período de espera del tout.

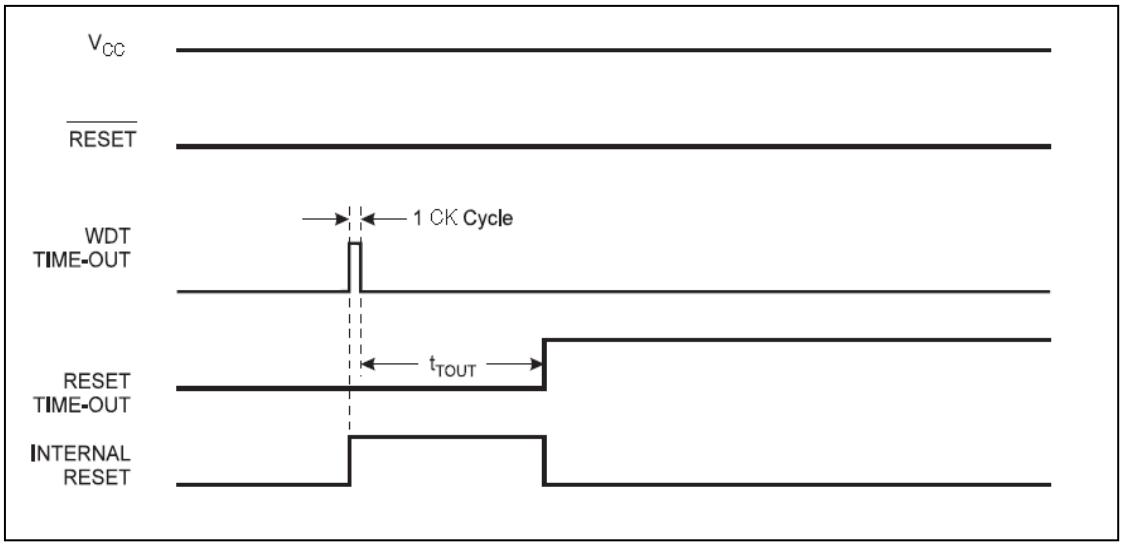


Figura 2.18 Reset del WatchDog en ejecución

2.6 Puertos de E/S

Los puertos de E/S son un conjunto de líneas (pines) programables como entrada ó salida que dispone el microcontrolador para comunicarse con el mundo exterior. El microcontrolador ATmega32A, presenta 4 puertos de E/S (Puertos A, B, C D). Todos los pines de cada puerto son programables como entrada o salida de datos configurando el registro asociado respectivo.

Cuando se programa el funcionamiento de un puerto se debe habilitar o deshabilitar las resistencias pull-up internas. Cada pin del puerto tiene independiente su resistencia pull-up como una resistencia invariante hacia la fuente de voltaje, además presenta diodos de protección, uno conectado a Vcc y el otro conectado a GND.

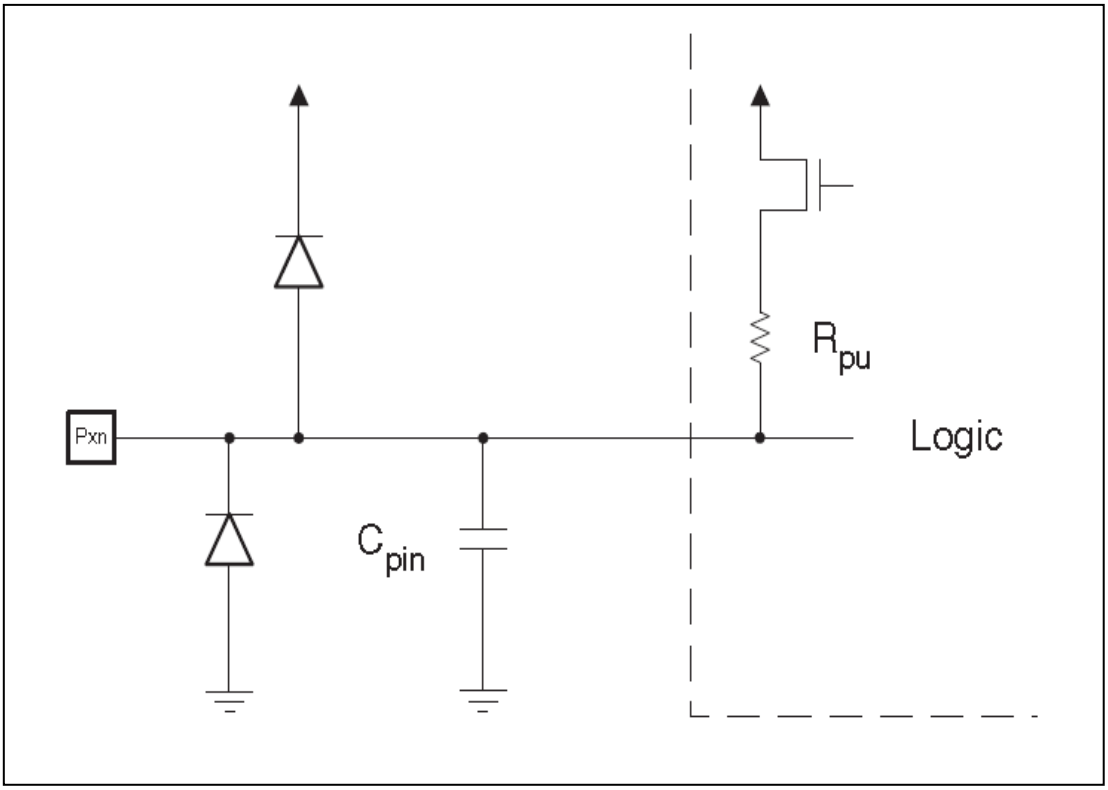


Figura 2.19 Diagrama Equivalente de un pin de E/S

En resumen, para cada puerto del microcontrolador (puertos B, C y D) existen tres registros de E/S que permiten configurar cada pin del puerto como entrada o salida, enviar datos a los pines configurados como salidas, y recibir datos de los pines configurados como entradas. Estos registros son: Registro de Direcciones de Datos – DDRx, Registro de Datos – PORTx, y el Puerto de Entrada de Pin – PINx,

En el párrafo anterior, "x" puede ser A, B, C ó D. Es decir, si nos referimos al puerto B, los registros son DDRB, PORTB y PINB. Los registros PINx son de sólo lectura, mientras que los registros PORTx y DDRx son de lectura/escritura. Adicionalmente, el bit "Pull-up Disable" – PUD, en el registro SFIOR inhabilita la función Pull-up para todos los pines de todos los puertos cuando es puesto a nivel alto. Cada pin del puerto de E/S, está asociado a 3 registros DDxn, PORTxn, y PINxn, cuyas direcciones en el espacio de memoria están indicados por los Registros DDRx E/S, PORTx E/S y PINx E/S..

El bit DDxn del Registro DDRx, establece la dirección de este pin. Si DDxn es escrito con "1" lógico, Pxn es configurado como un pin de salida. Si en DDxn es escrito un "0", Pxn es configurado como un pin de entrada. Si el pin PORTxn está configurado como un pin de entrada y es escrito un "1" lógico, entonces la resistencia pull-up está activada. En cambio si en PORTxn está configurado como un pin de salida ó es escrito con un "0" lógico, la resistencia pull-up está desactivada. Si el pin PORTxn está configurado como un pin de salida y es escrito con un "1" lógico, el pin toma un valor alto, en cambio si es escrito con "0" lógico el pin toma un valor bajo(cero lógico).

2.7 Protocolo I2C en MCU AVR

El IIC (Inter-Integrated Circuit) es un bus de interface de conexión incorporado en muchos dispositivos tales como sensores, RTC (por sus siglas en inglés Real Time Clock), y memorias EEPROM. El IIC es también conocido como I2C (I2C) en la literatura técnica. En esta sección se examinará los pines relacionados al bus I2C, y se centrará en el protocolo y la terminología asociada.

2.8 Bus I2C - TWI

Se trata de un protocolo serie desarrollado por Philips Semiconductors usado por muchos integrados para comunicarse entre ellos, para su funcionamiento requiere sólo dos líneas, una de reloj (SCL) y otra de datos (SDA) junto a dos resistencias de pull-up con cada una de estas líneas. Es un protocolo maestro esclavo en el que el maestro inicia / termina la comunicación y debe generar una señal de reloj (SCL), la línea de datos (SDA) es bidireccional (el maestro puede mandar o recibir), por lo general sólo suele haber un maestro (el microcontrolador) aunque el protocolo soporta más de uno. Todos los circuitos integrados conectados a este bus tienen una dirección física distinta de la de los demás. Transmisiones de 8 bits en serie pueden ser realizadas a 100 kbps en modo normal o estándar, 400 kbps en modo rápido y 3.4 Mbps en modo de Alta Velocidad.

En la familia ATMEL-AVR el protocolo I2C, lo encontramos con el nombre de TWI (por sus siglas en inglés Two Wire Interface), el cual nos permite conectar hasta 128

integrados al bus (limitado por la capacitancia del bus 400 pF) usando sólo dos líneas y añadiendo unas resistencias de pull-up.

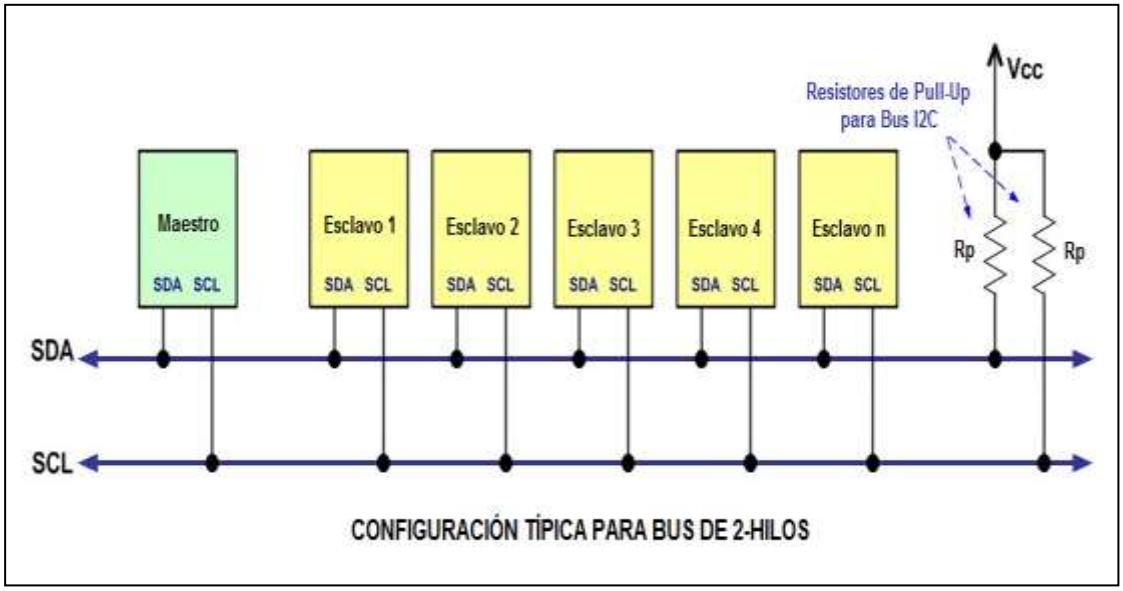


Figura 2.20 Conexión general para Bus I2C

2.8.1 Interface Serial TWI - 2 Hilos (2-wire)

Descripción:

- Interfase de comunicación flexible, solo necesita dos líneas del bus.
- Soporte para Operación como Maestro o Esclavo
- Dispositivos pueden operar como Transmisor o Receptor
- 7- bits de dirección permiten hasta 128 direcciones distintas de esclavo.
- Soporte arbitrario multi-maestro

- Velocidad de transferencia de datos superior a 400 KHz.

- Velocidad de Respuesta limitada por Salida de Drivers

- Circuitería de supresión de ruido y picos sobre las líneas del bus.

- Dirección de Esclavo Totalmente Programable con Soporte de Llamada General

- Reconocimiento de dirección al despertar el AVR del modo Sleep.

2.8.2 Interconexión Eléctrica

Ambas líneas del bus son conectadas a la fuente de voltaje positiva a través de las resistencias pull up. Los controladores del bus de todos los dispositivos TWI son de colector-abierto. Esto implementa una función AND con la cual es esencial la operación de la interfase. Un nivel bajo en la línea del bus TWI es generada cuando uno o más dispositivos de salida TWI son cero. Las salidas de todos los dispositivos conectados al bus deben ser de colector-abierto, por lo que cualquier integrado puede mantener las líneas a nivel bajo activando sus salidas.

Note que todos los dispositivos AVR conectados al bus TWI deben ser accionados con el fin de permitir cualquier operación del bus. El número de dispositivos que pueden ser conectados es solo limitado por el límite de capacitancia en el bus de 400 pF y los 7-bits de espacio para dirección de esclavo. Dos grupos diferentes de especificaciones están presentes aquí: una relevante para velocidades del bus por debajo de 100 Khz., y una válida para velocidades del bus superiores a 400 KHz.

2.8.3 Módulo TWI en el AVR.

Está compuesto por varios sub-módulos:

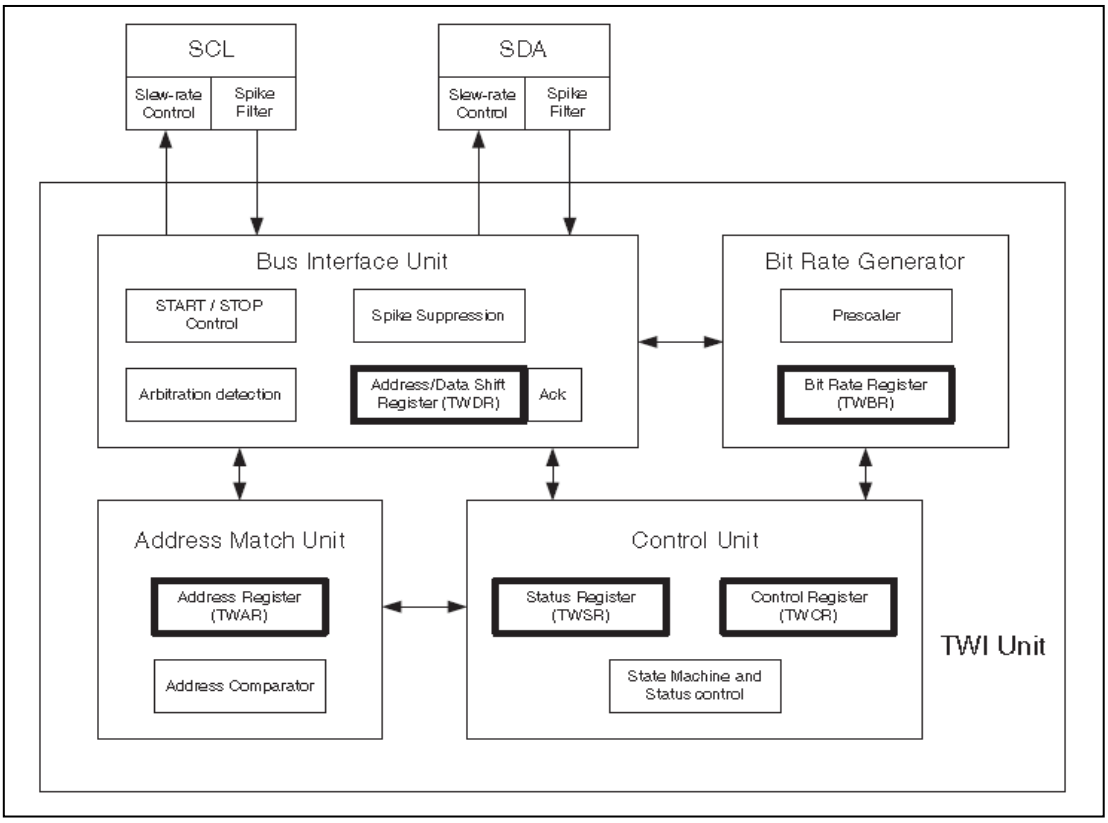


Figura 2.21 Descripción General del módulo TWI

2.8.4 Pines SCL y SDA

Estos son los pines de interfaz del módulo AVR-TWI con el resto de sistemas del MCU. La salida de los controladores presenta un limitador para la velocidad de respuesta con el fin de ajustarse a las especificaciones del protocolo TWI. Las etapas de entrada contienen supresores de pico unitarios permitiendo remover picos menores a 50 ns. Cabe notar que las resistencias de pull-up internas en los chips AVR pueden ser habilitadas por la configuración de los bits del Puerto correspondiente a los pines

SCL y SDA. La activación de las resistencias de pull-up internas del puerto puede evitar en algunas ocasiones tener que colocar las resistencias de pull-up externas.

2.8.5 Unidad Generadora de Tasa de Bits

Esta unidad controla el periodo de la señal SCL cuando se encuentra operando en modo Maestro. El periodo SCL es controlado por la configuración en el bit TWI del registro de velocidad (TWBR) y los bits del preescalador en el Registro de Estado del TWI (TWSR). La operación del esclavo no depende de la configuración de la Tas de bits o del prescalador, la frecuencia del reloj de la CPU del esclavo debe ser por lo menos 16 veces mayor que la frecuencia del reloj que generamos en la línea SCL, el esclavo siempre puede prolongar el tiempo de la parte baja del reloj si éste es demasiado rápido.

La frecuencia de reloj en este módulo es generada de acuerdo a la siguiente ecuación:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

TWBR = Valor del Registro Generador de Tasa de Bits

TWPS = Valor de los bits del Preescalador en el Registro de Estado TWI

Las resistencias del bus se deben elegir en función de esta frecuencia y de la capacitancia del bus.

2.8.6 Unidad de Interfaz del Bus

Esta unidad contiene el Dato y la Dirección del Registro de Desplazamiento (TWDR), un controlador de Inicio/Parada y un detector arbitrario de hardware. El TWDR contiene la dirección o los bytes de datos a ser transmitidos, o la dirección o los bytes recibidos. Adicionalmente a los 8-bits TWDR, la unidad del bus de interfaz también contiene el bit (N) ACK a ser transmitido o recibido. Este registro (N) ACK no es directamente accesible por el software aplicado. Sin embargo, cuando recibe, este puede ser puesto a uno o cero por manipulación del registro de control del TWI (TWCR). Mientras en el modo de transmisión, el valor de los bits (N)ACK recibidos puede ser determinado por el valor en el registro TWSR.

El controlador de Inicio/Parada es responsable de la generación y detección de las condiciones de Inicio, Inicio Repetido, y Parada. El controlador de Inicio/Parada también es capaz de detectar las condiciones de Inicio y Parada cuando el MCU AVR se encuentra en modo Sleep, permitiendo al MCU despertar si este es direccionado por un maestro.

Si el módulo TWI ha iniciado una transmisión como Maestro, la detección arbitraria por hardware continuamente monitorea la transmisión intentando determinar si la arbitración se encuentra en proceso. Si el módulo TWI ha perdido una arbitración, la unidad de control es informada. Permitiendo entonces tomar una acción correcta y generar los códigos de estado apropiados.

2.8.7 Unidad de Dirección

La Unidad de Dirección chequea si los bytes de dirección recibidos concuerdan con los 7-bits de dirección en el Registro de Dirección TWI (TWAR). Si el bit de Reconocimiento de Llamada General TWI habilitado (TWGCE) en el registro TWAR es puesto a uno, todos los bits de dirección siguientes serán también comparados a través de la dirección de Llamada General. Si una dirección coincide, la Unidad de Control es informada, permitiendo realizar una acción correctiva. El módulo TWI puede o no realizar un Acuse de Recibo (ACK) a esta dirección, dependiendo de la configuración en el Registro TWCR. La Unidad de Dirección es capaz de comparar direcciones aún cuando el MCU AVR se encuentre en modo Sleep, permitiendo al MCU despertar si este es direccionado por un maestro.

2.8.8 Unidad de Control

La unidad de control monitorea el bus TWI y genera respuestas correspondientemente a las configuraciones en el registro de control TWI (TWCR). Cuando un evento requiere la atención de las aplicaciones ocurridas en el bus del TWI, la bandera del Interrupción TWI (TWINT) es confirmada. En el siguiente ciclo de reloj, el Registro de estado TWI (TWSR) es actualizado con un código de estado que identifica el evento. El TWSR solo contiene información de estado relevante cuando la bandera de interrupción TWI es confirmada. Para el resto de casos, el TWSR contiene un código de estado especial indicando que no se dispone de información de estado relevante. Tan pronto como la bandera TWINT es habilitada, La línea SCL es puesta a nivel

bajo. Esto permite a las aplicaciones de software completar sus tareas antes de permitir que la transmisión TWI continúe.

La bandera TWINT es habilitada en las siguientes situaciones:

- Después que el TWI ha transmitido una condición de Inicio/Inicio Repetido.
- Después que el TWI ha transmitido SLA+R/W.
- Después que el TWI ha transmitido un byte de dirección.
- Después que el TWI ha perdido el arbitraje.
- Después que el TWI ha sido direccionado por su propia dirección de esclavo o Llamado general.
- Después que el TWI ha recibido un byte de datos.
- Después que ha sido recibido una Parada o Inicio Repetido mientras permanece direccionado como un esclavo.
- Cuando un error en el bus ha ocurrido debido a un Inicio ilegal o condición de Parada.

2.9 Utilizando el módulo TWI

El módulo AVR-TWI es orientado a bytes y basado en interrupciones. Las interrupciones son emitidas después de todos los eventos del bus, como la recepción de un byte o la transmisión de una condición de Inicio. Ya que el módulo TWI es basado en interrupciones el software de aplicación es libre de realizar otras operaciones durante una transferencia de bytes TWI. Cabe notar que el bit de Interrupción TWI habilitado (TWIE) en el registro TWCR junto con el bit de

Interrupciones Globales habilitado en el registro SREG permiten a la aplicación decidir habilitar o no la bandera TWINT para poder generar una solicitud de interrupción. Si el bit TWIE es deshabilitado, la aplicación debe sondear la bandera TWINT con el fin de detectar las acciones que ocurren en el bus TWI.

El funcionamiento del módulo TWI está basado en interrupciones que debe atender el microcontrolador, un ejemplo de transmisión siendo TWINT la bandera de interrupción, seguiría los siguientes pasos:

- El primer paso en toda transmisión es generar la señal de Inicio (Start), para ello se escribe un valor específico en el registro TWCR que indica al hardware generar la condición de Inicio, el bit TWINT del registro TWCR se deshabilita y luego el hardware comienza a generar la condición de Inicio.

- Cuando la señal de Inicio (Start) se ha transmitido el bit TWINT se pone a uno y el registro TWSR es actualizado con un código/valor que indica que se ha transmitido una condición de Inicio.

- La aplicación comprueba el registro TWSR para ver que ha sido enviado el Inicio (Start) y carga la dirección del esclavo (Slave Address) + el bit de Lectura/Escritura en el registro TWDR, luego escribe un código en TWCR que indica al hardware que debe enviar el valor contenido en TWDR. Se limpia el bit TWINT y se inicia la transmisión.

- Cuando se ha transmitido el paquete, el bit TWINT se pone a uno y TWSR es actualizado con un valor que indica que el byte se ha transmitido de forma correcta reflejando así el valor del bit de reconocimiento (ACK).

- El MCU AVR comprueba el registro TWSR y el valor del bit de reconocimiento (ACK) para ver lo que ha sucedido en el paso anterior, si todo ha ocurrido según lo esperado se carga el byte de datos en TWDR, luego se escribe en el registro TWCR el código para que el hardware lo envíe y se deshabilita el bit TWINT para iniciar la transmisión.

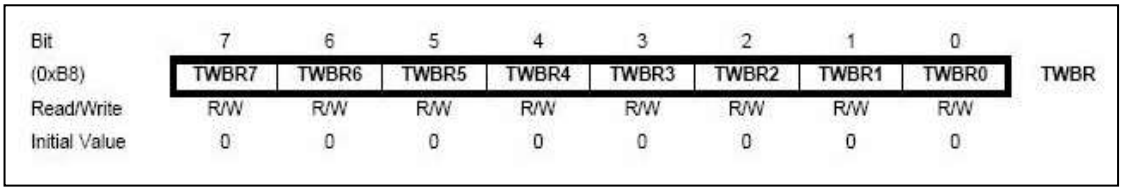
- Cuando se transmite el paquete de datos se habilita el bit TWINT y se actualiza el valor del registro TWSR con el resultado de la transmisión.

- La aplicación comprueba el registro TWSR para verificar lo que ha ocurrido y carga en el registro TWCR el código necesario para indicarle al hardware que envíe la condición de Parada (Stop), una vez que la aplicación deshabilita el bit TWINT. TWINT no es puesto a uno como resultado del envío de la condición de Stop.

2.9.1 Registros del módulo TWI

Los registros del MCU AVR necesarios para realizar una comunicación mediante el bus I2C/TWI son los siguientes:

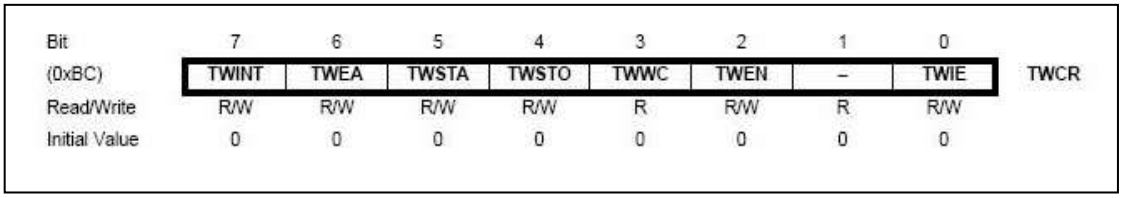
TWBR – Registro de Tasa de Bits TWI



Bit	7	6	5	4	3	2	1	0	
(0xB8)	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Estos bits son el valor del divisor de frecuencia utilizado para generar la señal de reloj SCL en modo maestro.

TWCR – Registro de Control TWI



Bit	7	6	5	4	3	2	1	0	
(0xBC)	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

El registro TWCR se utiliza para controlar la operación del bus TWI. Se emplea para habilitar el bus TWI, para iniciar el acceso de un Maestro aplicando una Condición de Inicio (Start) en el bus, para generar un Acuse de Recibo (ACK), para generar una Condición de Parada (Stop), y para la administración del bus en el proceso de escritura de bytes de datos.

- Bit 7, TWINT (TWI Interrupt Flag): se activa por hardware para indicar que el módulo TWI ha finalizado la tarea encomendada y está a la espera de instrucciones del programa saltando a la interrupción si está habilitada. Mientras este bit esté a 1 el reloj SCL está a nivel bajo. Cuando se ejecuta la interrupción TWINT no se limpia por lo que hay que hacerlo por software escribiendo un 1 en el bit, cuando se limpia el módulo TWI, comienza la

siguiente operación, determinada por los registros TWAR, TWSR y TWDR que deben ser modificados antes de limpiar el bit.

- Bit 6, TWEA: TWI (Enable Acknowledge Bit): se habilita el bit de "reconocimiento", si la dirección en el bus I2C del MCU o la dirección 0x00 (si está habilitada) ha sido recibida o a su vez si un byte de datos ha sido recibido en modo MR o SR. Escribiendo un 0 en este bit se desconecta el módulo TWI del MCU del bus.

- Bit 5, TWSTA: TWI (START Condition Bit): se pone a 1 cuando se quiere ser maestro y generar la señal de Start cuando el bus esté libre, se debe limpiar por software una vez que se ha transmitido la señal de Start.

- Bit 4, TWSTO: TWI (STOP Condition Bit): si se escribe un 1 estando en modo maestro se genera la señal de Stop, el bit se limpia por hardware cuando esto ocurre. En modo esclavo se utiliza para salir de un estado de error en el bus (0x00).

- Bit 3, TWWC: TWI (Write Collision Flag): se pone a 1 si se intenta escribir en TWDR estando TWINT a cero, este bit se limpia cuando se escribe en el registro TWDR estando TWINT a 1.

- Bit 2, TWEN: TWI (Enable Bit): enciende o apaga el módulo TWI del microcontrolador.

- Bit 1 – Res: Bits Reservados Estos bits son reservados y siempre son escritos a cero.

- Bit 0, TWIE: TWI (Interrupt Enable): cuando este bit es escrito a uno, y el bit-I en el registro SREG es habilitado, la respuesta de interrupción del TWI es activada mientras la bandera del TWINT esté en alto.

TWSR – Registro de Estado TWI

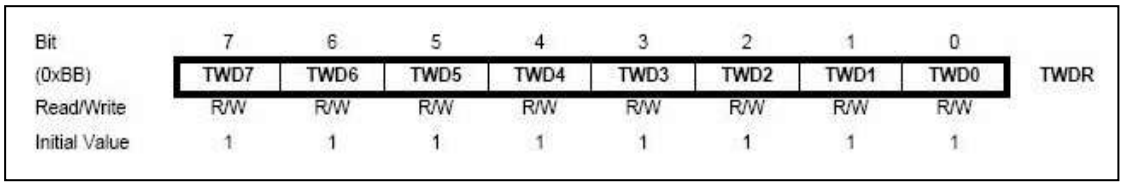
Bit	7	6	5	4	3	2	1	0	
(0xB9)	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

- Bits 7:3, TWS: TWI Status: indican el estado del módulo TWI según los distintos códigos anteriores.
- Bit 2 – Res: Bits Reservados: estos bits son reservados y siempre son escritos a cero.
- Bits 1:0, TWPS: (TWI Prescaler Bits): controlan el Preescalador.

TWPS1	TWPS0	Valor del Preescalador
0	0	1
0	1	4
1	0	16
1	1	64

Tabla 2.10 Tasa de bits del Preescalador

TWDR – Registro de Datos TWI

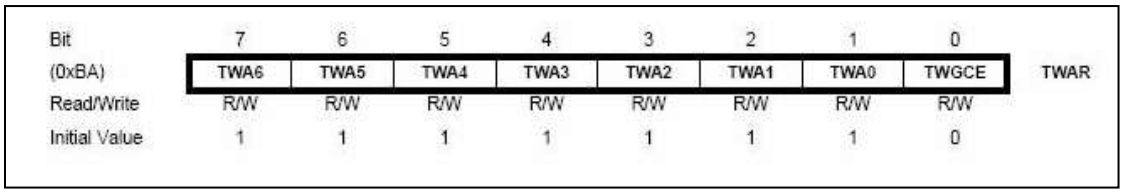


Bit	7	6	5	4	3	2	1	0	
(0xBB)	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

En modo de transmisión contiene el byte que se va a transmitir y en el modo de recepción contiene el último byte recibido.

- Bits 7:0 – TWD: Registro de Datos del TWI: estos ocho bits constituyen el siguiente byte de datos a ser transmitidos, o el último byte de datos recibidos en el Bus Serial de 2-hilos.

TWAR – Registro de Direcciones TWI (Esclavo)



Bit	7	6	5	4	3	2	1	0	
(0xBA)	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

El registro TWAR debe ser cargado con los 7-bits de dirección de esclavo, a lo cual el bus TWI responderá cuando el MCU es programado como un Esclavo Transmisor o Receptor. En sistemas Multimaestros, el registro TWAR debe ser configurado en modo Maestro los cuales pueden ser direccionados como esclavos por otros Maestros.

Los bits menos significativos del registro TWAR son utilizados para habilitar el reconocimiento de un Llamado General direccionado ($00). En este caso es un comparador de dirección asociado que busca en las direcciones de esclavos (o

Llamado general direccionado si está habilitado) en la recepción de direcciones seriales. Si existe una concordancia, un requerimiento de interrupción es generado.

- Bits 7:1, TWAM: (TWI Address Mask): indican la dirección del micro en el bus I2C.
- Bit 0, TWGCE: si está a 1 el micro responde a la dirección "global" 0x00.

Capítulo 3

Diseño e Implementación del Proyecto

3.1 Plataforma de Desarrollo

Para la correspondiente implementación de los ejercicios se ha hecho uso de varios elementos que nos ayudan a complementar el correcto funcionamiento planteado en cada caso, entre estos elementos mencionamos: Protoboard, Memorias EEPROM con interfaz I2C, etc.

3.2 Protoboard

3.2.1 Descripción

El protoboard en términos sencillos es una tabla que permite interconectar componentes electrónicos sin necesidad de soldarlos para fijarlos a una placa. Así, se puede experimentar de manera fácil y ágil a través del rápido armado y desarmado de

circuitos eléctricos. La lógica de operación del protoboard es muy sencilla, ésta es una tabla con orificios los cuales están conectados entre si en un orden coherente. Un protoboard comercial se presenta en la siguiente figura.

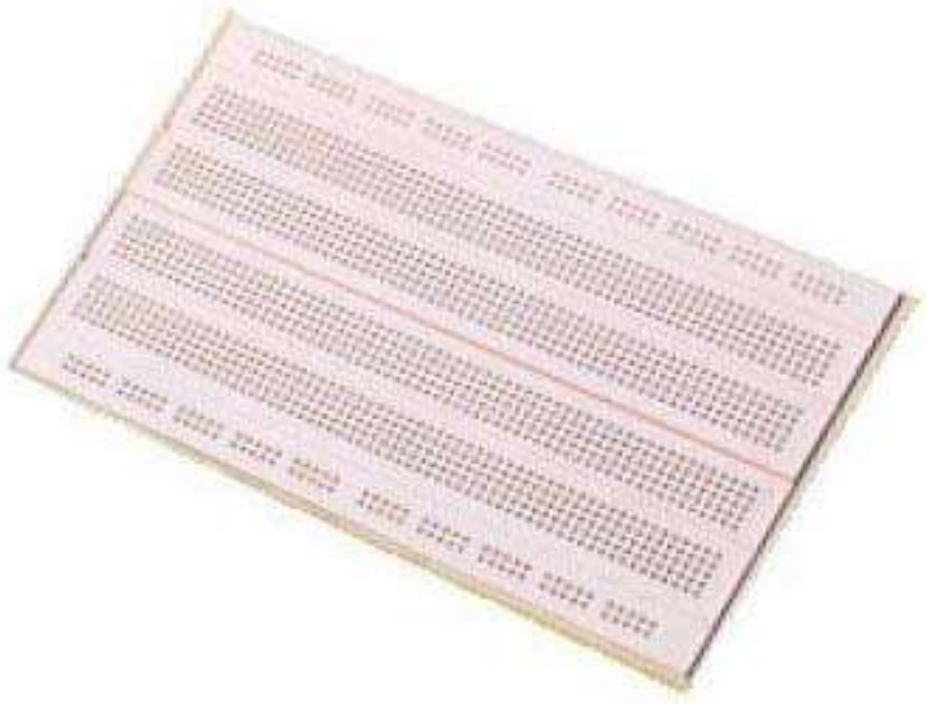


Figura 3.1 Aspecto General de un Protoboard

Si bien en la figura anterior se muestra su aspecto general, en la práctica el protoboard se puede ilustrar de acuerdo a la figura 3.2, en la cual se muestra una tabla con múltiples orificios los cuales se pueden ordenar, al igual que una matriz, en filas y columnas.

El protoboard se lo puede dividir en dos áreas principales que son los nodos y las pistas. Los nodos tienen conexión y por ende conducen a todo lo largo (aunque algunos fabricantes dividen estos nodos en dos partes). En los nodos se acostumbra a conectar la fuente de poder que usan los circuitos o las señales generalmente desde un equipo externo. Por su parte, las pistas proveen puntos de contacto para los pines o

terminales de los componentes que se colocan en el protoboard siguiendo el esquemático de cada circuito. Estos funcionan como mininodos y se usan para interconectar los puntos comunes de los circuitos que montas. Cuando no alcanzan los orificios disponibles, es posible realizar puentes de conexión, con un conductor desde la pista de interés a otra que se encuentre libre y continuar así con las conexiones.

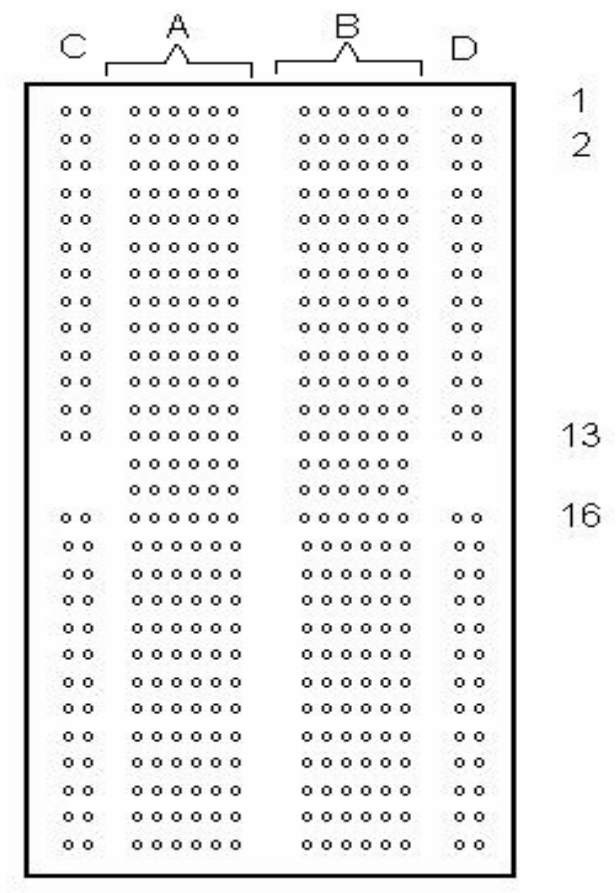


Figura 3.2 Esquema del protoboard

En particular el esquema ordenado como matriz muestra un protoboard de 28 filas y 16 columnas. Las columnas han sido concentradas en los grupos A, B, C y D. Cada fila del grupo A representa un nodo, al igual que cada fila del grupo B, es decir, si se conecta el terminal de algún elemento electrónico en el orificio (1,3), éste estará conectado directamente con el terminal de otro elemento electrónico que se conecta en el orificio (1,4). Además, cada columna del grupo C representa un nodo, al igual que

cada columna del grupo D. Las extensiones de las columnas de los grupos C y D están divididos en dos partes, desde la fila 1 a la 13, y desde la fila 16 a la 28, esto permite tener un mayor número de nodos.

En resumen, los distintos nodos quedan distribuidos dentro del protoboard según muestra la figura siguiente.

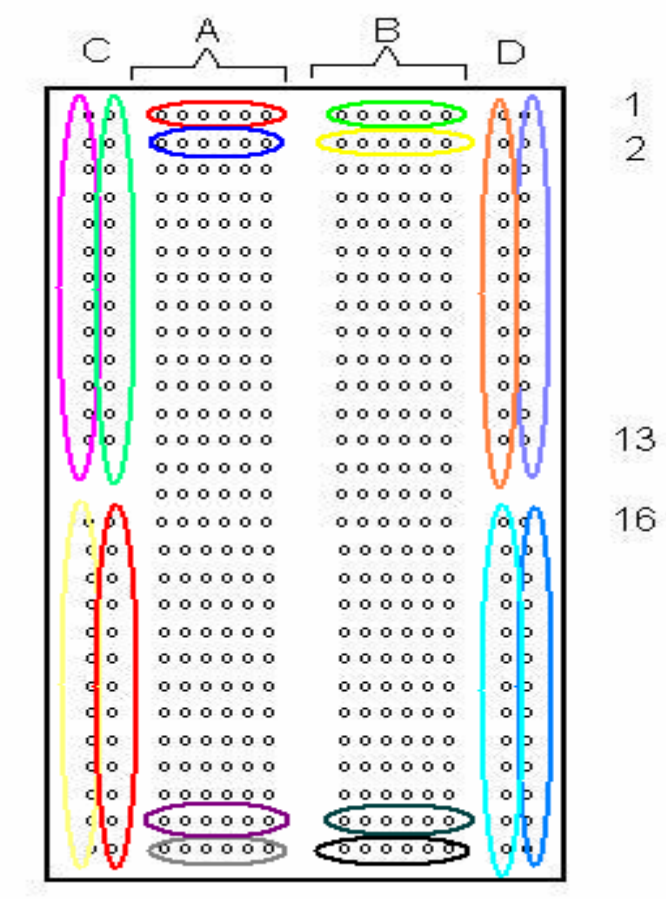


Figura 3.3 Ilustración de nodos de un protoboard

En la figura anterior se puede apreciar que el grupo A tiene 28 nodos, al igual que el grupo B. Además, los grupos C y D tienen 4 nodos cada uno. El total de nodos de este protoboard en particular es de 64 nodos. Por convención y comodidad, los grupos A y B se ocupan para interconexión de componentes en general, mientras que los nodos de los grupos C y D se utilizan para la alimentación de la tabla.

3.3 EEPROM - Memoria de Solo Lectura Eléctricamente Borrable y Programable – [22]

Este tipo de memorias tiene como característica principal, permitir almacenar y sobrescribir datos por medio de ciertos niveles de voltajes de operación presentes en los circuitos electrónicos, además de poder mantener dicha información por largo tiempo sin necesidad de una fuente de alimentación externa. En la actualidad es posible encontrar memorias EEPROM de interfaz serial, en las cuales ha sido posible reducir su control a unos cuantos pines, que son empleados como entrada o salida de datos en formato serial (1 ó 2 pines), habilitación (1 pin), reloj de sincronismo (1 pin), direccionamiento de dispositivo (3 pines), y por último los pines de alimentación del circuito (2 pines). Los datos y la dirección de las posiciones de memoria utilizarán únicamente uno o dos pines, dependiendo del tipo de comunicación utilizada (dos o tres hilos). La velocidad para transferencia de datos puede variar en el rango de 100 KHz hasta 600 MHz, en base al tipo de memoria y al protocolo de comunicación utilizado.

3.3.1 Memorias EEPROM seriales 24xxx

Las EEPROM seriales que se han utilizado pertenecen a la familia 24xxx. Hay varias compañías que fabrican muchos de estos modelos. En la tabla que se muestra a continuación se aprecia la diversidad de memorias disponibles (y todavía es posible hallar unas cuantas más, aunque menos usuales). Su nomenclatura indica que el número final de la identificación de cada dispositivo es la capacidad de la EEPROM en bits. Por ejemplo, la EEPROM 24xx64 tiene 64 Kbits = 8 Kbytes de memoria.

Dispositivo	Capacidad		Conexión en cascada
	Bits	Bytes	
24xx01	1 K	128	No
24xx02	2 K	256	No
24xx04	4 K	512	No
24xx08	8 K	1 K	No
24xx16	16 K	2 K	No
24xx32	32 K	4 K	Sí
24xx64	64 K	8 K	Sí
24xx128	128 K	16 K	Sí
24xx256	256 K	32 K	Sí
24xx512	512 K	64 K	Sí

Tabla 3.1 Dispositivos EEPROM de la familia 24XXX

También se puede notar que la capacidad de conexión en cascada divide a las EEPROM's en dos grupos: las que la tienen y las que no. Las que la tienen pueden reconfigurar su dirección de esclavo de modo que se puedan colgar varias EEPROM's de ese tipo en el mismo bus I2C. Por ejemplo, es posible tener hasta 8 EEPROM's 24xx256 para juntar un total de 128 Kbytes.

La división indicada coincide con el modo de acceso al contenido de las EEPROM's. Por ejemplo, la forma de leer o escribir un byte en una 24xx16 difiere ligeramente de la forma de hacerlo en una 24xx32. Además el significado de las letras XX, suele variar según el fabricante. En los modelos de Microchip dichas letras pueden ser AA, LC o FC y distinguen básicamente la velocidad de reloj y tensión de operación del dispositivo.

3.3.2 EEPROM serial 24xx128 [22]

Todo lo mencionado ahora es parte del datasheet. Sus principales características son: Los tres modelos 24AA128, 24LC128 y 24FC128 se diferencian únicamente por operar a diferente velocidad.

- Memoria de 128Kbits = 19Kbytes.
- Frecuencia máxima de reloj de 400Khz (Full Speed Mode).
- Tiempo de escritura máximo de 5ms.
- Retención de datos > 200 años.
- 1,000,000 de ciclos de lectura / escritura.
- Modo de escritura por página de 64 bytes.
- Conexión en cascada hasta de 8 dispositivos.
- Tensión de operación entre 2.5 V y 5.5 V.

3.3.2.1 Descripción de Pines

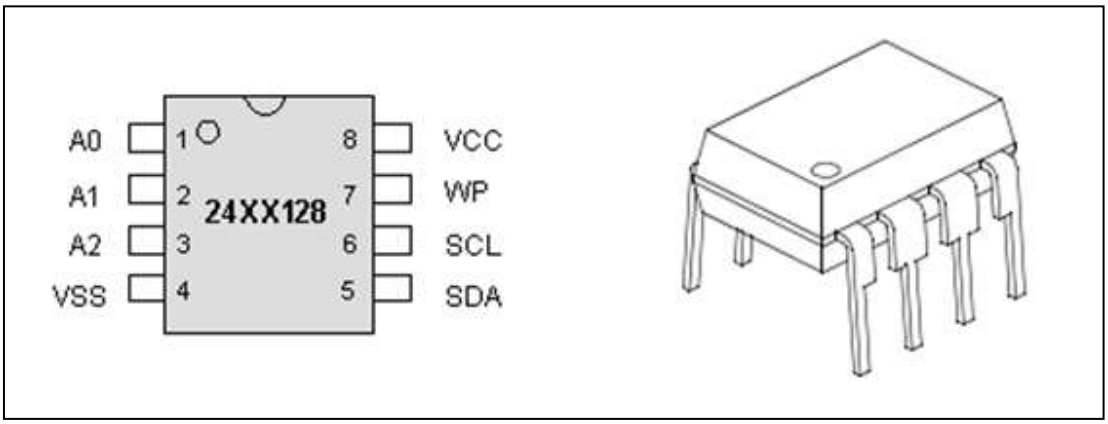


Figura 3.4 Esquema físico de EEPROM serial de la familia 24XX128

- A0, A1 y A2. Pines que establecen parte de la dirección de esclavo de este dispositivo.
- Vss y Vcc. Tierra y alimentación del dispositivo.
- SDA y SCL. Línea serial de datos y línea serial de reloj.
- El pin WP (Write Protection) o protección de escritura. Conectado a Vcc activa la protección de escritura, es decir, el contenido de la memoria podrá ser leído pero no escrito. Si WP se conecta a GND se podrá leer y escribir.

3.3.3 Dirección del dispositivo

Es necesario recordar que cada dispositivo esclavo conectado al bus I2C, debe estar identificado por una dirección de 7 bits. Pues bien, en la gran mayoría de los esclavos parte de esa dirección suele ser fija y la otra parte reconfigurable vía hardware. Por ejemplo, en las EEPROM's seriales de la familia 24xxx, la dirección de esclavo, en binario, es 1010xxx, siendo xxx la parte reconfigurable por los pines A2, A1 y A0 del dispositivo. Así se podrán formar ocho direcciones diferentes para conectar hasta 8 EEPROM's de este tipo. Como es conocido, la dirección de esclavo debe estar contenida en el byte de control de cada paquete de datos transmitido. El bit restante de dicho byte (R/W) indica si los siguientes bytes serán de lectura (R/W=1) o de escritura (R/W=0).

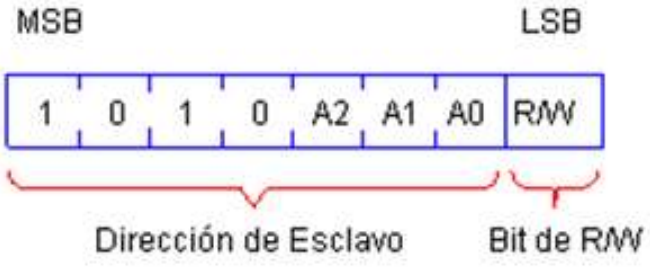


Figura 3.5 Byte de control (Dirección de esclavo + bit R/W)

3.3.4 Lectura y Escritura aleatoria de Bytes

Existen dos formas de realizar operaciones de lectura y escritura de datos en las EEPROM's 24xxx, una es byte por byte (acceso a un byte por cada paquete transmitido) y la otra es en bloques (varios bytes por cada paquete transmitido). En este apartado nos enfocamos a la primera forma, conocida como acceso aleatorio, porque se debe especificar la dirección de cada byte de dato accedido.

La locación de memoria a acceder depende de un registro interno llamado Puntero de memoria, el cual puede llegar a ser de 16 bits (2 bytes). Tras cada lectura el Puntero de memoria se incrementa en 1. Para las escrituras funciona ligeramente diferente.

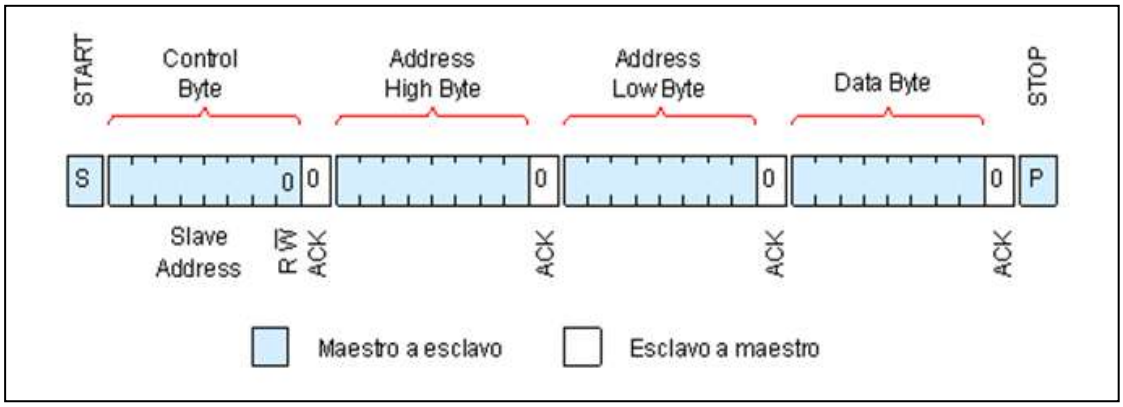


Figura 3.6 Secuencia de escritura de un byte en la EEPROM 24xx128

La figura de arriba indica que para escribir un Byte de Datos en la dirección Address de la memoria se debe seguir la siguiente secuencia:

- Enviar una condición de Start (iniciar transferencia).
- Enviar el byte de control para escritura (Slave address + Write).
- Enviar el byte alto de Address.

- Enviar el byte bajo de Address.
- Enviar el Byte de Datos.
- Enviar una Condición de STOP (cerrar transferencia).
- Tras la condición de STOP empieza el ciclo de escritura interno del dato enviado. Este ciclo dura a lo mucho 5 ms. Hay que poner un retardo.

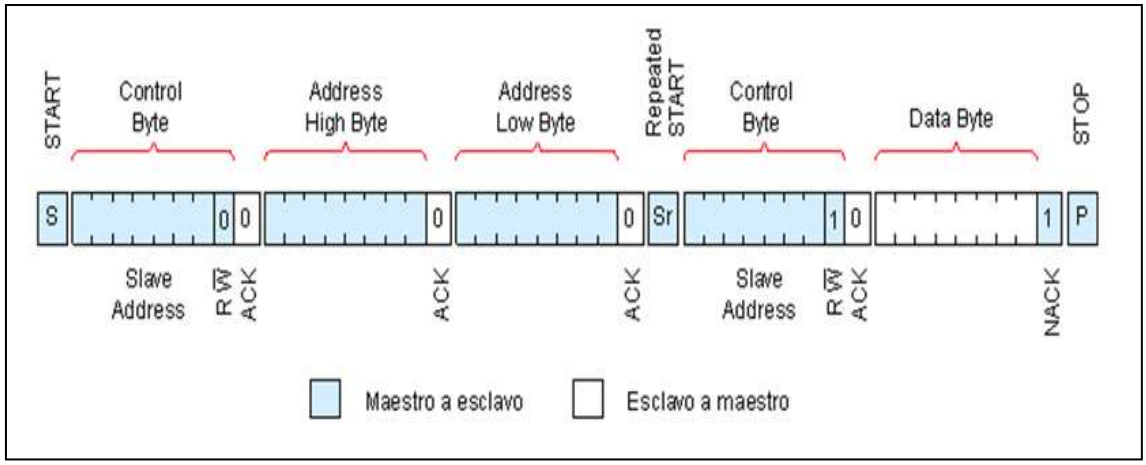


Figura 3.7 Secuencia de lectura de un byte en la EEPROM 24xx128

El esquema indica que para leer un Data byte de la dirección Address de la memoria se deben seguir los siguientes pasos:

- Enviar una condición de START.
- Enviar el byte de control para escritura (Slave address + Write).
- Enviar el byte bajo de Address.
- Enviar el byte alto de Address.
- Enviar una condición de START (llamada START repetido aunque sea lo mismo).
- Enviar el byte de control para lectura (Slave address + Read).
- Leer el byte de dato y devolver un NACK.
- Enviar una condición de STOP.

Notar que, aunque se vaya a leer un dato de la EEPROM, primero debemos especificar de que dirección, es decir, debemos escribir su dirección (en el Puntero de memoria). Por eso el primer byte de control es para escritura. Luego se vuelve a enviar el byte de control, esta vez para la lectura del dato en si.

3.4 Descripción de Ejercicios Propuestos

A continuación se detalla de manera resumida, cada uno de los ejercicios propuestos con el fin de aplicar los fundamentos teóricos establecidos en los capítulos anteriores acerca del bus I2C, y sus módulos componentes.

3.4.1 Ejercicio 1

Programación en Ensamblador del módulo TWI en Modo Maestro para Escritura

3.4.1.1Descripción

En este modo se transmite un cierto número de bytes de datos a un Esclavo Receptor (SR), configurando el octavo bit del primer byte R/W, para poder determinar la habilitación de Escritura de datos.

Para trabajar en modo Maestro, debe ser inicializado el bus TWI, transmitir una condición de Inicio (Start), enviar datos, y transmitir una condición de Parada (Stop).

3.4.1.2 Lista de Materiales

CANTIDAD	ITEM
1	MCU-AVR ATMega32
2	Capacitor Cerámico de 27pF.
1	Cristal Resonador de 8 Mhz.
2	Resistor de 4.7KΩ, 1/2 W.
1	Resistor de 1.5KΩ, 1/2 W.
1	Pulsador tipo botón N.A.
1	Potenciómetro regular de 10KΩ
1	Pantalla LCD, 16x2.
4	Memoria EEPROM 24LC32

Tabla 3.2 Lista de Materiales Transmisión en Ensamblador modo Maestro

3.4.2 Ejercicio 2

Programación en Ensamblador del módulo TWI en Modo Maestro para Lectura

3.4.2.1 Descripción

En este modo se recibe un cierto número de bytes de datos desde un Esclavo Receptor (SR), configurando el octavo bit del primer byte R/W, para poder determinar la habilitación de Lectura de datos.

Para trabajar en modo Maestro, debe ser inicializado el bus TWI, transmitir una condición de Inicio (Start), recibir datos, y transmitir una condición de Parada (Stop).

3.4.2.2 Lista de Materiales

CANTIDAD	ITEM
1	MCU-AVR ATMega32
2	Capacitor Cerámico de 27pF.
1	Cristal Resonador de 8 Mhz.
2	Resistor de 4.7KΩ, 1/2 W.
1	Resistor de 1.5KΩ, 1/2 W.
1	Pulsador tipo botón N.A.
1	Potenciómetro regular de 10KΩ
1	Pantalla LCD, 16x2.
4	Memoria EEPROM 24LC32

Tabla 3.3 Lista de Materiales Recepción en Ensamblador modo Maestro

3.4.3 Ejercicio 3

Programación en C del módulo TWI en Modo Maestro para Escritura

3.4.3.1 Descripción

Versión en lenguaje C del modo de Transmisión Maestro para Escritura por bus I2C. En este modo se recibe un cierto número de bytes de datos desde un Esclavo Receptor (SR), configurando el octavo bit del primer byte R/W, para poder determinar la habilitación de Lectura de datos. Para trabajar en modo Maestro, debe ser inicializado el bus TWI, transmitir una condición de Inicio (Start), enviar datos, y transmitir una condición de Parada (Stop).

3.4.3.2 Lista de Materiales

CANTIDAD	ITEM
1	MCU-AVR ATMega32
2	Capacitor Cerámico de 27pF.
1	Cristal Resonador de 8 Mhz.
2	Resistor de 4.7KΩ, 1/2 W.
1	Resistor de 1.5KΩ, 1/2 W.
1	Pulsador tipo botón N.A.
1	Potenciómetro regular de 10KΩ
1	Pantalla LCD, 16x2.
4	Memoria EEPROM 24LC32

Tabla 3.4 Lista de Materiales Transmisión en lenguaje C modo Maestro

3.4.4 Ejercicio 4

Programación en C del módulo TWI en Modo Maestro para Lectura

3.4.4.1 Descripción

Versión en lenguaje C del modo de Transmisión Maestro para Lectura por bus I2C. En este modo se recibe un cierto número de bytes de datos desde un Esclavo Receptor (SR), configurando el octavo bit del primer byte R/W, para poder determinar la habilitación de Lectura de datos. Para trabajar en modo Maestro, debe ser inicializado el bus TWI, transmitir una condición de Inicio (Start), recibir datos, y transmitir una condición de Parada (Stop).

3.4.4.2 Lista de Materiales

CANTIDAD	ITEM
1	MCU-AVR ATMega32
2	Capacitor Cerámico de 27pF.
1	Cristal Resonador de 8 Mhz.
2	Resistor de 4.7KΩ, 1/2 W.
1	Resistor de 1.5KΩ, 1/2 W.
1	Pulsador tipo botón N.A.
1	Potenciómetro regular de 10KΩ
1	Pantalla LCD, 16x2.
4	Memoria EEPROM 24LC32

Tabla 3.5 Lista de Materiales Recepción en lenguaje C modo Maestro

3.4.5 Ejercicio 5

Programación en Ensamblador del módulo TWI en Modo Maestro para Escritura / Lectura

3.4.5.1 Descripción

En este modo se transmite un cierto número de bytes de datos a un Esclavo Receptor (SR), dependiendo del octavo bit del primer byte R/W se determina si debe estar habilitado para Lectura o Escritura de datos.

Para trabajar en modo Maestro, debe ser inicializado el bus TWI, transmitir una condición de Inicio (Start), enviar o recibir datos, y transmitir una condición de Parada (Stop).

3.4.5.2 Lista de Materiales

CANTIDAD	ITEM
1	MCU-AVR ATMega32
2	Capacitor Cerámico de 27pF.
1	Cristal Resonador de 8 Mhz.
2	Resistor de 4.7KΩ, 1/2 W.
1	Resistor de 1.5KΩ, 1/2 W.
1	Pulsador tipo botón N.A.
1	Potenciómetro regular de 10KΩ
1	Pantalla LCD, 16x2.
4	Memoria EEPROM 24LC32

Tabla 3.6 Lista de Materiales Transmisión/Recepción en Ensamblador modo Maestro

3.4.6 Ejercicio 6

Programación en C del módulo TWI en Modo Maestro para Escritura / Lectura

3.4.6.1 Descripción

Versión en lenguaje C del modo de Transmisión Maestro para Lectura / Escritura por bus I2C. En este modo se transmite un cierto número de bytes de datos a un Esclavo Receptor (SR), dependiendo del octavo bit del primer byte R/W se determina si debe estar habilitado para Lectura o Escritura de datos.

Para trabajar en modo Maestro, debe ser inicializado el bus TWI, transmitir una condición de Inicio (Start), enviar o recibir datos, y transmitir una condición de Parada (Stop).

3.4.6.2 Lista de Materiales

CANTIDAD	ITEM
1	MCU-AVR ATMega32
2	Capacitor Cerámico de 27pF.
1	Cristal Resonador de 8 Mhz.
2	Resistor de 4.7KΩ, 1/2 W.
1	Resistor de 1.5KΩ, 1/2 W.
1	Pulsador tipo botón N.A.
1	Potenciómetro regular de 10KΩ
1	Pantalla LCD, 16x2.
4	Memoria EEPROM 24LC32

Tabla 3.7 Lista de Materiales Emisión/Recepción en lenguaje C modo Maestro

3.4.7 Ejercicio 7

Interfaz I2C, en Lenguaje C, para Reloj en Tiempo Real DS1307

3.4.7.1 Descripción

Programa en lenguaje C, que hace uso del modo de Transmisión Maestro para Lectura / Escritura de datos por bus I2C. En este modo se configura e inicializa el Esclavo

Receptor (RTC DS1307), con el fin de obtener un reloj programado y continuo en tiempo real.

3.4.7.2 Lista de Materiales

CANTIDAD	ITEM
1	MCU-AVR ATMega32
2	Capacitor Cerámico de 27pF.
1	Cristal Resonador de 8 Mhz.
2	Resistor de 4.7KΩ, 1/2 W.
1	Resistor de 1.5KΩ, 1/2 W.
1	Pulsador tipo botón N.A.
1	Potenciómetro regular de 10KΩ
1	Pantalla LCD, 16x2.
1	RTC DS1307

Tabla 3.8 Lista de Materiales Emisión/Recepción en lenguaje C RTC-DS1307

Capítulo 4

Desarrollo y Simulación del Proyecto

4.1 Introducción

En este capítulo detallamos exhaustivamente el proceso de desarrollo y simulación de los ejercicios propuestos, con el fin de explicar de manera clara y transparente la comunicación de dispositivos a través del Protocolo I2C, haciendo uso del lenguaje Ensamblador y lenguaje C para la programación del módulo TWI en el MCU AVR-ATMEl.

4.2 Modos de Operación del módulo TWI

El módulo TWI posee cuatro modos de operación principales. Estos son denominados: Maestro Transmisor (MT), Maestro Receptor (MR), Esclavo Transmisor (ST) y Esclavo Receptor (SR). Algunos de estos modos pueden ser utilizados en la misma aplicación. Como ejemplo, el módulo TWI puede emplearse en modo Maestro

Transmisor (MT) para escribir datos dentro de una EEPROM bajo el protocolo TWI, en modo Maestro Receptor (MR) para leer los datos desde la EEPROM.

Si otros Maestros están presentes en el sistema, algunos de ellos podrían transmitir datos al bus TWI, y luego el modo Esclavo Receptor (SR) puede ser empleado. Es el Software de Aplicación el que decide que modo debe ser empleado.

4.3 Programación del módulo TWI AVR en Ensamblador y C

A continuación se discutirá la programación del módulo TWI en lenguaje Ensamblador y C. Enfocándose en una forma simple de programación del TWI sin utilizar el chequeo del Registro de Estado.

En otras aplicaciones, si no se trata con sistemas críticos y no existe mas que un maestro sobre un bus simple, se puede utilizar este método. Si es necesario tratar con topologías Multimaestro o diseños críticos se debe chequear el valor de la bandera de Estado. En el protocolo I2C, un dispositivo puede ser Maestro o Esclavo, discutiremos los pasos para la programación en cada modo.

4.4 Desarrollo de Ejercicios Propuestos

4.4.1 EJERCICIO 1

4.4.1.1 Tema

Programación en Ensamblador del módulo TWI, Modo Maestro para Escritura.

4.4.1.2 Objetivos

- Exponer el procedimiento para configurar e inicializar el módulo TWI en lenguaje Ensamblador.
- Escribir el código fuente haciendo uso del compilador AVR-Studio, para acceder a los registros del módulo TWI.
- Escribir bytes de datos a través del bus I2C hacia un esclavo direccionado.

4.4.1.3 Descripción

En este modo se transmite un cierto número de bytes de datos a un Esclavo Receptor (SR), configurando el octavo bit del primer byte R/W a ‘0’, podemos determinar la habilitación de Escritura de datos. Para trabajar en modo Maestro, debe ser inicializado el bus TWI, transmitir una condición de Inicio (Start), enviar datos, y transmitir una condición de Parada (Stop).

- **Inicialización**

  Para inicializar el módulo TWI para operar en modo Maestro, se debe seguir los siguientes pasos:

  1. Habilitar la frecuencia de reloj del módulo TWI, configurando los valores del registro TWBR y los bits TWPS [1:0] en el registro TWSR.
  2. Habilitar el módulo TWI, configurando a uno el bit TWEN en el registro TWCR.

- **Transmisión de Condición de Inicio**

Para iniciar la transferencia de datos en operación de modo Maestro, es necesario transmitir una condición de Inicio (Start). Esto es realizado por la configuración de los registros TWEN, TWSTA, y al poner en uno el bit TWINT del registro TWCR. Configurando el bit TWEN a uno habilita el módulo TWI. Configurando el bit TWSTA a uno permite al módulo TWI enviar una condición de inicio cuando el bus se encuentra libre, y configurando a uno el bit TWINT limpia la bandera de interrupción para iniciar la operación del módulo TWI transmitiendo la condición de Inicio. Además se debe sondear la bandera TWINT en el registro TWCR para conocer si la condición de Inicio ha sido transmitida completamente.

- **Transmisión de Datos**

Para enviar un byte de datos, después de transmitir una condición de Inicio (Start), se debe realizar los siguientes pasos:

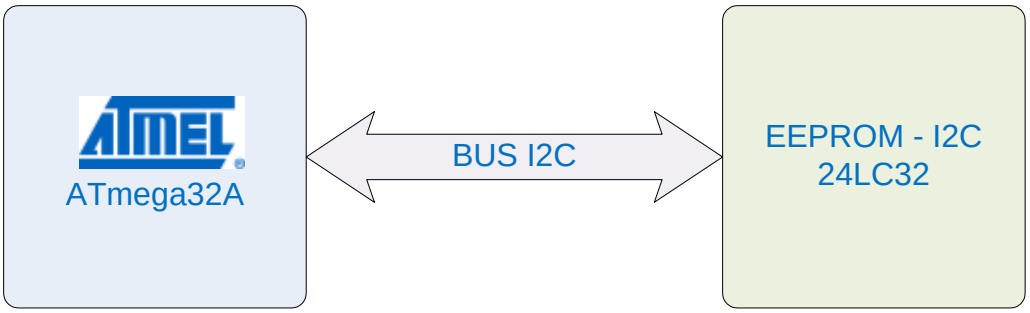
1. Copiar el byte de datos al registro TWDR.
2. Poner a uno el bit TWEN y el bit TWINT del registro TWCR para iniciar el envío del byte de datos.
3. Revisar la bandera TWINT en el registro TWCR para conocer si el byte ha sido transmitido completamente.

Cabe notar que después de una correcta condición de Inicio (Start), podemos transmitir el byte SLA+W (Slave Address+Write) o el byte SLA+R (Slave Address+Read). Además como ha sido explicado, después de enviar correctamente el byte SLA+W podemos escribir a un Esclavo, y después de enviar correctamente el byte SLA+R podemos leer desde un Esclavo.

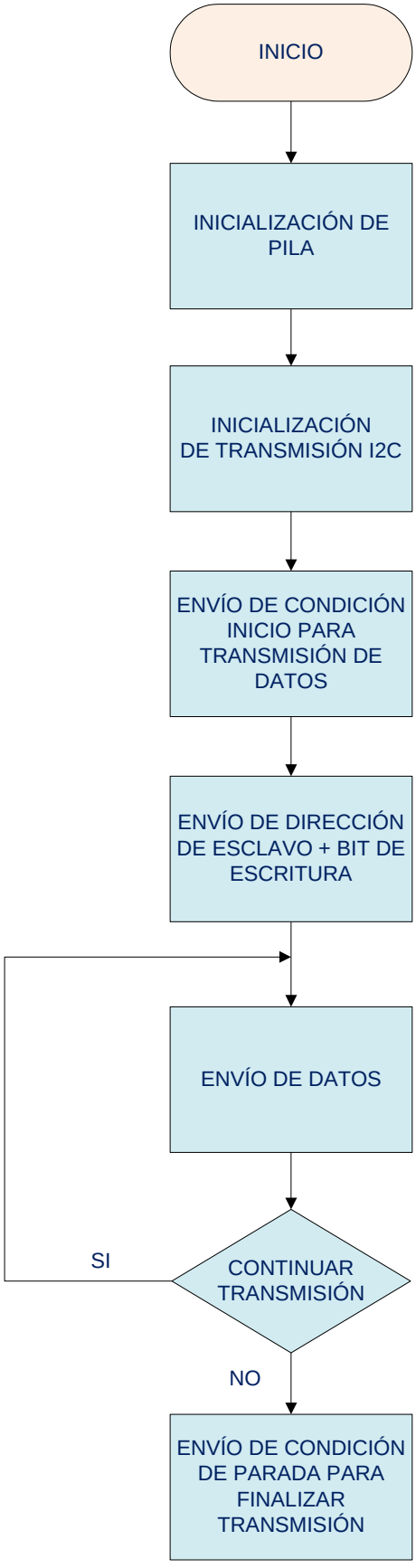
- **Transmisión de Condición de Parada**

Para detener la transferencia de datos, debemos transmitir una Condición de Parada (Stop). Esto se realiza configurando los registros TWEN, TWSTO, y configurando a uno el bit TWINT del registro TWCR. Notar que no podemos consultar el estado de la bandera TWINT después de transmitir una condición de Parada.

4.4.1.4 Diagrama de Bloques



4.4.1.5 Diagrama de Flujo



4.4.1.6 Descripción del Algoritmo

Basándose en el diagrama de bloques mostrado, podemos dar la siguiente descripción:

1. Se declara el Inicio u origen del Código Fuente
2. Es necesario inicializar la pila para el manejo de valores temporales
3. Para trabajar en modo Maestro debe ser inicializado el bus I2C
4. Transmitir una condición de Inicio (Start)
5. Enviar la dirección para el reconocimiento del Esclavo y el bit de Escritura
6. Enviar los datos y
7. Transmitir una condición de Parada (Stop) para liberar el bus y finalizar la transmisión.

4.4.1.7 Código Fuente

Programa 4.1: Muestra como un Maestro escribe datos a un Esclavo direccionado

(Continúa en la siguiente página)

```
.INCLUDE "M32DEF.INC"

.CSEG
.ORG            0x0000

RJMP    INICIO

INICIO:

; INICIALIZACIÓN DE PILA

LDI     R21,    HIGH(RAMEND)
OUT     SPH,    R21
LDI     R21,    LOW(RAMEND)
OUT     SPL,    R21
```

```
; CONFIGURACIÓN E INICIALIZACIÓN DE TRANSMISIÓN

        CALL    I2C_BUS_INICIO
        CALL    I2C_INICIO
        LDI     R27,0xA0        //Dirección de Esclavo + Bit de Escritura  SLA+W
        CALL    I2C_ESCRIBIR
        LDI     R27,0x00
        CALL    I2C_ESCRIBIR

; ENVÍO DE BYTE DE DATOS A DISPOSITIVO I2C (MEMORIA EEPROM)

        LDI     R27,0x48
        CALL    I2C_ESCRIBIR
        LDI     R27,0x4F
        CALL    I2C_ESCRIBIR
        LDI     R27,0x4C
        CALL    I2C_ESCRIBIR
        LDI     R27,0x41
        CALL    I2C_ESCRIBIR
        LDI     R27,0x00
        CALL    I2C_ESCRIBIR
        LDI     R27,0x4D
        CALL    I2C_ESCRIBIR
        LDI     R27,0x55
        CALL    I2C_ESCRIBIR
        LDI     R27,0x4E
        CALL    I2C_ESCRIBIR
        LDI     R27,0x44
        CALL    I2C_ESCRIBIR
        LDI     R27,0x4F
        CALL    I2C_ESCRIBIR
        CALL    I2C_PARADA

LAZO1:  RJMP    LAZO1


; SUBRUTINAS EMPLEADAS

I2C_BUS_INICIO:
        LDI     R21,    0
        OUT     TWSR,   R21
        LDI     R21,    0x47
        OUT     TWBR,   R21
        LDI     R21,    (1<<TWEN)
        OUT     TWCR,   R21
        RET

I2C_INICIO:
        LDI     R21,    (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
        OUT     TWCR,   R21

ESPERA_1:
        IN              R21,    TWCR
        SBRS    R21,    TWINT
        RJMP    ESPERA_1
        RET
```

Programa 4.1: Escritura de datos en Modo Maestro (Continúa en la siguiente página)

Programa 4.1: Escritura de datos en Modo Maestro (Viene de la página anterior)

```
; SUBRUTINAS EMPLEADAS


I2C_ESCRIBIR:
        OUT     TWDR, R27
        LDI     R21,    (1<<TWINT)|(1<<TWEN)
        OUT     TWCR, R21

ESPERA_2:
        IN              R21,    TWCR
        SBRS    R21,    TWINT
        RJMP    ESPERA_2
        RET

I2C_PARADA:
        LDI     R21,    (1<<TWINT)|(1<<TWSTO)|(1<<TWEN)
        OUT     TWCR, R21
        RET

; FIN DEL PROGRAMA
```

4.4.1.8 Simulación

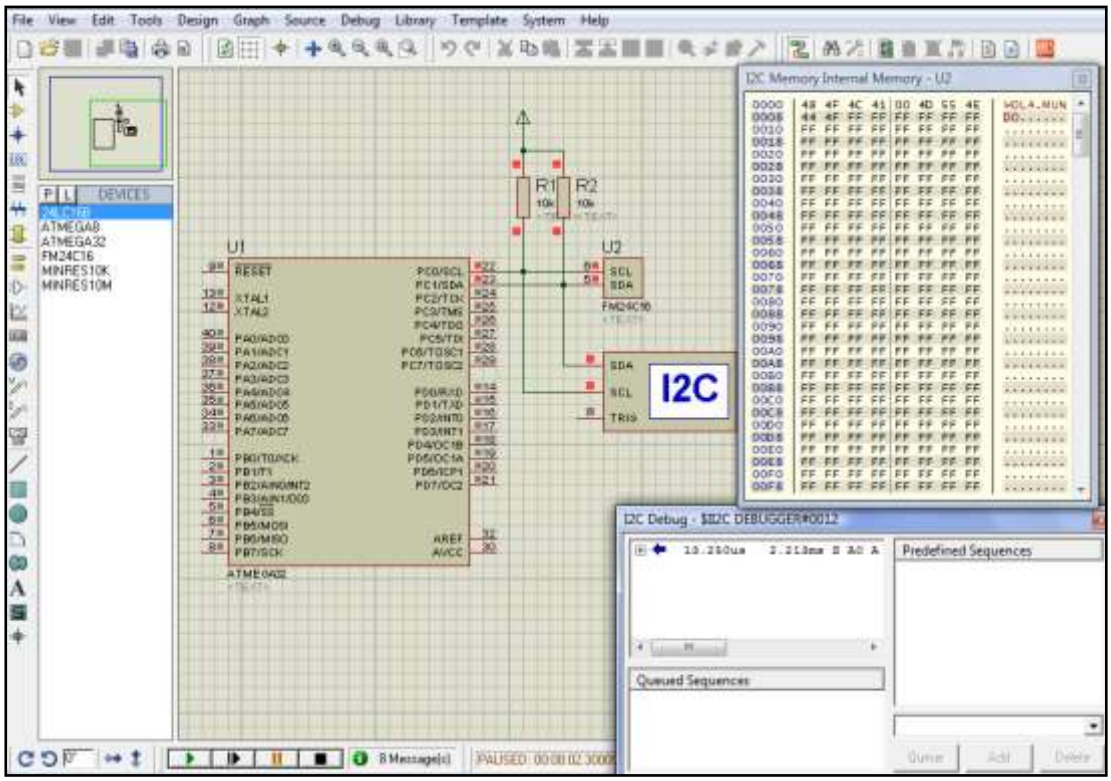


Figura 4.1 Simulación en Proteus programación en Ensamblador para Escritura de Datos

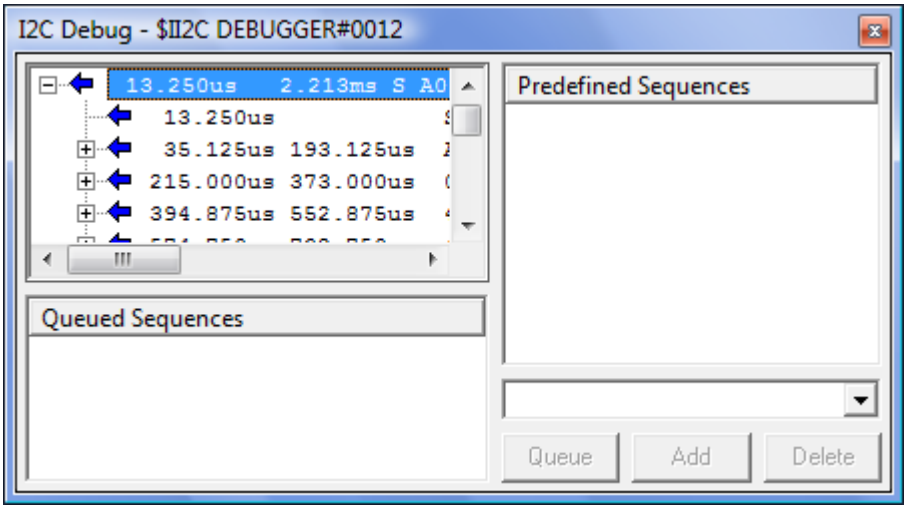


Figura 4.2 Ventana del Depurador para Transmisión I2C en Proteus

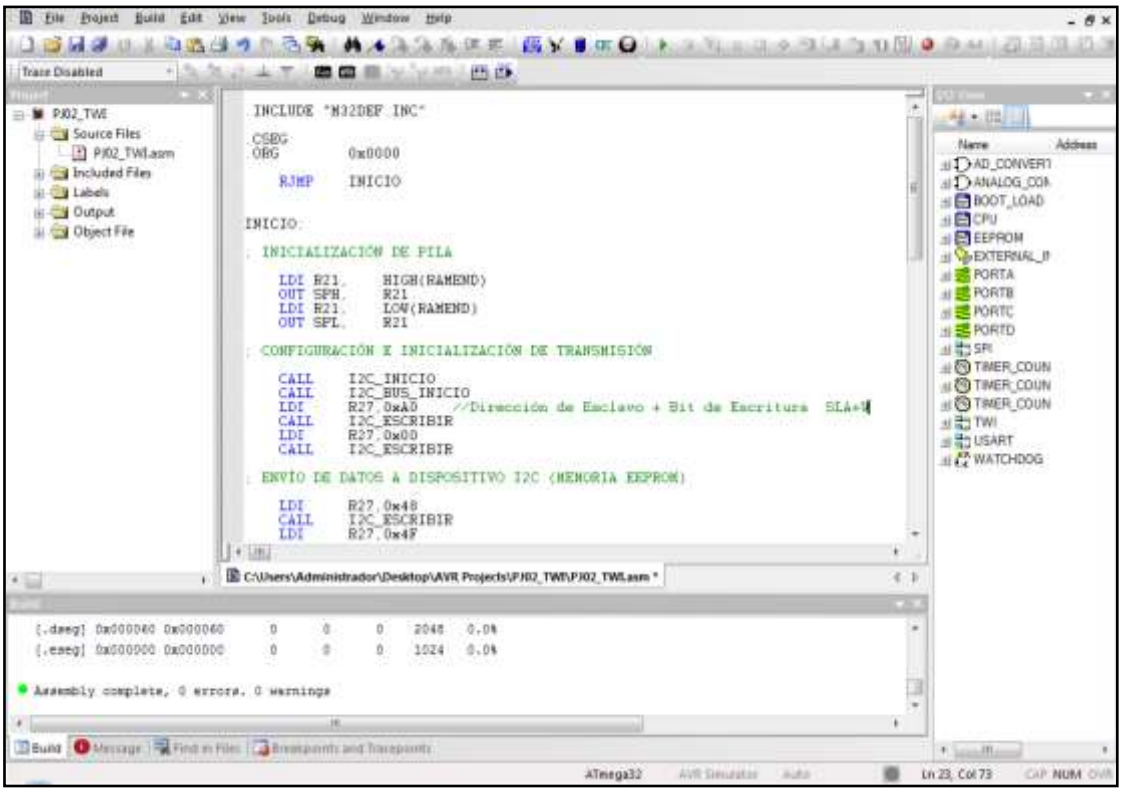


Figura 4.3 AVR-Studio Programación en Ensamblador para Escritura de Datos

4.4.2 EJERCICIO 2

4.4.2.1 Tema

Programación en Ensamblador del módulo TWI, Modo Maestro para Lectura.

4.4.2.2 Objetivos

- Exponer el procedimiento para configurar e inicializar el módulo TWI en lenguaje Ensamblador.
- Escribir el código fuente haciendo uso del compilador AVR-Studio, para acceder a los registros del módulo TWI.
- Leer bytes de datos a través del bus I2C desde un esclavo direccionado.

4.4.2.3 Descripción

En este modo se recibe un cierto número de bytes de datos desde un Esclavo Receptor (SR), configurando el octavo bit del primer byte R/W a '1', podemos determinar la habilitación de Lectura de datos. Para trabajar en modo Maestro, debe ser inicializado el bus TWI, transmitir una condición de Inicio (Start), recibir datos, y transmitir una condición de Parada (Stop).

- **Inicialización**

  Para inicializar el módulo TWI para operar en modo Maestro, se debe seguir los siguientes pasos:

3. Habilitar la frecuencia de reloj del módulo TWI, configurando los valores del registro TWBR y los bits TWPS [1:0] en el registro TWSR.
4. Habilitar el módulo TWI, configurando a uno el bit TWEN en el registro TWCR.

- **Transmisión de Condición de Inicio**

Para iniciar la transferencia de datos en operación de modo Maestro, es necesario transmitir una condición de Inicio (Start). Esto es realizado por la configuración de los registros TWEN, TWSTA, y al poner en uno el bit TWINT del registro TWCR. Configurando el bit TWEN a uno habilita el módulo TWI. Configurando el bit TWSTA a uno permite al módulo TWI enviar una condición de inicio cuando el bus se encuentra libre, y configurando a uno el bit TWINT limpia la bandera de interrupción para iniciar la operación del módulo TWI transmitiendo la condición de Inicio. Además se debe sondear la bandera TWINT en el registro TWCR para conocer si la condición de Inicio ha sido transmitida completamente.

- **Recepción de Datos**

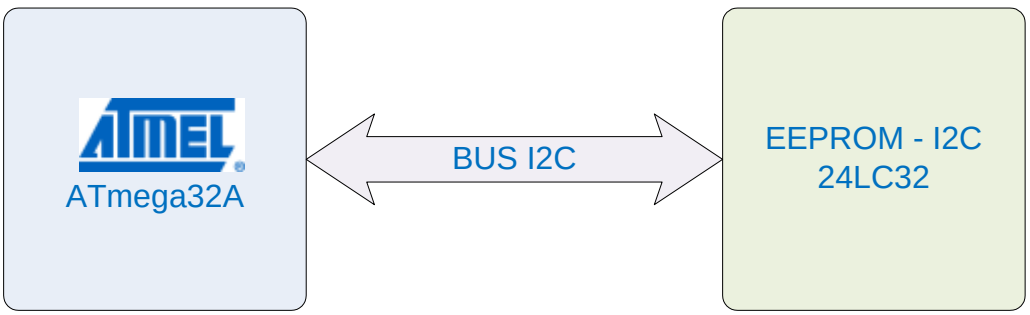
Para recibir un byte de Datos, luego de transmitir el byte SLA+R, debemos realizar los siguientes pasos:

1. Configurar a uno los bits TWEN y TWINT del registro TWCR para iniciar la recepción de un byte de datos. Notar que si es necesario retornar un Acuse de Recibo (ACK) después de la recepción de datos es necesario también configurar a uno el bit TWEA del registro TWCR.
2. Revisar la bandera TWINT en el registro TWCR para conocer si un byte ha sido completamente recibido.
3. Copiar el byte de datos recibido desde el registro TWDR a otro registro como respaldo del mismo.

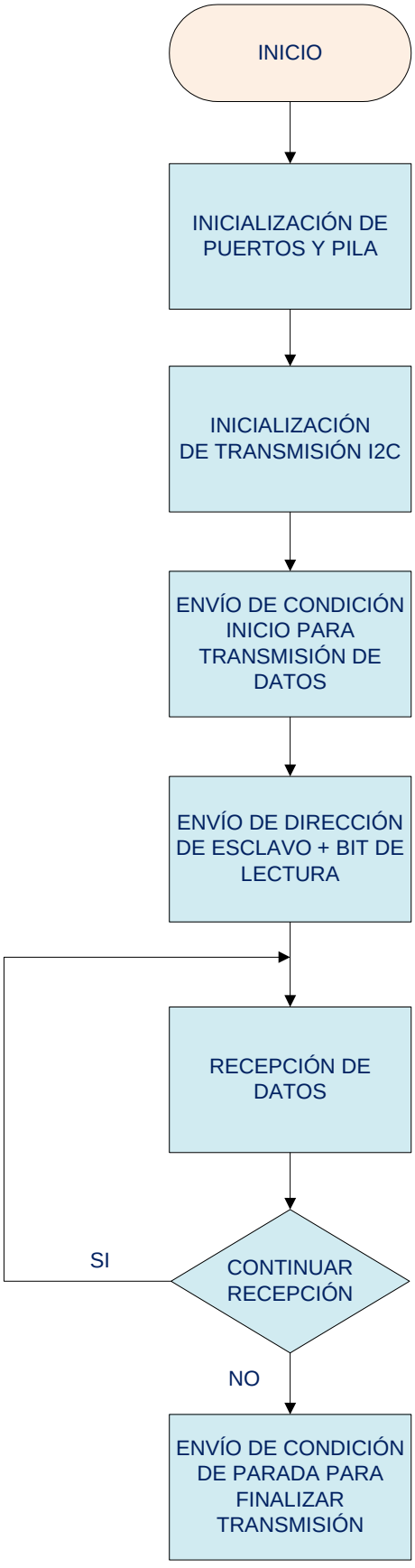
- **Transmisión de Condición de Parada**

Para detener la transferencia de datos, debemos transmitir una Condición de Parada (Stop). Esto se realiza configurando los registros TWEN, TWSTO, y configurando a uno el bit TWINT del registro TWCR. Notar que no podemos consultar el estado de la bandera TWINT después de transmitir una condición de Parada.

4.4.2.4 Diagrama de Bloques



4.4.2.5 Diagrama de Flujo



4.4.2.6 Descripción del Algoritmo

Basándose en el diagrama de bloques mostrado, podemos dar la siguiente descripción:

1. Se declara el Inicio u origen del Código Fuente
2. Es necesario inicializar la pila para el manejo de valores temporales y el puerto para recepción de Datos
3. Para trabajar en modo Maestro debe ser inicializado el bus I2C
4. Transmitir una condición de Inicio (Start)
5. Enviar la dirección para el reconocimiento del Esclavo y el bit de Lectura
6. Recibir los datos y
7. Transmitir una condición de Parada (Stop) para liberar el bus y finalizar la transmisión.

4.4.2.7 Código Fuente

Programa 4.2: Muestra como leer un byte de datos desde un Esclavo direccionado, presenta el resultado en el Puerto A. (Continúa en la siguiente página)

```
.INCLUDE "M32DEF.INC"

.CSEG
.ORG    0x0000

RJMP    INICIO
INICIO:

; INICIALIZACIÓN DE PUERTOS

        LDI     R21,    $FF
        OUT     DDRA,   R21
```

Programa 4.2: Lectura de datos en Modo Maestro (Viene de la página anterior)

```
; INICIALIZACIÓN DE PILA

        LDI     R21,    HIGH(RAMEND)
        OUT     SPH,    R21
        LDI     R21,    LOW(RAMEND)
        OUT     SPL,    R21

; CONFIGURACIÓN E INICIALIZACIÓN DE LECTURA

        CALL    I2C_BUS_INICIO
        CALL    I2C_INICIO
        LDI     R27,0xA1        //Dirección de Esclavo + Bit de Lectura  SLA+R
        CALL    I2C_ESCRIBIR
        CALL    I2C_LEER
        OUT     PORTA,R27
        CALL    I2C_PARADA

LAZO1:  RJMP    LAZO1


; SUBRUTINAS EMPLEADAS

I2C_BUS_INICIO:
        LDI     R21,    0
        OUT     TWSR,   R21
        LDI     R21,    0x47
        OUT     TWBR,   R21
        LDI     R21,    (1<<TWEN)
        OUT     TWCR,   R21
        RET

I2C_INICIO:
        LDI     R21,    (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
        OUT     TWCR,   R21

ESPERA_1:
        IN      R21,    TWCR
        SBRS    R21,    TWINT
        RJMP    ESPERA_1
        RET

I2C_LEER:
        LDI     R21,    (1<<TWINT)|(1<<TWEN)
        OUT     TWCR,   R21

ESPERA_2:
        IN      R21,    TWCR
        SBRS    R21,    TWINT
        RJMP    ESPERA_2
        IN      R27,    TWDR
        RET
```

Programa 4.2: Lectura de datos en Modo Maestro (Continúa en la siguiente página)

Programa 4.2: Lectura de datos en Modo Maestro (Viene de la página anterior)

```
I2C_ESCRIBIR:
        OUT     TWDR, R27
        LDI     R21,    (1<<TWINT)|(1<<TWEN)
        OUT     TWCR, R21

ESPERA_3:
        IN      R21,    TWCR
        SBRS    R21,    TWINT
        RJMP    ESPERA_3
        RET

I2C_PARADA:
        LDI     R21,    (1<<TWINT)|(1<<TWSTO)|(1<<TWEN)
        OUT     TWCR, R21
        RET

; FIN DEL PROGRAMA
```

4.4.2.8 Simulación

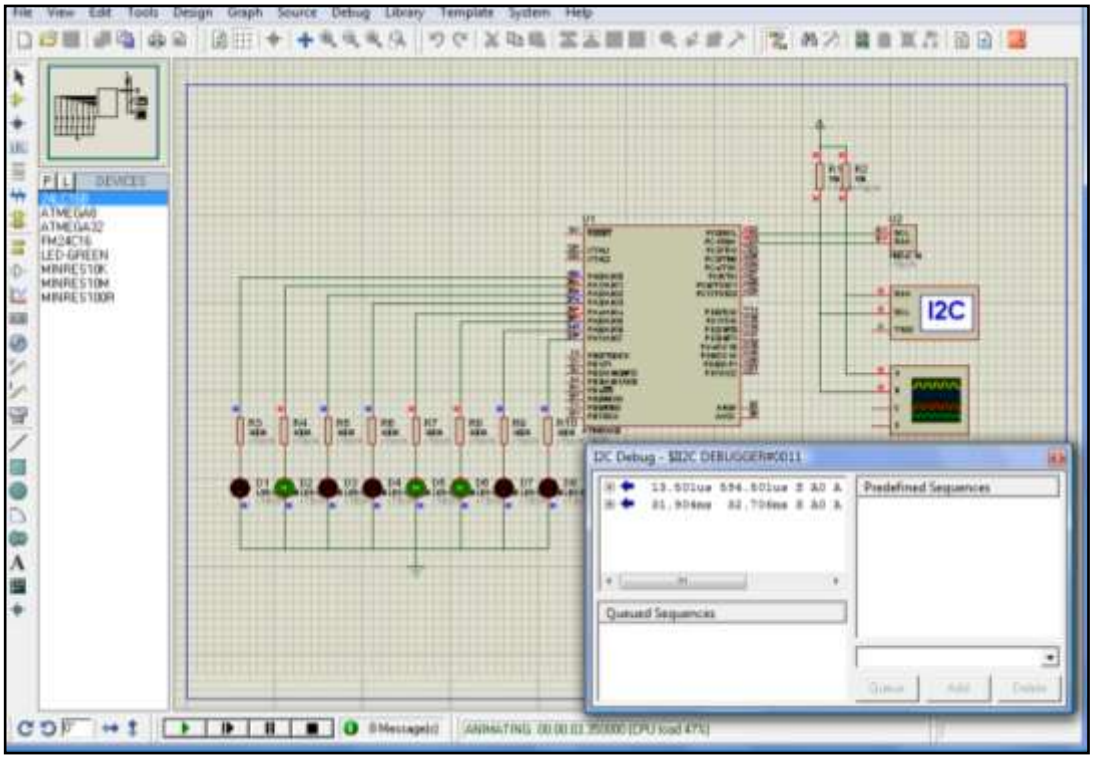


Figura 4.4 Simulación en Proteus programación en Ensamblador para Lectura de Datos

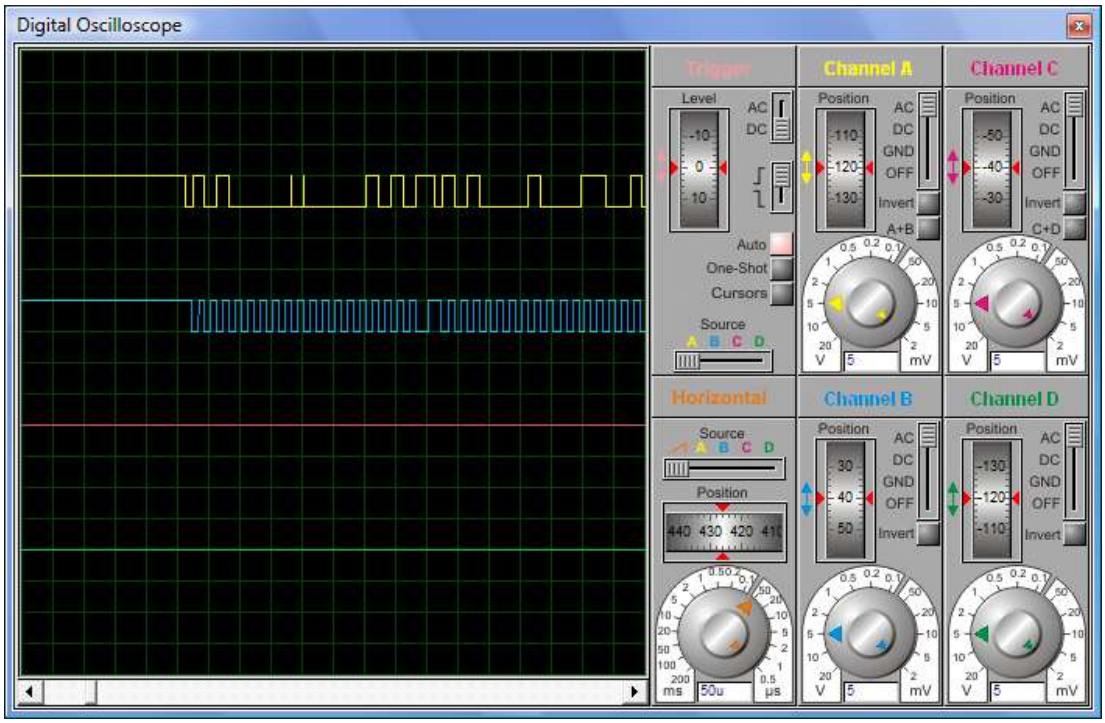


Figura 4.5 Ventana del Osciloscopio para Visualización de Transmisión I2C en Proteus

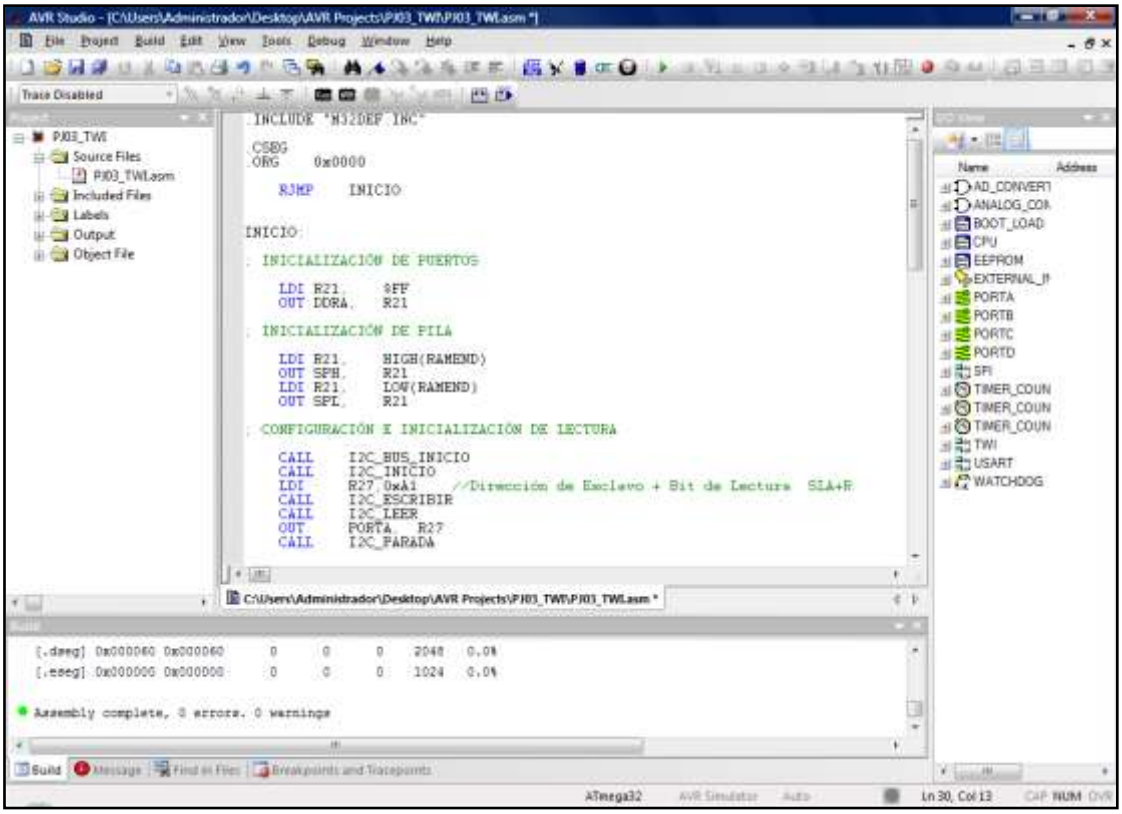


Figura 4.6 AVR-Studio Programación en Ensamblador para Lectura de Datos

4.4.3 EJERCICIO 3

4.4.3.1 Tema

Programación en C del módulo TWI configurado en Modo Maestro para Escritura

4.4.3.2 Objetivos

- Exponer el procedimiento para configurar e inicializar el módulo TWI en lenguaje C.
- Escribir el código fuente haciendo uso del compilador AVR-Studio y la plataforma WINAVR, para acceder a los registros del módulo TWI.
- Escribir bytes de datos a través del bus I2C por medio de un esclavo direccionado.

4.4.3.3 Descripción

Versión en lenguaje C del modo de Transmisión Maestro para Escritura por bus I2C. En este modo se transmite un cierto número de bytes de datos hacia un Esclavo Receptor (SR), configurando el octavo bit del primer byte R/W a ‘0’, para poder determinar la habilitación de Escritura de datos.

Para trabajar en modo Maestro, debe ser inicializado el bus TWI, transmitir una condición de Inicio (Start), enviar datos, y transmitir una condición de Parada (Stop) para liberar el bus y finalizar la transmisión.

- **Inicialización**

Para inicializar el módulo TWI para operar en modo Maestro, se debe seguir los siguientes pasos:

1. Habilitar la frecuencia de reloj del módulo TWI, configurando los valores del registro TWBR y los bits TWPS [1:0] en el registro TWSR.
2. Habilitar el módulo TWI, configurando a uno el bit TWEN en el registro TWCR.

- **Transmisión de Condición de Inicio**

Para iniciar la transferencia de datos en operación de modo Maestro, es necesario transmitir una condición de Inicio (Start). Esto es realizado por la configuración de los registros TWEN, TWSTA, y al poner en uno el bit TWINT del registro TWCR. Configurando el bit TWEN a uno habilita el módulo TWI. Configurando el bit TWSTA a uno permite al módulo TWI enviar una condición de inicio cuando el bus se encuentra libre, y configurando a uno el bit TWINT limpia la bandera de interrupción para iniciar la operación del módulo TWI transmitiendo la condición de Inicio. Además se debe sondear la bandera TWINT en el registro TWCR para conocer si la condición de Inicio ha sido transmitida completamente.

- **Transmisión de Datos**

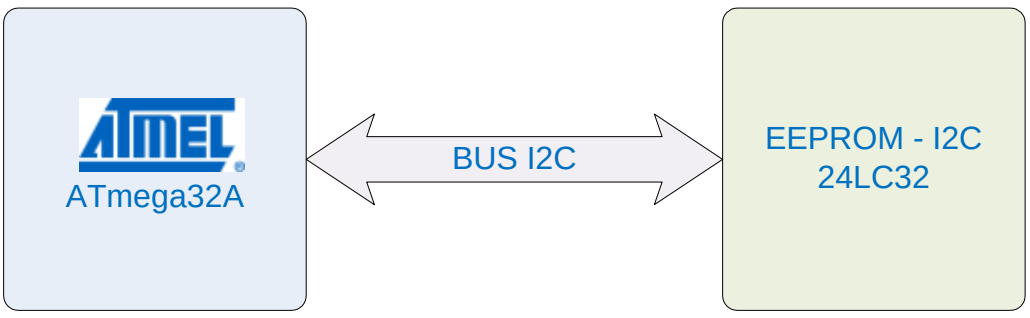
Para enviar un byte de datos, después de transmitir una condición de Inicio (Start), se debe realizar los siguientes pasos:

1. Copiar el byte de datos al registro TWDR.
2. Poner a uno el bit TWEN y el bit TWINT del registro TWCR para iniciar el envío del byte de datos.
3. Revisar la bandera TWINT en el registro TWCR para conocer si el byte ha sido transmitido completamente.

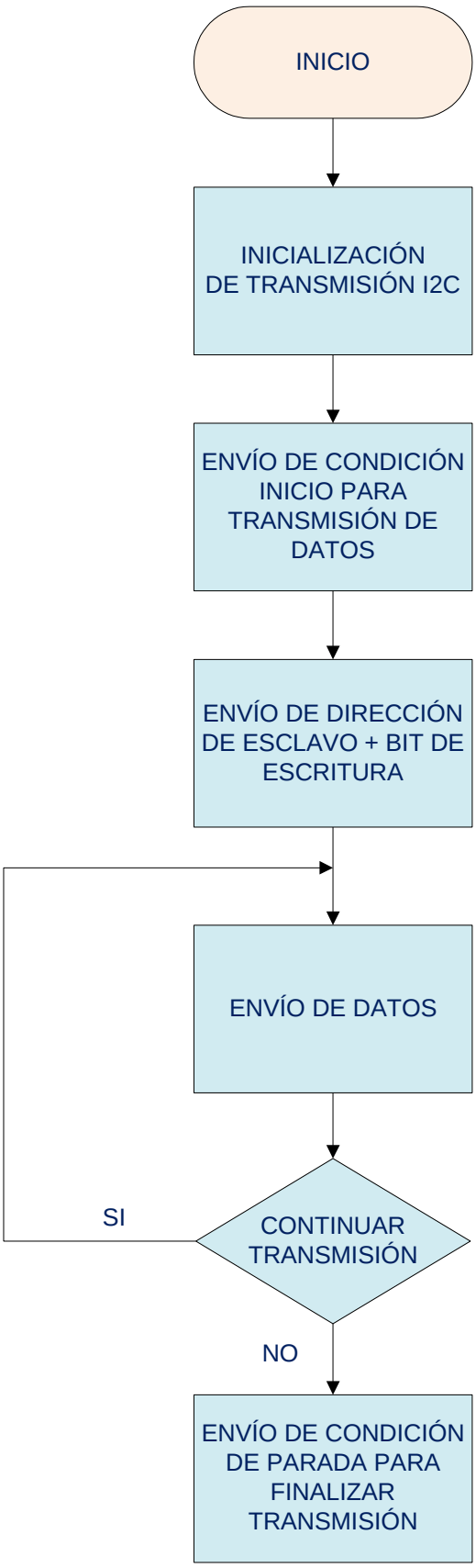
- **Transmisión de Condición de Parada**

Para detener la transferencia de datos, debemos transmitir una Condición de Parada (Stop). Esto se realiza configurando los registros TWEN, TWSTO, y configurando a uno el bit TWINT del registro TWCR. Notar que no podemos consultar el estado de la bandera TWINT después de transmitir una condición de Parada.

4.4.3.4 Diagrama de Bloques



4.4.3.5 Diagrama de Flujo



4.4.3.6 Descripción del Algoritmo

Basándose en el diagrama de bloques mostrado, podemos dar la siguiente descripción:

1. Se declara el bloque del Programa principal
2. Para trabajar en modo Maestro debe ser inicializado el bus I2C
3. Transmitir una condición de Inicio (Start)
4. Enviar la dirección para el reconocimiento del Esclavo y el bit de Escritura
5. Enviar los datos y
6. Transmitir una condición de Parada (Stop) para liberar el bus y finalizar la transmisión.

4.4.3.7 Código Fuente

Programa 4.3: Muestra como un Maestro escribe datos a un Esclavo direccionado en C (Continúa en la siguiente página)

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


//*****************************************

void i2c_escribir(unsigned char dato)
{

        TWDR = dato;
        TWCR = (1<<TWINT)|(1<<TWEN);

        while((TWCR & (1<<TWINT)) == 0);

}

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

Programa 4.3: Escritura de datos en Modo Maestro en C (Viene de la página anterior)

```
void i2c_inicio(void)
{
        TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);

        while((TWCR & (1<<TWINT)) == 0);
}

//*****************************************

void i2c_parada ()
{
        TWCR = (1<<TWINT)|(1<<TWSTO)|(1<<TWEN);
}

//*****************************************

void i2c_bus_inicio (void)
{
        TWSR = 0x00;
        TWBR = 0x47;
        TWCR = 0x04;
}

//*****************************************
//      Programa Principal
//*****************************************

int main (void)
{

        i2c_bus_inicio();
        i2c_inicio();
        i2c_escribir(0xA0);
        i2c_escribir(0x00);
        i2c_escribir(0x48);
        i2c_escribir(0x4F);
        i2c_escribir(0x4C);
        i2c_escribir(0x41);
        i2c_escribir(0x00);
        i2c_escribir(0x4D);
        i2c_escribir(0x55);
        i2c_escribir(0x4E);
        i2c_escribir(0x44);
        i2c_escribir(0x4F);
        i2c_parada();

        while(1);
        return 0;

}

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

4.4.3.8 Simulación

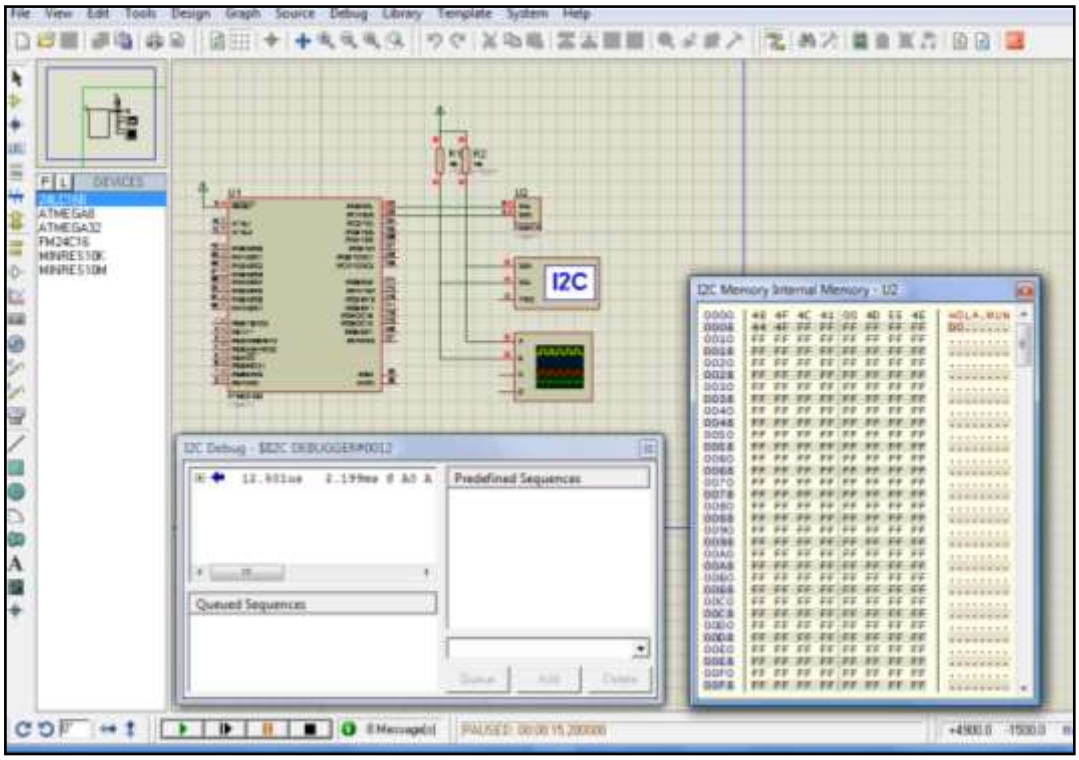


Figura 4.7 Simulación en Proteus programación en C para Escritura de Datos

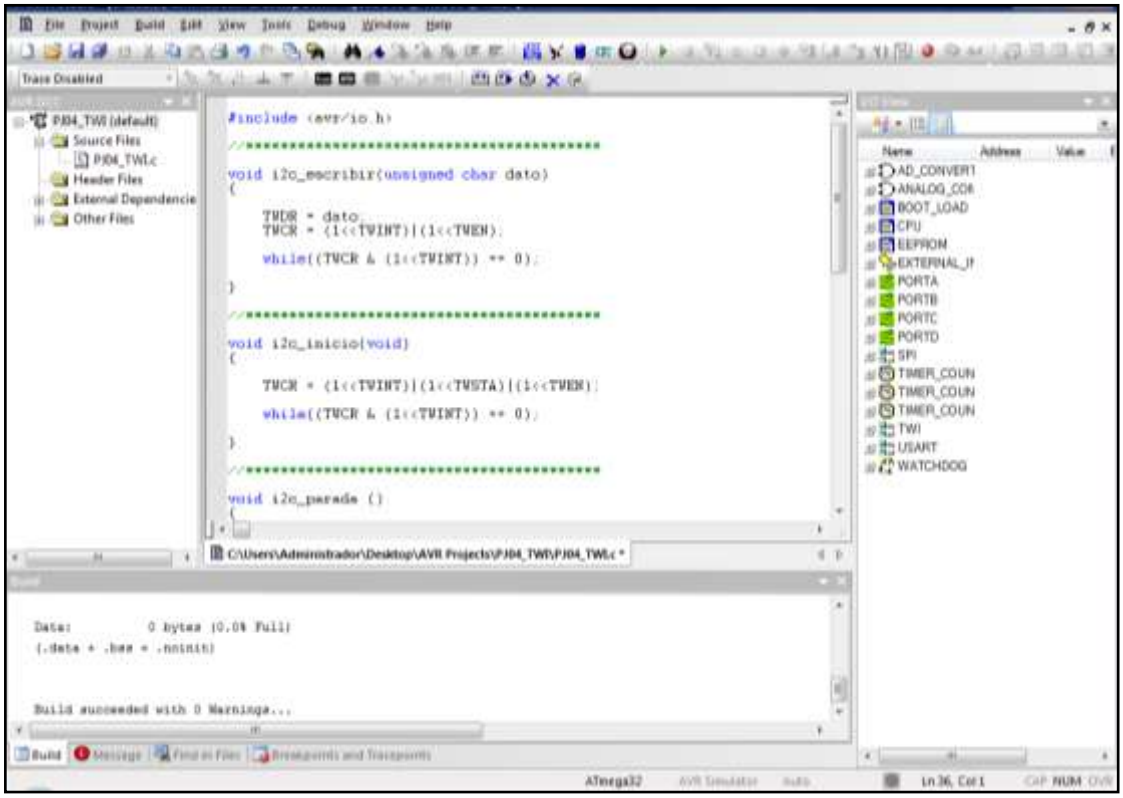


Figura 4.8 AVR-Studio Programación en C para Escritura de Datos

4.4.4 EJERCICIO 4

4.4.4.1 Tema

Programación en C del módulo TWI configurado en Modo Maestro para Lectura

4.4.4.2 Objetivos

- Exponer el procedimiento para configurar e inicializar el módulo TWI en lenguaje C.
- Escribir el código fuente haciendo uso del compilador AVR-Studio y la plataforma WINAVR, para acceder a los registros del módulo TWI.
- Leer bytes de datos a través del bus I2C por medio de un esclavo direccionado.

4.4.4.3 Descripción

Versión en lenguaje C del modo de Recepción Maestro para Lectura por bus I2C. En este modo se recibe un cierto número de bytes de datos desde un Esclavo direccionado (SR), configurando el octavo bit del primer byte R/W a ‘1’, para poder determinar la habilitación de Lectura de datos.

Para trabajar en modo Maestro, debe ser inicializado el bus TWI, transmitir una condición de Inicio (Start), enviar datos, y transmitir una condición de Parada (Stop) para liberar el bus y finalizar la transmisión.

- **Inicialización**

Para inicializar el módulo TWI para operar en modo Maestro, se debe seguir los siguientes pasos:

1. Habilitar la frecuencia de reloj del módulo TWI, configurando los valores del registro TWBR y los bits TWPS [1:0] en el registro TWSR.
2. Habilitar el módulo TWI, configurando a uno el bit TWEN en el registro TWCR.

- **Transmisión de Condición de Inicio**

Para iniciar la transferencia de datos en operación de modo Maestro, es necesario transmitir una condición de Inicio (Start). Esto es realizado por la configuración de los registros TWEN, TWSTA, y al poner en uno el bit TWINT del registro TWCR. Configurando el bit TWEN a uno habilita el módulo TWI. Configurando el bit TWSTA a uno permite al módulo TWI enviar una condición de inicio cuando el bus se encuentra libre, y configurando a uno el bit TWINT limpia la bandera de interrupción para iniciar la operación del módulo TWI transmitiendo la condición de Inicio. Además se debe sondear la bandera TWINT en el registro TWCR para conocer si la condición de Inicio ha sido transmitida completamente.

- **Recepción de Datos**

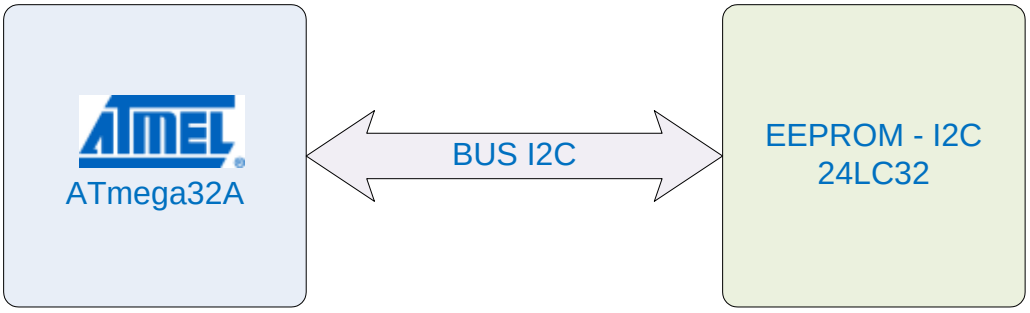
  Para recibir un byte de Datos, luego de transmitir el byte SLA+R, debemos realizar los siguientes pasos:

  1. Configurar a uno los bits TWEN y TWINT del registro TWCR para iniciar la recepción de un byte de datos. Notar que si es necesario retornar un Acuse de Recibo (ACK) después de la recepción de datos es necesario también configurar a uno el bit TWEA del registro TWCR.
  2. Revisar la bandera TWINT en el registro TWCR para conocer si un byte ha sido completamente recibido.
  3. Copiar el byte de datos recibido desde el registro TWDR a otro registro como respaldo del mismo.

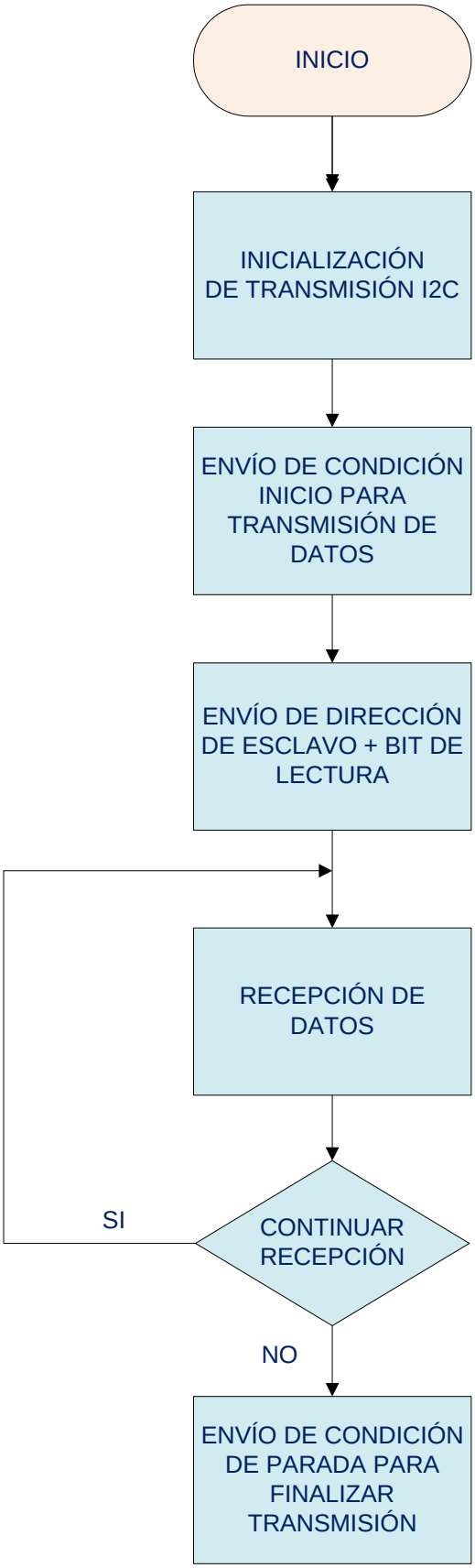
- **Transmisión de Condición de Parada**

  Para detener la transferencia de datos, debemos transmitir una Condición de Parada (Stop). Esto se realiza configurando los registros TWEN, TWSTO, y configurando a uno el bit TWINT del registro TWCR.

4.4.4.4 Diagrama de Bloques



4.4.4.5 Diagrama de Flujo



4.4.4.6 Descripción del Algoritmo

Basándose en el diagrama de bloques mostrado, podemos dar la siguiente descripción:

1. Se declara el bloque del Programa principal
2. Para trabajar en modo Maestro debe ser inicializado el bus I2C
3. Transmitir una condición de Inicio (Start)
4. Enviar la dirección para el reconocimiento del Esclavo y el bit de Lectura
5. Recibir los datos y
6. Transmitir una condición de Parada (Stop) para liberar el bus y finalizar la transmisión.

4.4.4.7 Código Fuente

Programa 4.4: Muestra como un Maestro Lee datos desde un Esclavo direccionado en C (Continúa en la siguiente página)

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


//*****************************************

void i2c_escribir(unsigned char dato)
{

        TWDR = dato;
        TWCR = (1<<TWINT)|(1<<TWEN);

        while((TWCR & (1<<TWINT)) == 0);

}

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

Programa 4.4: Lectura de datos en Modo Maestro en C (Viene de la página anterior)

```
//*****************************************
unsigned char i2c_leer(unsigned char datos)
{
if(datos==0)
        TWCR = (1<<TWINT)|(1<<TWEA)|(1<<TWEN);
else
        TWCR = (1<<TWINT)|(1<<TWEN);
while((TWCR & (1<<TWINT)) == 0);
return TWDR;
}

//*****************************************
void i2c_inicio(void)
{
        TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
        while((TWCR & (1<<TWINT)) == 0);
}

//*****************************************

void i2c_parada ()
{
        TWCR = (1<<TWINT)|(1<<TWSTO)|(1<<TWEN);
}

//*****************************************

void i2c_bus_inicio (void)
{
        TWSR = 0x00;
        TWBR = 0x47;
        TWCR = 0x04;
}

//*****************************************
//      Programa Principal
//*****************************************
int main (void)
{
        unsigned char i=0;
        DDRA = 0xFF;

        i2c_bus_inicio();
        i2c_inicio();
        i2c_escribir(0xA1);
        i = i2c_leer(1);
        PORTA = i;
        i2c_parada();
        while(1);
        return 0;
        while(1);
        return 0;
}
```

4.4.4.8 Simulación

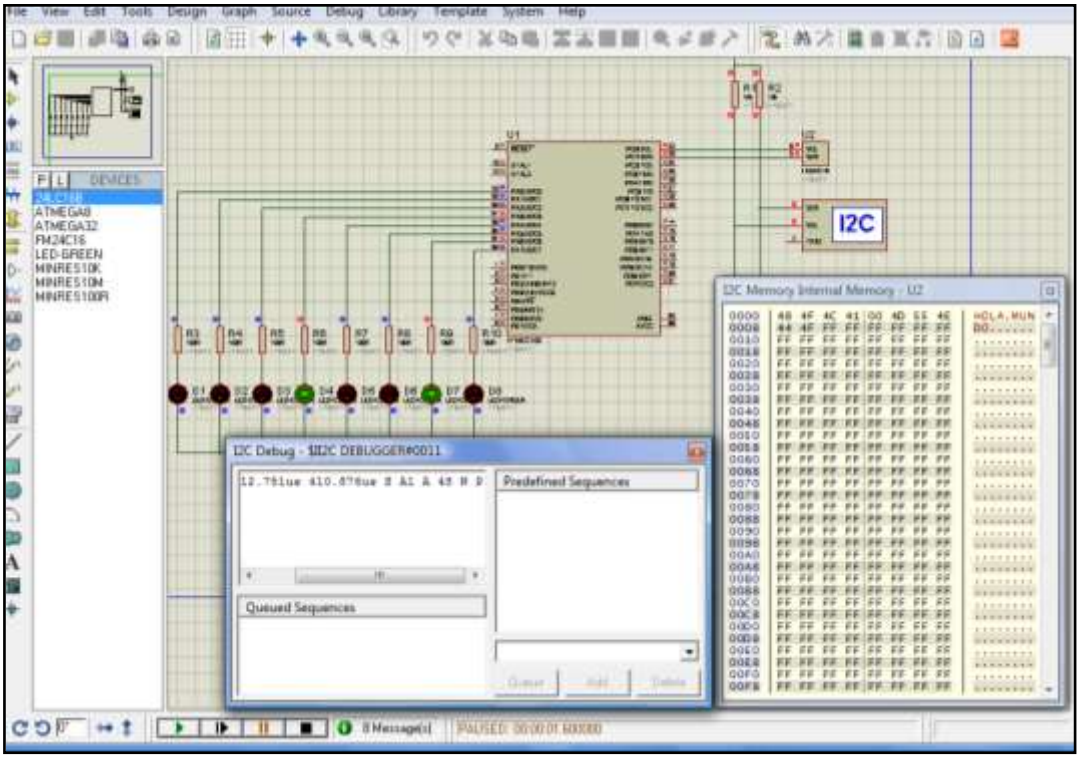


Figura 4.9 Simulación en Proteus programación en C para Lectura de Datos

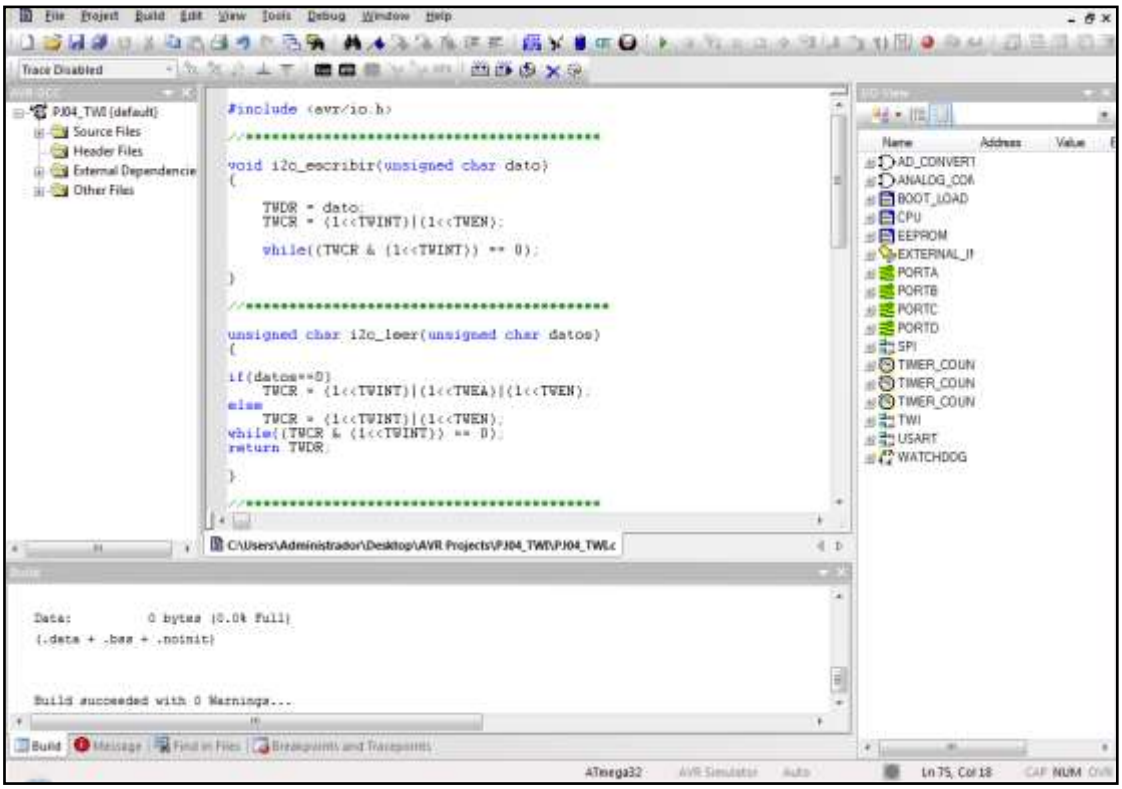


Figura 4.10 AVR-Studio Programación en C para lectura de Datos

4.4.5 EJERCICIO 5

4.4.5.1 Tema

Programación en Ensamblador del módulo TWI configurado en Modo Maestro para Escritura / Lectura

4.4.5.2 Objetivos

- Exponer el procedimiento para configurar e inicializar el módulo TWI en lenguaje Ensamblador.
- Escribir el código fuente haciendo uso del compilador AVR-Studio, para acceder a los registros del módulo TWI.
- Escribir y Leer bytes de datos a través del bus I2C por medio de un esclavo direccionado.

4.4.5.3 Descripción

En este modo se transmite y recibe un cierto número de bytes de datos por medio de un Esclavo Receptor (SR), configurando el octavo bit del primer byte R/W se habilita la Escritura (byte R/W a ‘0’) y Lectura (byte R/W a ‘1’) de datos.

- **Inicialización**

  Para inicializar el módulo TWI para operar en modo Maestro, se debe seguir los siguientes pasos:

1. Habilitar la frecuencia de reloj del módulo TWI, configurando los valores del registro TWBR y los bits TWPS [1:0] en el registro TWSR.
2. Habilitar el módulo TWI, configurando a uno el bit TWEN en el registro TWCR.

- **Transmisión de Condición de Inicio**

Para iniciar la transferencia de datos en operación de modo Maestro, es necesario transmitir una condición de Inicio (Start). Esto es realizado por la configuración de los registros TWEN, TWSTA, y al poner en uno el bit TWINT del registro TWCR. Configurando el bit TWEN a uno habilita el módulo TWI. Configurando el bit TWSTA a uno permite al módulo TWI enviar una condición de inicio cuando el bus se encuentra libre, y configurando a uno el bit TWINT limpia la bandera de interrupción para iniciar la operación del módulo TWI transmitiendo la condición de Inicio. Además se debe sondear la bandera TWINT en el registro TWCR para conocer si la condición de Inicio ha sido transmitida completamente.

- **Recepción de Datos**

Para recibir un byte de Datos, luego de transmitir el byte SLA+R, debemos realizar los siguientes pasos:

1. Configurar a uno los bits TWEN y TWINT del registro TWCR para iniciar la recepción de un byte de datos. Notar que si es necesario retornar un Acuse de Recibo (ACK) después de la recepción de datos

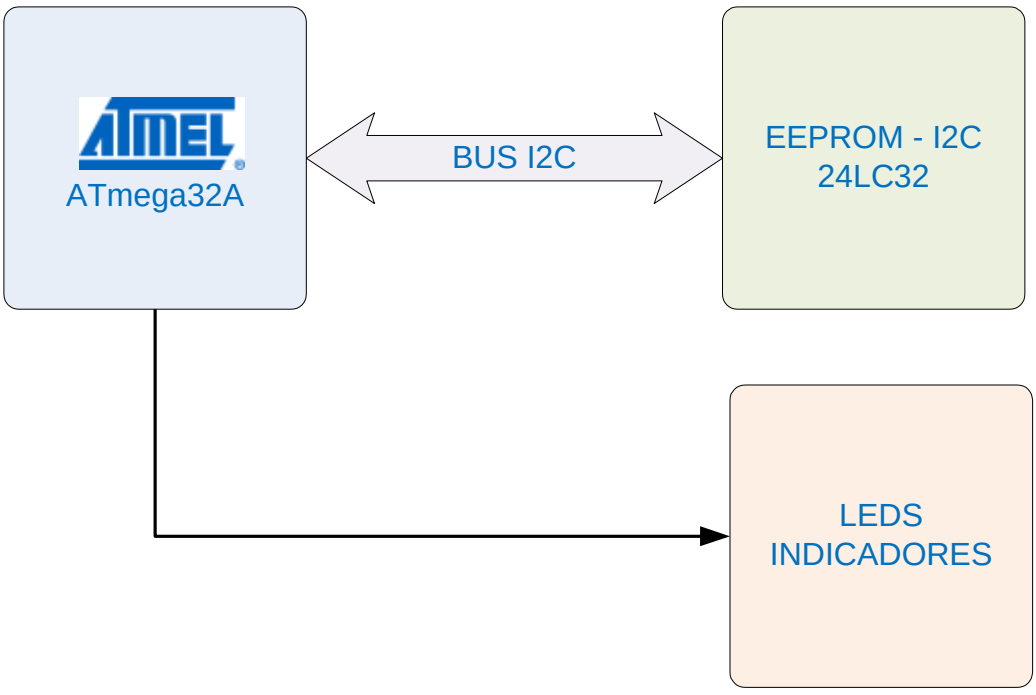
es necesario también configurar a uno el bit TWEA del registro TWCR.

2. Revisar la bandera TWINT en el registro TWCR para conocer si un byte ha sido completamente recibido.
3. Copiar el byte de datos recibido desde el registro TWDR a otro registro como respaldo del mismo.

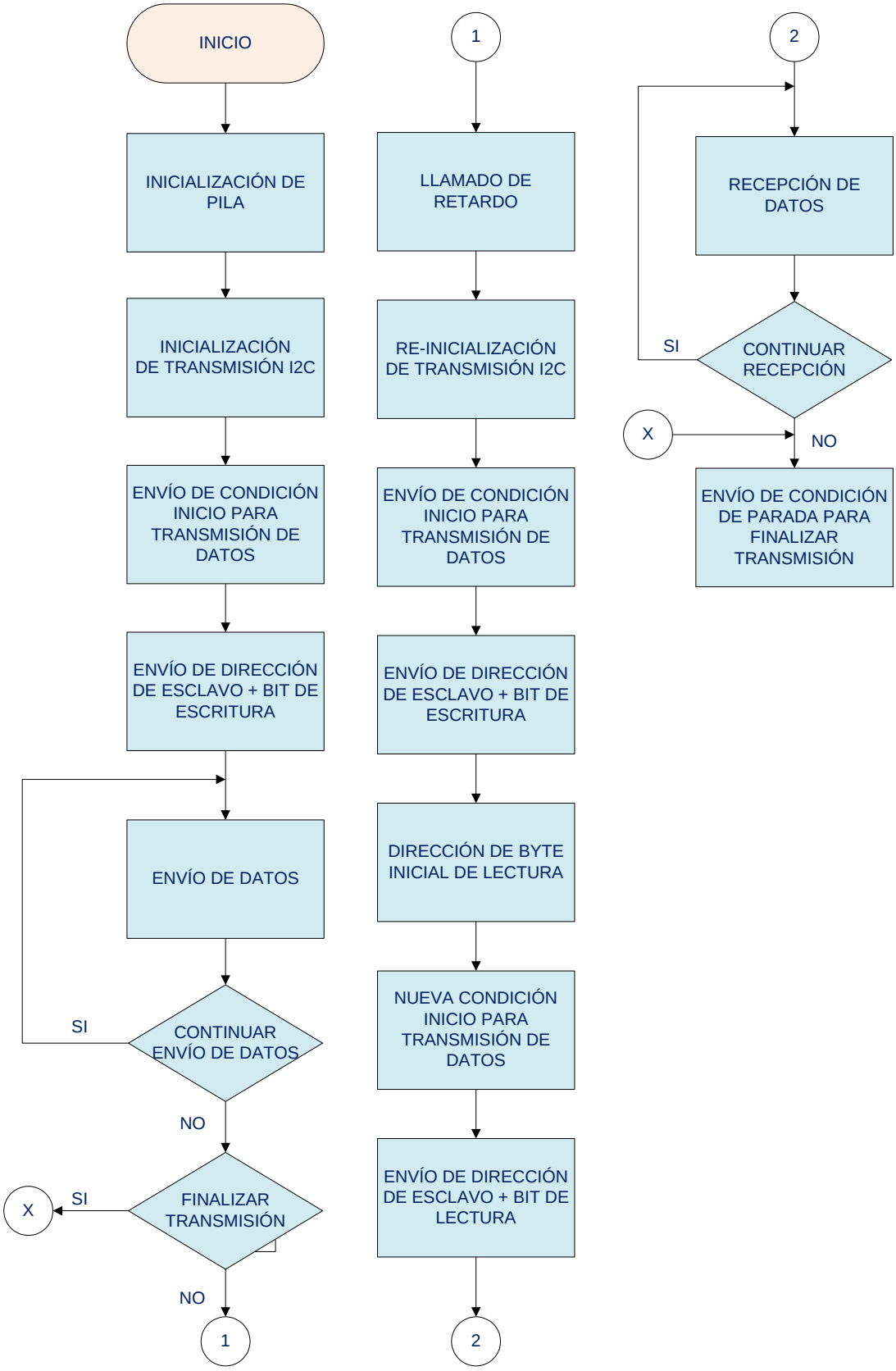
- **Transmisión de Condición de Parada**

  Para detener la transferencia de datos, debemos transmitir una Condición de Parada (Stop). Esto se realiza configurando los registros TWEN, TWSTO, y configurando a uno el bit TWINT del registro TWCR.

4.4.5.4 Diagrama de Bloques



4.4.5.5 Diagrama de Flujo



4.4.5.6 Descripción del Algoritmo

Basándose en el diagrama de bloques mostrado, podemos dar la siguiente descripción:

1. Se declara el Inicio u origen del Código Fuente
2. Es necesario inicializar la pila para el manejo de valores temporales y el puerto para recepción de Datos
3. Para trabajar en modo Maestro debe ser inicializado el bus I2C
4. Transmitir una condición de Inicio (Start)
5. Enviar la dirección para el reconocimiento del Esclavo y el bit de Escritura
6. Transmitir los datos y
7. Transmitir una condición de Parada (Stop) solo en caso de ser necesario finalizar la transmisión.
8. Se emplea un retardo de ¼ de segundo para un oscilador de 8Mhz, para estabilizar el bus I2C
9. Se repiten los pasos 3,4 y 5
10. Enviar la dirección del byte inicial para el proceso de Lectura
11. Se envía una nueva condición de Inicio para cambiar a estado de Lectura
12. Enviar la dirección para el reconocimiento del Esclavo y el bit de Lectura
13. Recibir los datos y
14. Transmitir una condición de Parada (Stop) final para liberar el bus y culminar en su totalidad la transmisión.

4.4.5.7 Código Fuente

Programa 4.5: Muestra como un Maestro Escribe y Lee datos por medio de un Esclavo direccionado en Ensamblador (Continúa en la siguiente página)

```
.INCLUDE "M32DEF.INC"

.CSEG
.ORG    0x0000

        RJMP    INICIO

INICIO:

; INICIALIZACIÓN DE PUERTOS

        LDI     R21,    $FF
        OUT     DDRA,   R21

; INICIALIZACIÓN DE PILA

        LDI     R21,    HIGH(RAMEND)
        OUT     SPH,    R21
        LDI     R21,    LOW(RAMEND)
        OUT     SPL,    R21

; CONFIGURACIÓN E INICIALIZACIÓN DE ESCRITURA

        CALL    I2C_BUS_INICIO
        CALL    I2C_INICIO
        LDI     R27,0xA0        //Dirección de Esclavo + Bit de Escritura  SLA+W
        CALL    I2C_ESCRIBIR
        LDI     R27,0x05
        CALL    I2C_ESCRIBIR
        LDI     R27,0x32
        CALL    I2C_ESCRIBIR
        CALL    I2C_PARADA

        CALL    RETARDO

; CONFIGURACIÓN E INICIALIZACIÓN DE LECTURA

        CALL    I2C_BUS_INICIO
        CALL    I2C_INICIO
        LDI     R27,0xA0
        CALL    I2C_ESCRIBIR
        LDI     R27,0x05
        CALL    I2C_ESCRIBIR
        CALL    I2C_INICIO
        LDI     R27,0xA1        //Dirección de Esclavo + Bit de Lectura  SLA+R
        CALL    I2C_ESCRIBIR
        CALL    I2C_LEER
        OUT     PORTA,R27
        CALL    I2C_PARADA
```

Programa 4.5: Escritura y Lectura de datos en Modo Maestro en Ensamblador (Viene de la página anterior)

```
LAZO1: RJMP   LAZO1

; SUBRUTINAS EMPLEADAS

I2C_BUS_INICIO:
        LDI     R21,    0
        OUT     TWSR,   R21
        LDI     R21,    0x47
        OUT     TWBR,   R21
        LDI     R21,    (1<<TWEN)
        OUT     TWCR,   R21
        RET

I2C_INICIO:
        LDI     R21,    (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
        OUT     TWCR,   R21

ESPERA_1:
        IN      R21,    TWCR
        SBRS    R21,    TWINT
        RJMP    ESPERA_1
        RET

I2C_LEER:
        LDI     R21,    (1<<TWINT)|(1<<TWEN)
        OUT     TWCR,   R21

ESPERA_2:
        IN      R21,    TWCR
        SBRS    R21,    TWINT
        RJMP    ESPERA_2
        IN      R27,    TWDR
        RET

I2C_ESCRIBIR:
        OUT     TWDR,   R27
        LDI     R21,    (1<<TWINT)|(1<<TWEN)
        OUT     TWCR,   R21

ESPERA_3:
        IN      R21,    TWCR
        SBRS    R21,    TWINT
        RJMP    ESPERA_3
        RET

I2C_PARADA:
        LDI     R21,    (1<<TWINT)|(1<<TWSTO)|(1<<TWEN)
        OUT     TWCR,   R21
        RET
```

Programa 4.5: Escritura y lectura de datos en Modo Maestro (Continúa en la siguiente página)

Programa 4.5: Escritura y Lectura de datos en Modo Maestro (Viene de la página anterior)

```
;--------------- RETARDO DE 1/4 DE SEGUNDO----------------

RETARDO:
        LDI     R21,    200
D1:     LDI     R22,    250
D2:     NOP
        NOP
        DEC     R22
        BRNE    D2
        DEC     R21
        BRNE    D1
        RET

; FIN DEL PROGRAMA
```

4.4.5.8 Simulación

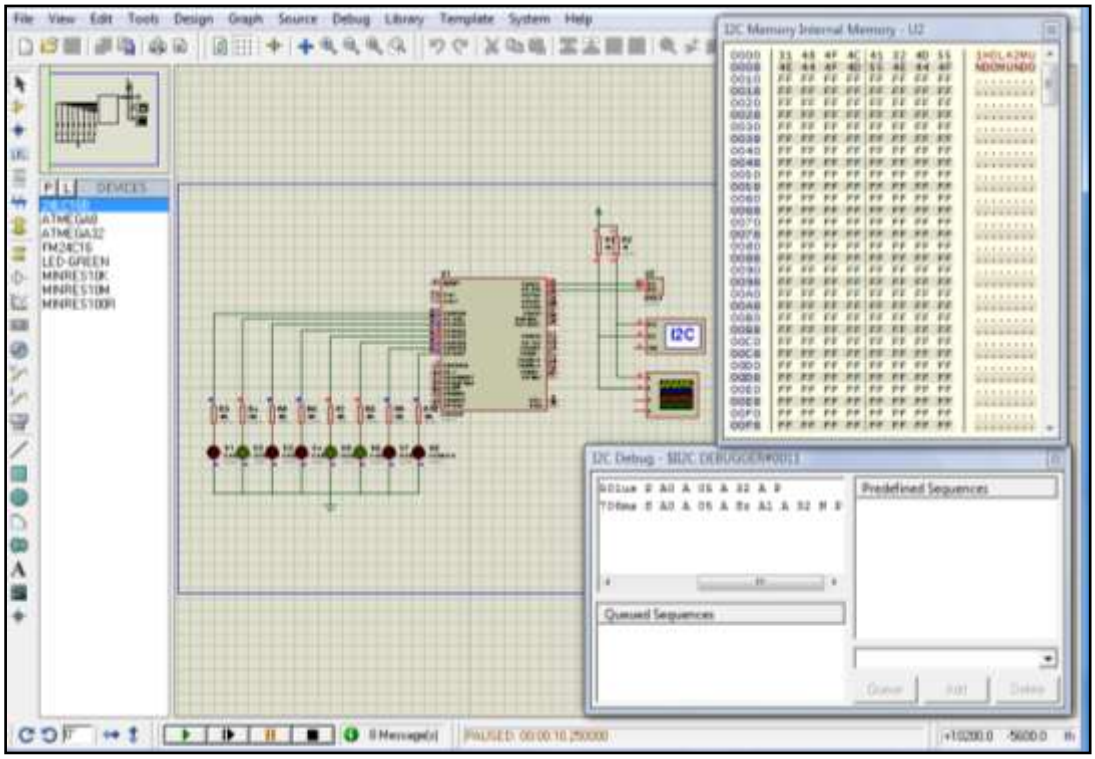


Figura 4.11 Simulación en Proteus programación en Ensamblador para Escritura y Lectura de Datos

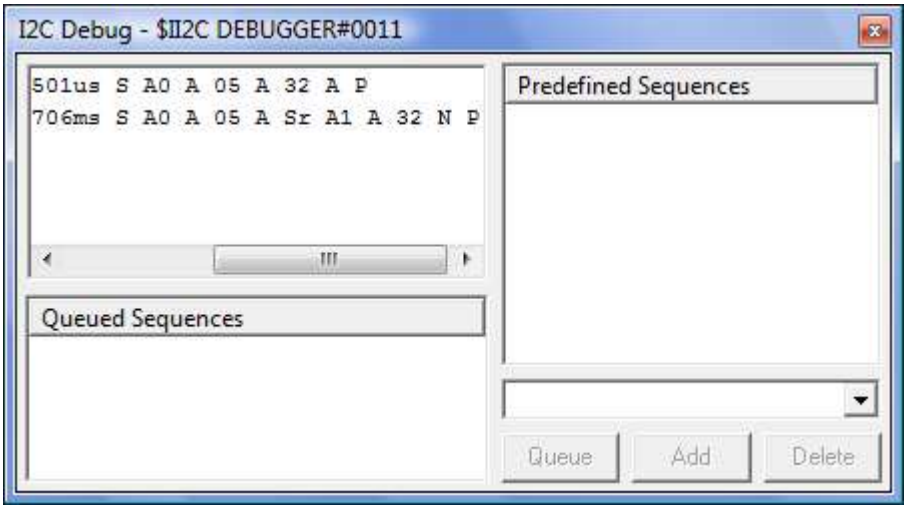


Figura 4.12 Ventana del Depurador para Escritura y lectura en Transmisión I2C en Proteus

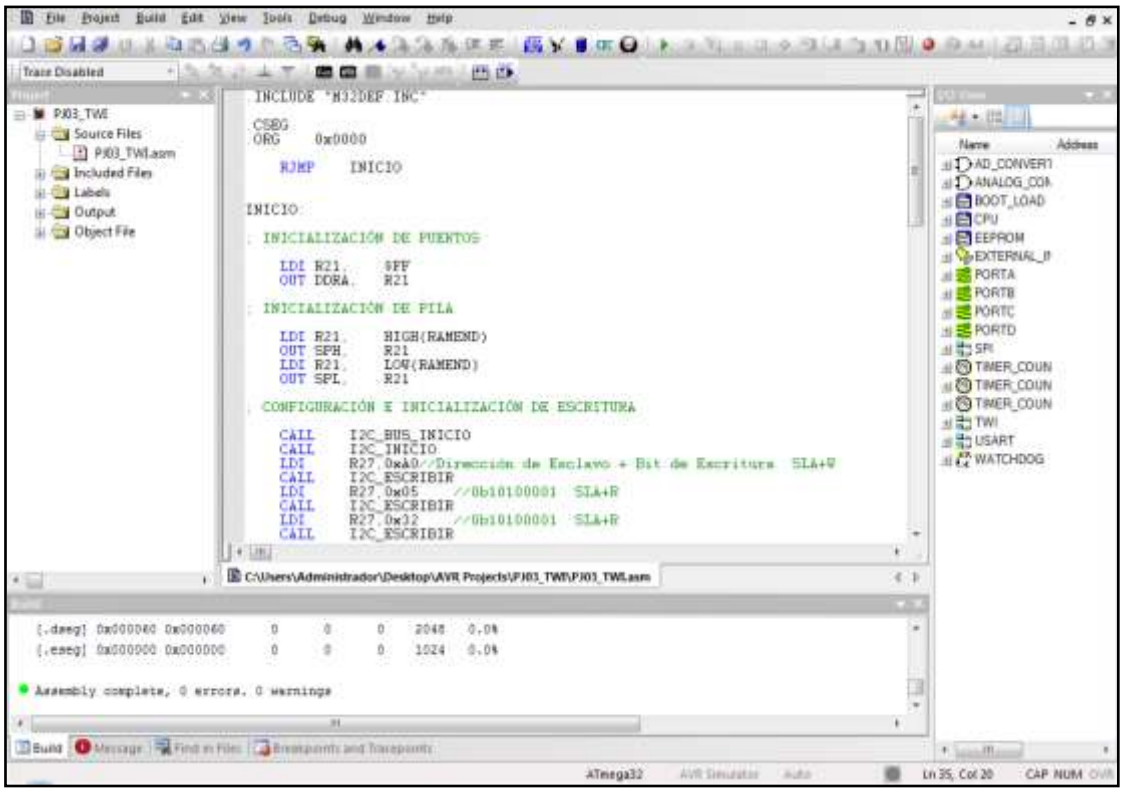


Figura 4.13 AVR-Studio Programación en Ensamblador para Escritura y Lectura de Datos

4.4.6 EJERCICIO 6

4.4.6.1 Tema

Programación en Lenguaje C del módulo TWI configurado en Modo Maestro para Escritura / Lectura

4.4.6.2 Objetivos

- Exponer el procedimiento para configurar e inicializar el módulo TWI en lenguaje Ensamblador.
- Escribir el código fuente haciendo uso del compilador AVR-Studio y la Plataforma WINAVR, para acceder a los registros del módulo TWI.
- Escribir y Leer bytes de datos a través del bus I2C por medio de un esclavo direccionado.

4.4.6.3 Descripción

Versión en lenguaje C del modo de Transmisión Maestro para Lectura / Escritura por bus I2C. En este modo se transmite un cierto número de bytes de datos a un Esclavo Receptor (SR), dependiendo del octavo bit del primer byte R/W se determina si debe estar habilitado para Lectura o Escritura de datos.

- **Inicialización**

  Para inicializar el módulo TWI para operar en modo Maestro, se debe seguir los siguientes pasos:

1. Habilitar la frecuencia de reloj del módulo TWI, configurando los valores del registro TWBR y los bits TWPS [1:0] en el registro TWSR.
2. Habilitar el módulo TWI, configurando a uno el bit TWEN en el registro TWCR.

- **Transmisión de Condición de Inicio**

Para iniciar la transferencia de datos en operación de modo Maestro, es necesario transmitir una condición de Inicio (Start). Esto es realizado por la configuración de los registros TWEN, TWSTA, y al poner en uno el bit TWINT del registro TWCR. Configurando el bit TWEN a uno habilita el módulo TWI. Configurando el bit TWSTA a uno permite al módulo TWI enviar una condición de inicio cuando el bus se encuentra libre, y configurando a uno el bit TWINT limpia la bandera de interrupción para iniciar la operación del módulo TWI transmitiendo la condición de Inicio. Además se debe sondear la bandera TWINT en el registro TWCR para conocer si la condición de Inicio ha sido transmitida completamente.

- **Recepción de Datos**

Para recibir un byte de Datos, luego de transmitir el byte SLA+R, debemos realizar los siguientes pasos:

1. Configurar a uno los bits TWEN y TWINT del registro TWCR para iniciar la recepción de un byte de datos. Notar que si es necesario retornar un Acuse de Recibo (ACK) después de la recepción de datos

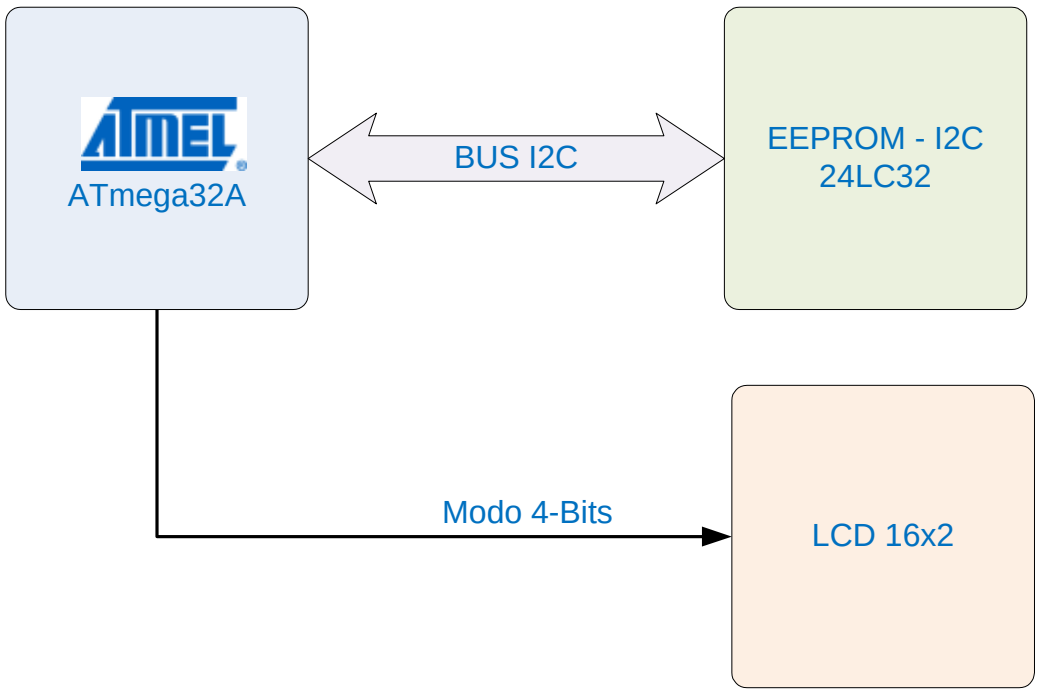
es necesario también configurar a uno el bit TWEA del registro TWCR.

2. Revisar la bandera TWINT en el registro TWCR para conocer si un byte ha sido completamente recibido.
3. Copiar el byte de datos recibido desde el registro TWDR a otro registro como respaldo del mismo.

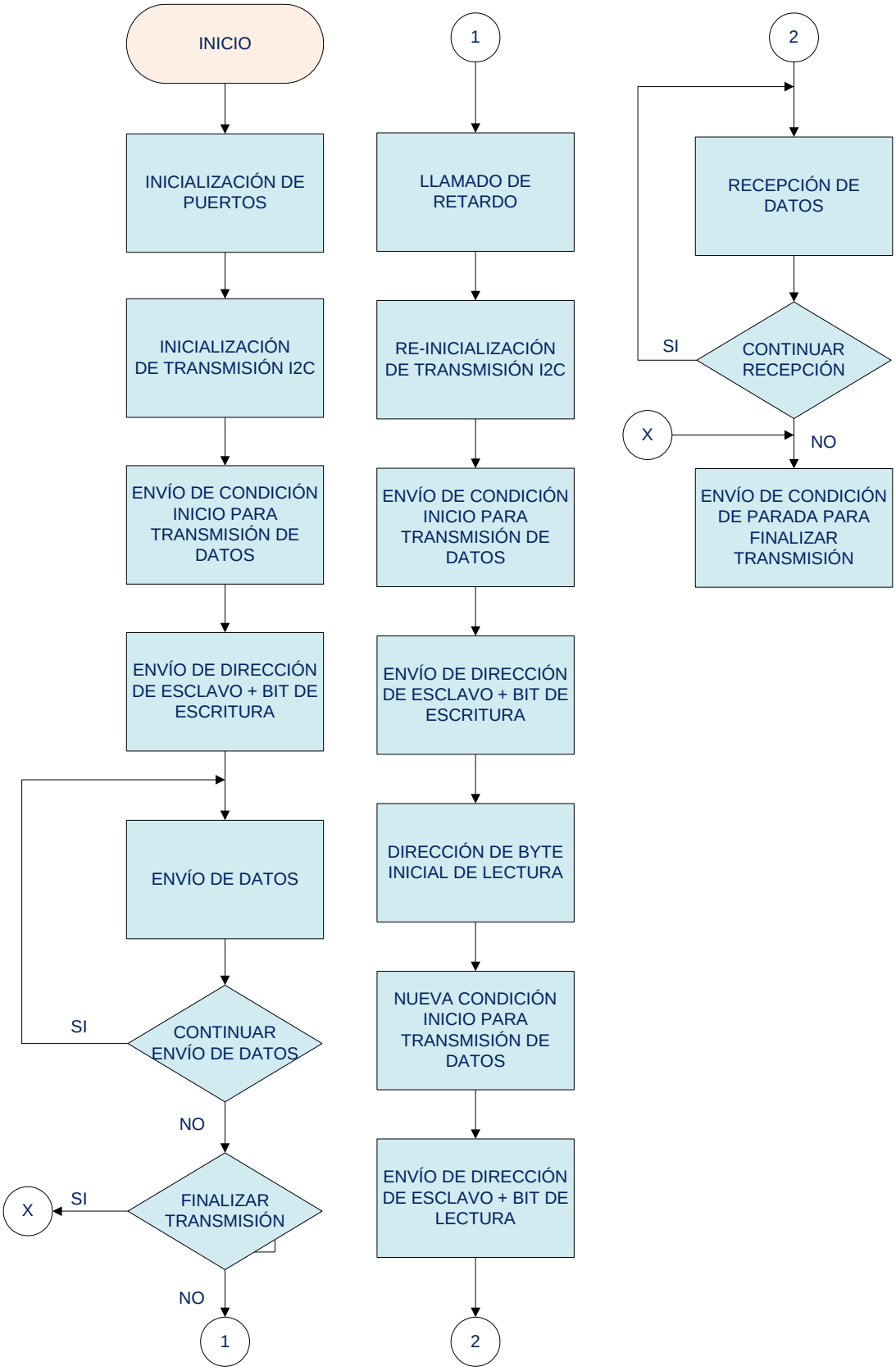
- **Transmisión de Condición de Parada**

  Para detener la transferencia de datos, debemos transmitir una Condición de Parada (Stop). Esto se realiza configurando los registros TWEN, TWSTO, y configurando a uno el bit TWINT del registro TWCR.

4.4.6.4 Diagrama de Bloques



4.4.6.5 Diagrama de Flujo



4.4.6.6 Descripción del Algoritmo

Basándose en el diagrama de bloques mostrado, podemos dar la siguiente descripción:

1. Se declara el Inicio del Programa Principal
2. Para trabajar en modo Maestro debe ser inicializado el bus I2C
3. Transmitir una condición de Inicio (Start)
4. Enviar la dirección para el reconocimiento del Esclavo y el bit de Escritura
5. Transmitir los datos y
6. Transmitir una condición de Parada (Stop) solo en caso de ser necesario finalizar la transmisión.
7. Se emplea un retardo de ¼ de segundo para un oscilador de 8Mhz, para estabilizar el bus I2C
8. Se repiten los pasos 3,4 y 5
9. Enviar la dirección del byte inicial para el proceso de Lectura
10. Se envía una nueva condición de Inicio para cambiar a estado de Lectura
11. Enviar la dirección para el reconocimiento del Esclavo y el bit de Lectura
12. Recibir los datos y
13. Transmitir una condición de Parada (Stop) final para liberar el bus y culminar en su totalidad la transmisión.

4.4.6.7 Código Fuente

Programa 4.6: Muestra como un Maestro Escribe y Lee datos por medio de un Esclavo direccionado en C (Continúa en la siguiente página)

```
#include <avr/io.h>
#include <util/delay.h>
#include "lcd.h"

//************************************************************
void i2c_bus_inicio(void){

        TWSR = 0x00;
        TWBR = 0x48;
        TWCR = 0x04;

}

//************************************************************
void i2c_inicio(void){

        TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
        while (!(TWCR & (1<<TWINT)));

}

//************************************************************
void i2c_escribir(unsigned char data){

        TWDR = data;
        TWCR = (1<<TWINT)|(1<<TWEN);
        while (!(TWCR & (1<<TWINT)));

}

//************************************************************
unsigned char i2c_leer(unsigned char ackVal){

        TWCR = (1<<TWINT)|(1<<TWEN)|(ackVal<<TWEA);
        while (!(TWCR & (1<<TWINT)));
        return TWDR;

}

//************************************************************
void i2c_parada(){

        TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);
        for( int k=0; k<100; k++);

}

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

Programa 4.6: Escritura y Lectura de datos en Modo Maestro en C (Viene de la página anterior)

```
int main (void){

unsigned int i=0;
char str[12];

        LCDInit(LS_BLINK);
        DDRB = 0xFF;

        LCDClear();
        LCDWriteString("Transmision I2C");
        LCDWriteStringXY(0,1,"AVR ATMEL");

        _delay_ms(8000);

        LCDClear();
        LCDWriteString("EEPROM Externa");
        LCDWriteStringXY(0,1,"Test...");

        _delay_ms(8000);

        LCDClear();
        LCDWriteString("Escribiendo...");

        i2c_bus_inicio();
        i2c_inicio();
        i2c_escribir(0b10101000);           //0b10101000
        i2c_escribir(0x00);
        i2c_escribir(0x48);
        i2c_escribir(0x4F);
        i2c_escribir(0x4C);
        i2c_escribir(0x41);
        i2c_escribir(0x16);
        i2c_escribir(0x4D);
        i2c_escribir(0x55);
        i2c_escribir(0x4E);
        i2c_escribir(0x44);
        i2c_escribir(0x4F);
        i2c_parada();

        _delay_ms(10000);

        LCDClear();
        LCDWriteString("Leyendo...");
        i2c_bus_inicio();
        i2c_inicio();
        i2c_escribir(0b10101000);
        i2c_escribir(0x00);
        i2c_inicio();
        i2c_escribir(0b10101001);
```

Programa 4.6: Muestra como un Maestro Escribe y Lee datos por medio de un Esclavo direccionado en C (Continúa en la siguiente página)

Programa 4.6: Escritura y Lectura de datos en Modo Maestro en C (Viene de la página anterior)

```
for (i=0; i<10; i++)
{
        str[i] = i2c_leer(1);
}

        i2c_parada();

        _delay_ms(8000);

        LCDClear();
        LCDWriteString("Msj Recibido:");
        LCDWriteStringXY(0,2,str);

}

//Fin del Programa
```

4.4.6.8 Simulación

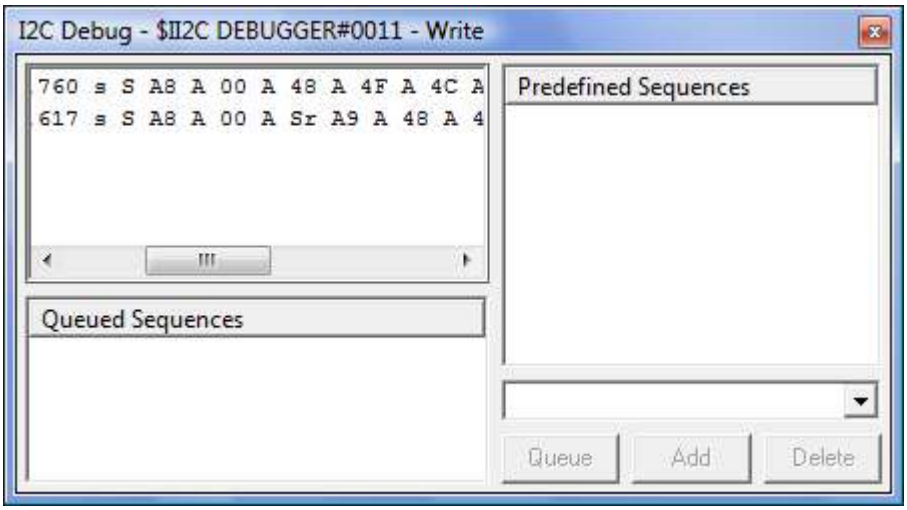


Figura 4.14 Ventana del Depurador para Escritura y Lectura en C para Transmisión I2C en Proteus

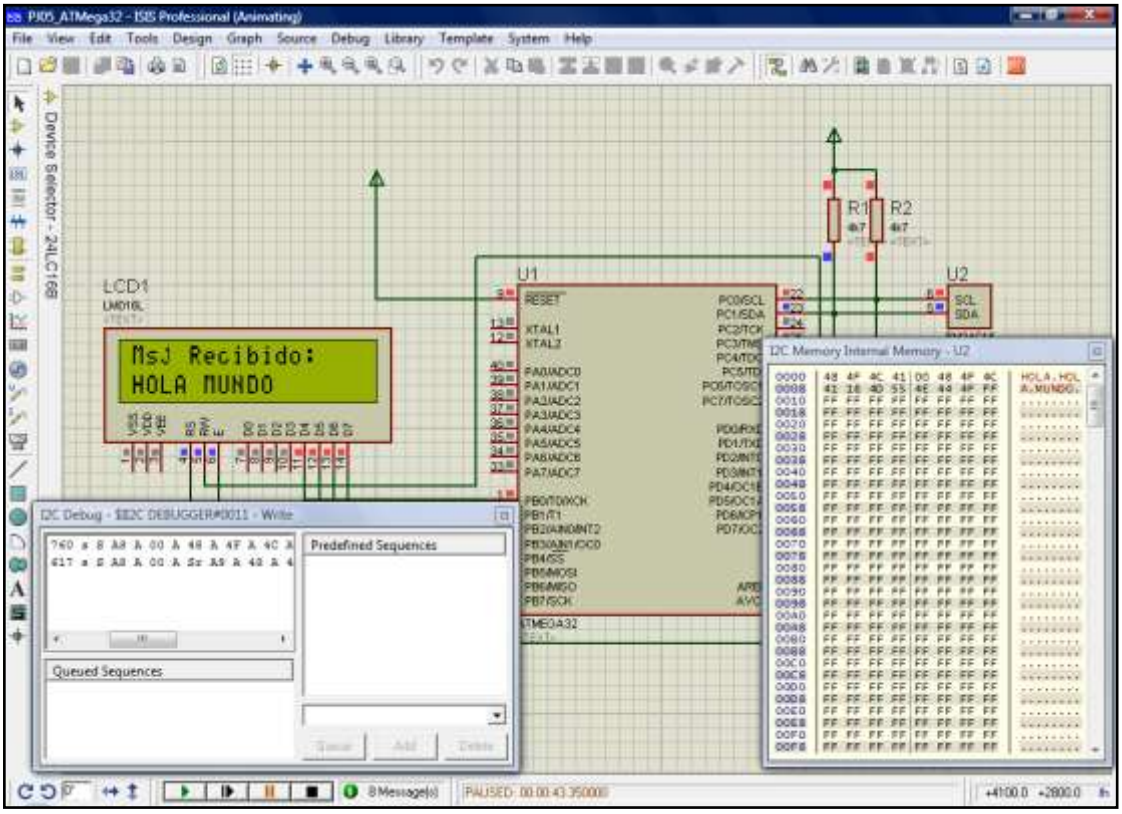


Figura 4.15 Simulación en Proteus programación en C para Escritura y Lectura de Datos

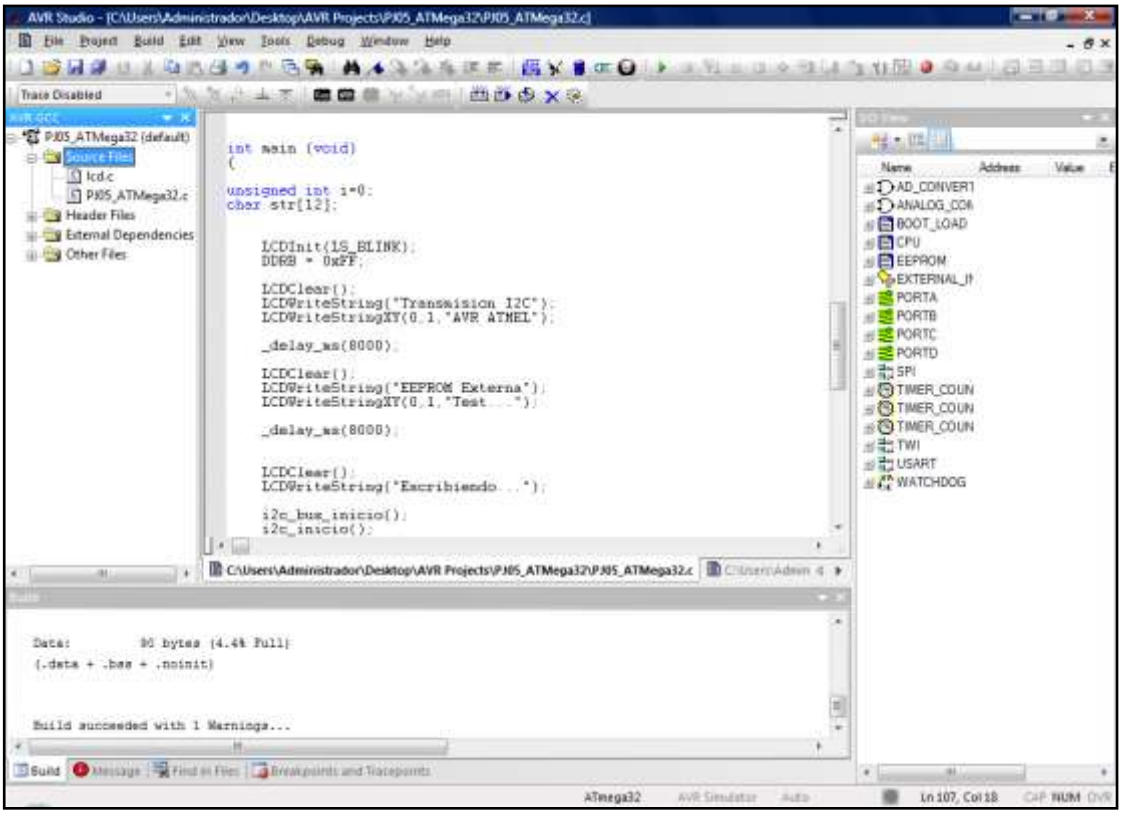


Figura 4.16 AVR-Studio Programación en C para Escritura y Lectura de Datos

4.4.7 EJERCICIO 7

4.4.7.1 Tema

Interfaz I2C, en Lenguaje C, para Reloj en Tiempo Real DS1307.

4.4.7.2 Objetivos

- Exponer el procedimiento para configurar e inicializar el módulo TWI en lenguaje C.
- Escribir el código fuente haciendo uso del compilador AVR-Studio y la Plataforma WINAVR, para acceder a los registros del módulo TWI.
- Acceder y configurar un dispositivo compatible con el estándar I2C.

4.4.7.3 Descripción

Programa en lenguaje C, que hace uso del modo de Transmisión Maestro para Lectura / Escritura de datos por bus I2C. En este modo se configura e inicializa el Esclavo Receptor (RTC DS1307), con el fin de obtener un reloj programado y continuo en tiempo real.

- **Escribiendo datos al DS1307**

  Para configurar el valor del puntero de registros y escribir uno o mas bytes de datos al DS1307, se debe seguir los siguientes pasos:

1. Para acceder al DS1307 por una operación de escritura, después de enviar una condición de Inicio (Start), se debe transmitir la dirección en modo de escritura del DS1307 (0xD0 por ser escritura).
2. El primer byte del dato en la operación de escritura configurará el puntero de registros. Por ejemplo si es necesario acceder al registro de control se debe enviar el valor 0x07.
3. Si es necesario escribir uno o mas bytes de datos, se debe transmitir un byte a la vez. Recordando que el puntero de registros se incrementa automáticamente en una posición y simplemente se debe transmitir los bytes de datos a locaciones consecutivas en una escritura continua de bytes de datos.
4. Finalmente transmitir una condición de Parada (Stop).

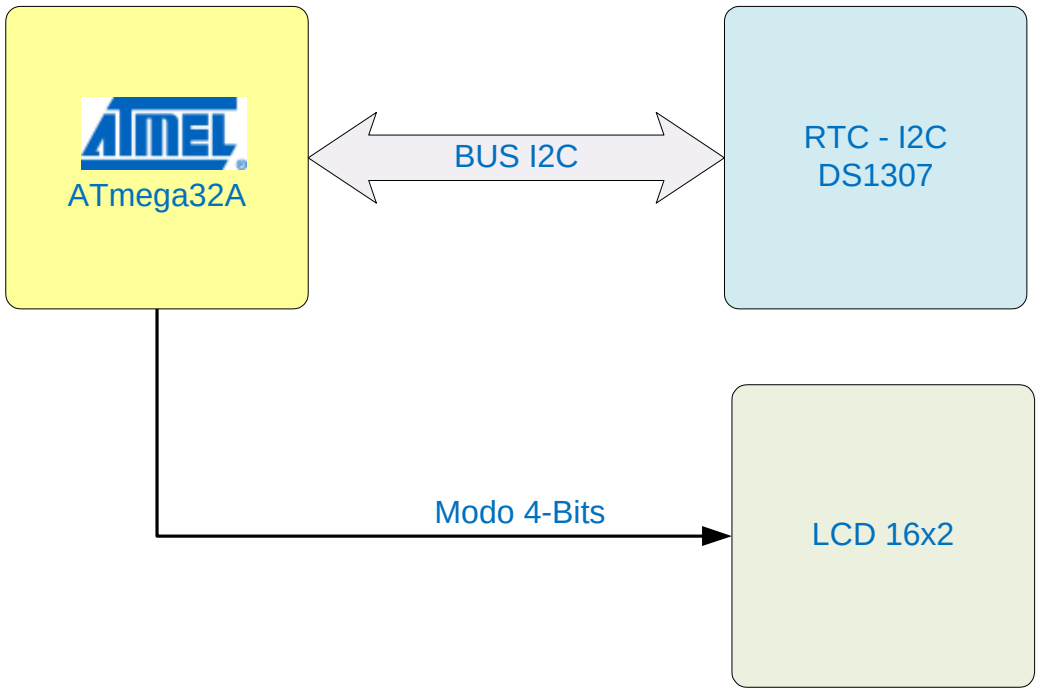
- **Leyendo datos desde el DS1307**

Observar que antes de la lectura de un byte se debe cargar la dirección del mismo al puntero de registros, como se mencionó en el proceso de escritura. Para leer uno o mas bytes de datos desde el DS1307, se debe seguir los siguientes pasos:

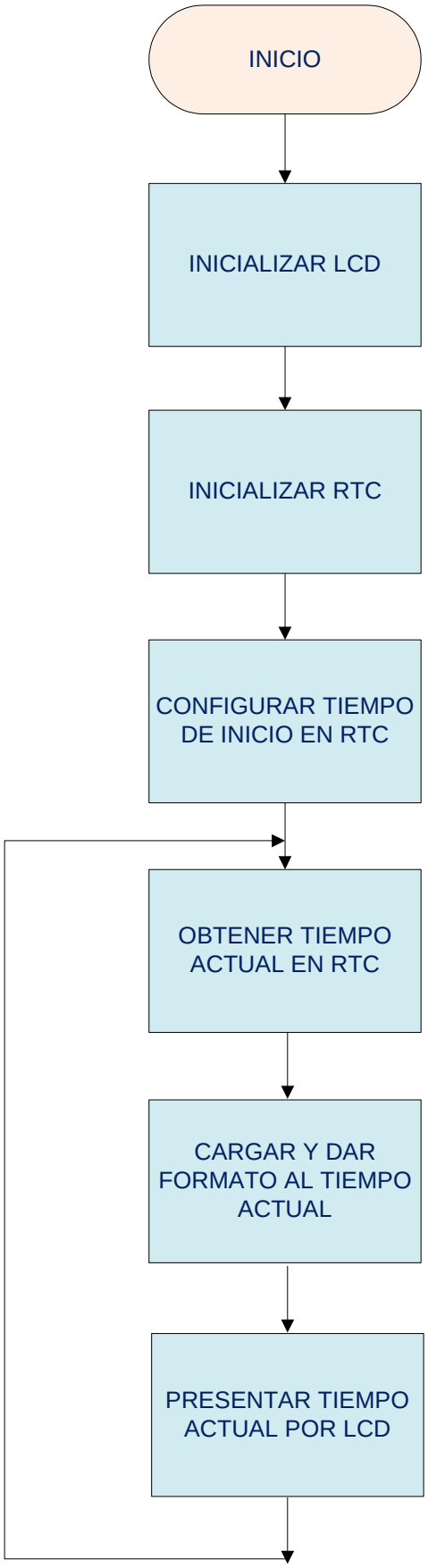
1. Para acceder al DS1307 por una operación de lectura, después de enviar una condición de Inicio (Start), se debe transmitir la dirección en modo de lectura del DS1307 (0xD1 por ser lectura).

2. A continuación es posible leer uno o mas bytes de datos. Recordar que el puntero de registros indica la dirección del registro que será accedido para su respectiva lectura. Notar también que el puntero de registros se incrementa automáticamente en una posición y simplemente se debe recibir bytes de datos consecutivos en una lectura continua de datos.
3. Finalmente transmitir una condición de Parada (Stop).

4.4.7.4 Diagrama de Bloques



4.4.7.5 Diagrama de Flujo



4.4.7.6 Descripción del Algoritmo

Basándose en el diagrama de bloques mostrado, podemos dar la siguiente descripción:

1. Se declara el Inicio del Programa Principal
2. Se inicializa la pantalla LCD o display
3. Se inicializa el dispositivo RTCDS1307
4. Se configura el tiempo inicial desde el que comenzará a trabajar el reloj en tiempo real
5. Transmisión de datos
6. Lectura y actualización de tiempo desde el dispositivo esclavo hacia el maestro
7. Se emplea un retardo de ¼ de segundo para un oscilador de 8Mhz, para estabilizar el bus I2C
8. Se da formato a los datos recibidos para su posterior presentación por pantalla LCD o display
9. Se presenta el tiempo actualizado por pantalla LCD o display
10. Se repiten pasos 6, 7, 8 y 9.

4.4.7.7 Código Fuente

Programa 4.7: Muestra como un Maestro Escribe y Lee datos hacia un Esclavo RTC-DS1307 en C (Continúa en la siguiente página)

```
#include <avr/io.h>
#include <util/delay.h>

#include "lcd.h"


//*************************************************************

void i2c_bus_inicio(void){

        TWSR = 0x00;
        TWBR = 0x48;
        TWCR = 0x04;

}

//*************************************************************

void i2c_inicio(void){

        TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
        while (!(TWCR & (1<<TWINT)));

}

//*************************************************************

void i2c_escribir(unsigned char data){

        TWDR = data;
        TWCR = (1<<TWINT)|(1<<TWEN);
        while (!(TWCR & (1<<TWINT)));

}


//*************************************************************

unsigned char i2c_leer(unsigned char ackVal){

        TWCR = (1<<TWINT)|(1<<TWEN)|(ackVal<<TWEA);
        while (!(TWCR & (1<<TWINT)));
        return TWDR;

}

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

Programa 4.7: Escritura y Lectura de datos en Modo Maestro hacia el RTC-DS1307 en C (Viene de la página anterior)

```
void i2c_parada(){

        TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);
        for( int k=0; k<100; k++);
}
//************************************************************

void rtc_Configurar_Tiempo(unsigned char h, unsigned char m, unsigned char s){

        i2c_bus_inicio();
        i2c_inicio();
        i2c_escribir(0xD0);
        i2c_escribir(0);
        i2c_escribir(s);
        i2c_escribir(m);
        i2c_escribir(h);
        i2c_parada();
}
//************************************************************

void rtc_Configurar_Fecha(unsigned char y, unsigned char m, unsigned char d){

        i2c_inicio();
        i2c_escribir(0xD0);
        i2c_escribir(0x04);
        i2c_escribir(d);
        i2c_escribir(m);
        i2c_escribir(y);
        i2c_parada();
}
//************************************************************

void rtc_Obtener_Tiempo(unsigned char *h, unsigned char *m, unsigned char *s){

        i2c_bus_inicio();
        i2c_inicio();
        i2c_escribir(0xD0);
        //i2c_escribir(0x04);
        i2c_escribir(0);
        i2c_parada();

        i2c_bus_inicio();
        i2c_inicio();
        i2c_escribir(0xD1);
        *s = i2c_leer(1);
        *m = i2c_leer(1);
        *h = i2c_leer(0);
        i2c_parada();
}
```

Programa 4.7: Escritura y lectura de datos en Modo Maestro hacia el RTC-DS1307 (Continúa en la siguiente página)

Programa 4.7: Escritura y Lectura de datos en Modo Maestro hacia el RTC-DS1307 en C (Viene de la página anterior)

```
void rtc_Obtener_Fecha(unsigned char *y, unsigned char *m, unsigned char *d){

	i2c_inicio();
	i2c_escribir(0xD0);
	i2c_escribir(0x04);
	i2c_escribir(0);
	i2c_parada();

	i2c_inicio();
	i2c_escribir(0xD1);
	*d = i2c_leer(1);
	*m = i2c_leer(1);
	*y = i2c_leer(0);
	i2c_parada();
}
//************************************************************

void rtc_Inicializar()
{
	//Inicializa interfaz del RTC DS1307
	#define CH 7

	uint8_t temp;

	i2c_bus_inicio();
	i2c_inicio();
	i2c_escribir(0xD1);
	temp = i2c_leer(0);
	i2c_parada();

	temp&=(~(1<<CH));

	i2c_bus_inicio();
	i2c_inicio();
	i2c_escribir(0xD0);
	i2c_escribir(0x07);
	i2c_escribir(0x90);
	i2c_parada();

	i2c_bus_inicio();
	i2c_inicio();
	i2c_escribir(0xD0);
	i2c_escribir(0x00);
	i2c_escribir(temp);
	i2c_parada();
```

Programa 4.7: Escritura y lectura de datos en Modo Maestro hacia el RTC-DS1307 (Continúa en la siguiente página)

Programa 4.7: Escritura y Lectura de datos en Modo Maestro hacia el RTC-DS1307 en C (Viene de la página anterior)

```
//Configuramos formato 24horas
        i2c_bus_inicio();
        i2c_inicio();
        i2c_escribir(0xD0);
        i2c_escribir(0x02);
        i2c_inicio();
        i2c_escribir(0xD1);
        temp = i2c_leer(0);
        i2c_parada();
        temp|=(0b00000000);
        i2c_bus_inicio();
        i2c_inicio();
        i2c_escribir(0xD0);
        i2c_escribir(0x02);
        i2c_escribir(temp);
        i2c_parada();
        return 1;
}
//************************************************************
int main (void){
unsigned char hora, min, seg;

        LCDInit(LS_BLINK);
        LCDClear();
        LCDWriteString("RTC I2C");
        LCDWriteStringXY(0,1,"AVR ATMEL");
        _delay_ms(8000);
        LCDClear();
        LCDWriteString("DS1307 Externo");
        LCDWriteStringXY(0,1,"Test...");
        _delay_ms(8000);
        rtc_Inicializar();
        rtc_Configurar_Tiempo(0x22, 0x15, 0x20);

while (1){
        char Time[12];    //hh:mm:ss
        rtc_Obtener_Tiempo(&hora, &min, &seg);
        Time[10]='\0';
        Time[9]=48+(seg & 0b00001111);
        Time[8]=48+((seg & 0b01110000)>>4);
        Time[7]=':';
        Time[6]=' ';
        Time[5]=48+(min & 0b00001111);
        Time[4]=48+((min & 0b01110000)>>4);
        Time[3]=':';
        Time[2]=' ';
        Time[1]=48+(hora & 0b00001111);
        Time[0]=48+((hora & 0b00110000)>>4);
        LCDClear();
        LCDWriteString("Hor Min Seg");
        LCDWriteStringXY(0,2,Time);
        _delay_ms(800);
        }
        return 0;
}
```

4.4.7.8 Simulación

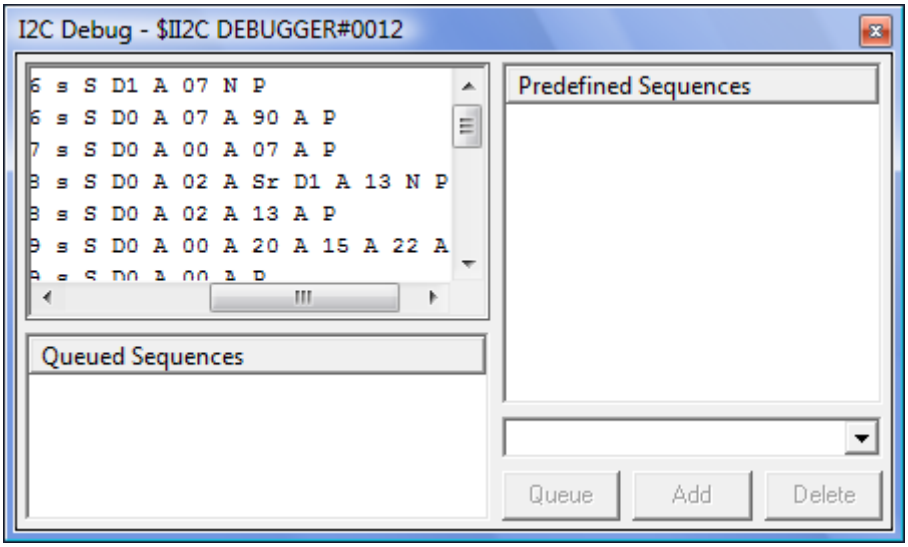


Figura 4.17 Ventana del Depurador para Escritura y Lectura al DS1307 en C para Transmisión I2C en Proteus

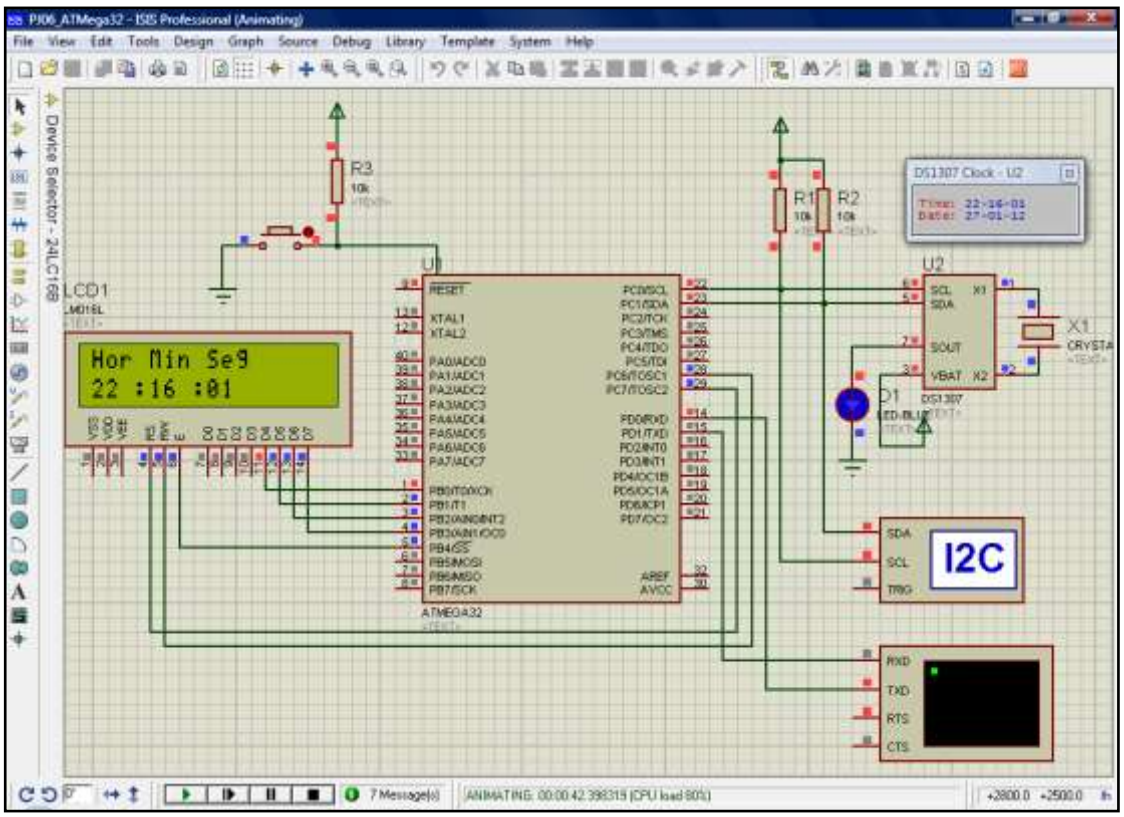


Figura 4.18 Simulación en Proteus programación en C para Escritura y Lectura de Datos hacia el DS1307

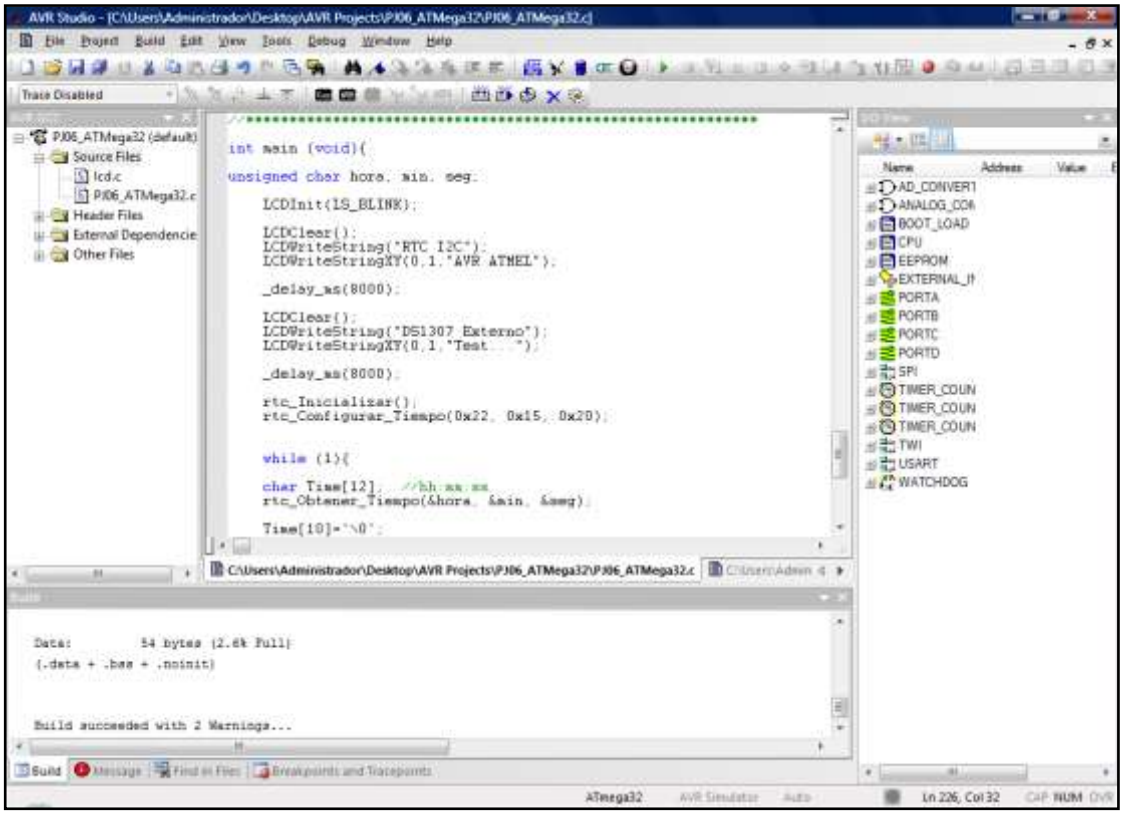


Figura 4.19 AVR-Studio Programación en C para Escritura y Lectura de Datos hacia el RTC-DS1307

CONCLUSIONES

1. Se ha presentado de forma conjunta las especificaciones técnicas y el procedimiento para llevar a cabo una transmisión haciendo uso del módulo TWI, a través de la interfaz I2C, observando así la versatilidad y las distintas ventajas que ofrece este tipo de comunicación por medio de dos hilos. Aunque para dicha comunicación se ha utilizado una EEPROM en la práctica existen diversos dispositivos que incluyen este tipo de interfaz siempre compatibles por su estandarización.

2. Ciertamente el nivel de seguridad establecido para la realización de una transmisión debe ser óptimo en toda transferencia, es por esto que se presenta como una alternativa válida un proceso de monitoreo del registro de Estado, en el módulo TWI método por el cual se puede establecer procedimientos para control de errores que se pueden presentar de manera fortuita en nuestra transferencia.

3. Al hacer uso de la EEPROM, su formato de lectura puede ser de cuatro formas: lectura de la dirección actual en el bus, lectura de una dirección cualquiera, lectura secuencial a partir de la dirección actual y lectura secuencial a partir de una dirección cualquiera. Se ha hecho uso de la forma más usual, la cual es la de leer una dirección cualquiera definida por el usuario, donde el proceso es muy similar al de escritura y tras una reinicialización hay dos ciclos donde se indica el modo de lectura y se envía el dato.

RECOMENDACIONES

1. Para cada proyecto a realizar, guardar todos los archivos con extensión .c y .h de manera conjunta, de esta forma conservaremos un orden adecuado, y a su vez, si es necesario reutilizar el código, lo podremos hacer sin ningún inconveniente. Con la ventaja de poder crear nuestro propio conjunto de librerías e instrucciones.

2. Debemos aprovechar el uso del paquete AVR Studio ya que provee un entorno de programación amigable, además posee herramientas que nos ayudan a configurar nuestro espacio de trabajo, de forma que, usando su Ayuda para el desarrollo de los proyectos nos ahorra tiempo de configurar las directivas básicas para el funcionamiento del MCU AVR.

3. Recordar que se cuenta con dos líneas que permiten realizar una comunicación I2C, la línea SCL (Señal de Reloj) y la línea SDA (Señal de Datos), y mediante ellas se define las diferentes operaciones que pueden establecerse a través de este Protocolo de 2-hilos.

ANEXOS

Librerías empleadas para uso de LCD en modo de 4-bits

LCD.c: Librería empleada para uso de LCD en modo de 4-bits (Continúa en la siguiente página)

```
#include <avr/io.h>
#include <inttypes.h>

#ifndef F_CPU
#define F_CPU 12000000UL
#endif

#include <util/delay.h>
#include "lcd.h"

#define LCD_DATA_PORT          PORT(LCD_DATA)
#define LCD_E_PORT             PORT(LCD_E)
#define LCD_RS_PORT      PORT(LCD_RS)
#define LCD_RW_PORT     PORT(LCD_RW)

#define LCD_DATA_DDR   DDR(LCD_DATA)
#define LCD_E_DDR              DDR(LCD_E)
#define LCD_RS_DDR             DDR(LCD_RS)
#define LCD_RW_DDR             DDR(LCD_RW)

#define LCD_DATA_PIN    PIN(LCD_DATA)

#define SET_E() (LCD_E_PORT|=(1<<LCD_E_POS))
#define SET_RS() (LCD_RS_PORT|=(1<<LCD_RS_POS))
#define SET_RW() (LCD_RW_PORT|=(1<<LCD_RW_POS))

#define CLEAR_E() (LCD_E_PORT&=(~(1<<LCD_E_POS)))
#define CLEAR_RS() (LCD_RS_PORT&=(~(1<<LCD_RS_POS)))
#define CLEAR_RW() (LCD_RW_PORT&=(~(1<<LCD_RW_POS)))
```

LCD.c: Librería empleada para uso de LCD en modo de 4-bits (Viene de la página anterior)

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

void LCDByte(uint8_t c,uint8_t isdata)
{
/*****************************************
        Envía un byte al LCD en modo de 4-bits
        cmd=0 para datos
        cmd=1 para comandos
*****************************************/
uint8_t hn,ln;
uint8_t temp;

        hn=c>>4;
        ln=(c & 0x0F);

if(isdata==0)
        CLEAR_RS();
else
        SET_RS();

        _delay_us(0.500);

        SET_E();

//Envía nibble alto

        temp=(LCD_DATA_PORT & 0XF0)|(hn);
        LCD_DATA_PORT = temp;
        _delay_us(1);
        CLEAR_E();
        _delay_us(1);

//Envía nibble bajo

        SET_E();
        temp=(LCD_DATA_PORT & 0XF0)|(ln);
        LCD_DATA_PORT=temp;
        _delay_us(1);


        CLEAR_E();
        _delay_us(1);
        LCDBusyLoop();
}

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

LCD.c: Librería empleada para uso de LCD en modo de 4-bits (Continúa en la siguiente página)

LCD.c: Librería empleada para uso de LCD en modo de 4-bits (Viene de la página anterior)

```
//************************************************************
void LCDBusyLoop()
{
	//Esta función espera si el módulo LCD está ocupado

	uint8_t busy,status=0x00,temp;

	//Cambiamos el puerto a Entrada por lectura de datos

	LCD_DATA_DDR&=0xF0;

	//Cambio de modo del LCD
	SET_RW();		//Modo Lectura
	CLEAR_RS();		//Lectura de Estado

	_delay_us(0.5);

	do
	{
		SET_E();

		_delay_us(0.5);
		status=LCD_DATA_PIN;
		status=status<<4;
		_delay_us(0.5);
		CLEAR_E();
		_delay_us(1);
		SET_E();
		_delay_us(0.5);
		temp=LCD_DATA_PIN;
		temp&=0x0F;
		status=status|temp;
		busy=status & 0b10000000;

		_delay_us(0.5);
		CLEAR_E();
		_delay_us(1);
	}while(busy);

	CLEAR_RW();		//Modo Escritura

	//Cambiamos pueto a Salida
	LCD_DATA_DDR|=0x0F;

}

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

LCD.c: Librería empleada para uso de LCD en modo de 4-bits (Continúa en la siguiente página)

LCD.c: Librería empleada para uso de LCD en modo de 4-bits (Viene de la página anterior)

```
void LCDInit(uint8_t style)
{
	/*********************************************************
	Esta función inicializa el módulo LCD
	se debe llamar antes que cualquier otra función del LCD.
	*********************************************************/

	_delay_ms(30);

	//Configuración de Puertos I/O
	LCD_DATA_DDR|=(0x0F);
	LCD_E_DDR|=(1<<LCD_E_POS);
	LCD_RS_DDR|=(1<<LCD_RS_POS);
	LCD_RW_DDR|=(1<<LCD_RW_POS);

	LCD_DATA_PORT&=0XF0;
	CLEAR_E();
	CLEAR_RW();
	CLEAR_RS();

	//Configura modo de 4-bits
	_delay_us(0.3);   //tAS

	SET_E();
	LCD_DATA_PORT|=(0b00000010);
	_delay_us(1);
	CLEAR_E();
	_delay_us(1);

	LCDBusyLoop();

	LCDCmd(0b00001100|style);        //Enciende Display
	LCDCmd(0b00101000);
}

//************************************************************

void LCDWriteString(const char *msg)
{
	/*********************************************************
	Esta función escribe una cadena de caracteres al LCD en
	la posición actual del cursor.
	*********************************************************/
 while(*msg!='\0')
 {
	LCDData(*msg);
	msg++;
 }
}
//************************************************************
```

LCD.c: Librería empleada para uso de LCD en modo de 4-bits (Continúa en la siguiente página)

LCD.c: Librería empleada para uso de LCD en modo de 4-bits (Viene de la página anterior)

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

void LCDWriteInt(int val,unsigned int camp_long)
{
	/*********************************************************

	Esta función escribe un valor entero al módulo LCD

	*********************************************************/

	char str[5]={0,0,0,0,0};
	int i=4,j=0;
	while(val)
	{
	str[i]=val%10;
	val=val/10;
	i--;
	}
	if(camp_long==-1)
		while(str[j]==0) j++;
	else
		j=5-camp_long;

	if(val<0) LCDData('-');
	for(i=j;i<5;i++)
	{
	LCDData(48+str[i]);
	}
}

void LCDGotoXY(uint8_t x,uint8_t y)
{
 if(x<40)
 {
  if(y) x|=0b01000000;
		x|=0b10000000;
		LCDCmd(x);
  }
}

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

LCD.h: Librería empleada para uso de LCD en modo de 4-bits (Continúa en la siguiente página)

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

#ifndef F_CPU
#define F_CPU 12000000UL
#endif

#include <util/delay.h>
#include "myutils.h"

#ifndef _LCD_H
#define _LCD_H

//*************************************************
                    Conexión LCD
//*************************************************
#define LCD_DATA B        //PB0-PB3 se debe conectar a D4-D7

#define LCD_E B           //Señal de Habilitación
#define LCD_E_POS PB4     //Posición del pin de la señal de Habilitación

#define LCD_RS C
#define LCD_RS_POS PC7

#define LCD_RW C
#define LCD_RW_POS PC6

//*************************************************

#define LS_BLINK 0B00000001
#define LS_ULINE 0B00000010

//*************************************************
                    FUNCIONES
//*************************************************
void LCDInit(uint8_t style);
void LCDWriteString(const char *msg);
void LCDWriteInt(int val,unsigned int field_length);
void LCDGotoXY(uint8_t x,uint8_t y);

void LCDByte(uint8_t,uint8_t);
#define LCDCmd(c) (LCDByte(c,0))
#define LCDData(d) (LCDByte(d,1))

void LCDBusyLoop();
```

LCD.h: Librería empleada para uso de LCD en modo de 4-bits (Viene de la página anterior)

```
/************************************************
        M A C R O S
************************************************/

#define LCDClear() LCDCmd(0b00000001)
#define LCDHome() LCDCmd(0b00000010);

#define LCDWriteStringXY(x,y,msg) {

        LCDGotoXY(x,y);\
        LCDWriteString(msg);\

}

#define LCDWriteIntXY(x,y,val,fl) {\

        LCDGotoXY(x,y);\
        LCDWriteInt(val,fl);\
}

/************************************************/

#endif
```

myutils.h: Librería empleada para manejo de puertos del MCU AVR (Viene de la página anterior)

```
#ifndef MYUTILS_H

/************************************************/

 #define MYUTILS_H

 #define _CONCAT(a,b) a##b
 #define PORT(x) _CONCAT(PORT,x)
 #define PIN(x) _CONCAT(PIN,x)
 #define DDR(x) _CONCAT(DDR,x)

/************************************************/

#endif
```

REFERENCIAS BIBLIOGRÁFICAS

[1] Atmel, ATmega32A Data Sheet Revision 8155C, http://www.atmel.com/Images/doc8155.pdf, fecha de consulta Julio 2011.

[2] AVR, LibC Reference Manual 1.4.4, http://www.nongnu.org/avr-libc/, fecha de consulta Julio 2011.

[3] Atmel, AVR315: Using the TWI module as I2C master Application Note. Revision 2564C, http://www.atmel.com/Images/doc2564.pdf, fecha de consulta Julio 2011.

[4] Atmel, AVR311: Using the TWI module as I2C slave Application Note. Revision 2565D 2009, http://www.atmel.com/Images/doc2565.pdf, fecha de consulta Julio 2011.

[5] Atmel, AVR245: Code Lock with 4x4 Keypad and I2C LCD Application Note. Revision 8013A 2005, http://www.atmel.com/Images/doc8013.pdf, fecha de consulta Julio 2011.

[6] Philips Semiconductors, THE I2C-BUS SPECIFICATION V2.1, http://i2c2p.twibright.com/spec/i2c.pdf, fecha de consulta Julio 2011.

[7] JMNLAB, Sensor de Ultrasonido I2C, http://jmnlab.com/i2c/i2c, fecha de consulta Agosto 2011.

[8] Mann, Richard, How to Program an 8-bit Microcontroller Using C Language, www.atmel.com, fecha de consulta Agosto 2011.

[9] AVR, Development Pages, http://www.nongnu.org/avr-libc/user-

manual/group__twi__demo.html, fecha de consulta Octubre 2011.

[10] AVR, AVRProg User Guide, www.atmel.com, fecha de consulta Octubre 2011.

[11] Barnette, M.D., AVR Simulation with the ATMEL AVR Studio 4, www.avr.freaks.com, fecha de consulta Octubre 2011.

[12] Svendlsi, Odd Jostein, Atmel's Self-Programming Flash Microcontrollers, www.atmel.com, fecha de consulta Octubre 2011.

[13] O'Flynn, Colin, Downloading, Installing and Configuring WinAVR, http://winavr.sourceforge.net/, fecha de consulta Octubre 2011.

[14] Snilsberg, Rolf Kristian, AVR064: A Temperature Monitoring System with LCD Output, www.atmel.com, fecha de consulta Octubre 2011.

[15] García, V, http://www.hispavila.com/3ds/atmega/i2c-eeprom.html, fecha de consulta Noviembre 2011.

[16] AVRBeginners, The AVR TWI (I^2C) Interface, http://www.avrbeginners.net/architecture/twi/twi.html, fecha de consulta Noviembre 2011.

[17] Miklebust, Gaute, The AVR Microcontroller and C Compiler Co-Design, www.atmel.com, fecha de consulta Noviembre 2011.

[18] Atmel, AT45DB041B DataFlash, www.atmel.com, fecha de consulta Noviembre 2011.

[19] AVR, AVR ScienceProg, http://winavr.scienceprog.com/example-avr-projects, fecha de consulta Noviembre 2011.

[20] Shawn, Johnson, http://www.cursomicros.com/avr/i2c-eeprom/eeprom-24xxx.html, fecha de consulta Noviembre 2011.