ESCUELA DE INGENIERÍA INFORMÁTICA

Grado en Ingeniería Informática

PERIFÉRICOS E INTERFACES

PRÁCTICAS DE LABORATORIO

Módulo 2 – Práctica 3

El Bus I2C

Actualizado: 29/10/2023

Tabla de contenido

1. Competencias y objetivos de la práctica

La tercera práctica de la asignatura Periféricos e Interfaces tiene como objetivo principal reforzar y completar los conocimientos teóricos adquiridos en el módulo 2, en la parte dedicada a los buses de interconexión. Para alcanzar este propósito lo que se propone es la conexión de un módulo I2C usando un puerto de entrada/salida paralelo de propósito general. Concretamente, se utilizará como primer elemento a conectar el módulo de memoria 24LC64 de 8KBytes, 64kbit (en sustitución de M24C01 de 128 bytes de capacidad - 1 kbit o el M24C02 de 256 bytes, 2 Kbit, usados otros años, porque era el que estaba disponible en la tarjeta experimental en concreto).
Para realizar la práctica se usará una tarjeta "*Arduino Mega*" y el circuito de memoria 24LC64 adecuadamente montado en una tarjeta experimental de laboratorio pero usando el simulador Proteus 8.11; el esquema eléctrico se detalla en esta memoria y en la correspondiente explicación y análisis llevados a cabo en la sesión de presentación de la práctica. Como entorno de desarrollo se utiliza el Arduino IDE (dentro del proteus).

La práctica tiene como objetivo darle a las competencias adquiridas una aplicación práctica lo más cerca posible al mundo real. Así, se desarrollará y experimentará con el protocolo del bus I2C, para profundizar en su conocimiento y conseguir dominar su uso básico. Para esto se tendrá que implementar a través de los terminales (o *pines*) de entrada/salida del Arduino (que se especifican más adelante) el protocolo de comunicación correspondiente al bus síncrono I2C y comprobar que las operaciones de lectura y escritura sobre el dispositivo externo se realizan correctamente. Se utilizarán habilidades de programación y acceso a las señales físicas.

Es tarea del estudiante planificar y desarrollar el software de bajo nivel para programar funcionamiento de estos dispositivos y realizar las transferencias de datos sincronizando la actividad de comunicación y el almacenamiento y lectura en la memoria I2C. Con la realización de esta práctica el alumno alcanzará las siguientes competencias:

1. Capacidad para entender los diversos aspectos software y hardware involucrados en la gestión de un bus serie de entrada salida.
2. Capacidad para diseñar e implementar el software necesario para la gestión mediante un procesador de propósito general del acceso a un periférico I2C.
3. Capacidad para hacer uso de la información aportada por un fabricante de un dispositivo periférico real para utilizarlo en la resolución de un problema práctico real.
4. Capacidad para aprender y aplicar nuevos conceptos de forma autónoma e interdisciplinar.
5. Capacidad para emplear la creatividad en la resolución de los problemas.

Para alcanzar estas competencias se plantea la consecución de los siguientes objetivos:

1. Conocer y usar el bus I2C y su protocolo de comunicación básico.
2. Conocer y entender los aspectos básicos de funcionamiento de los periféricos I2C y, concretamente, la memoria 24LC64 a partir de la hoja de características proporcionada por el fabricante.
3. Realizar rutinas sencillas que permitan operaciones de escritura y lectura en cualquier dirección de memoria del dispositivo y en los distintos modos soportados.
4. Verificar el correcto funcionamiento del hardware y del software de comprobación desarrollado. Observación de las señales, si fuese necesario.
5. Planificar el desarrollo software completo de la segunda parte de la práctica. Expresarlo en un organigrama general que refleje la secuencia de acciones necesarias a realizar.

2. Documentación

La documentación básica a manejar en la realización de esta práctica está disponible en la plataforma Moodle institucional, y se concreta en:

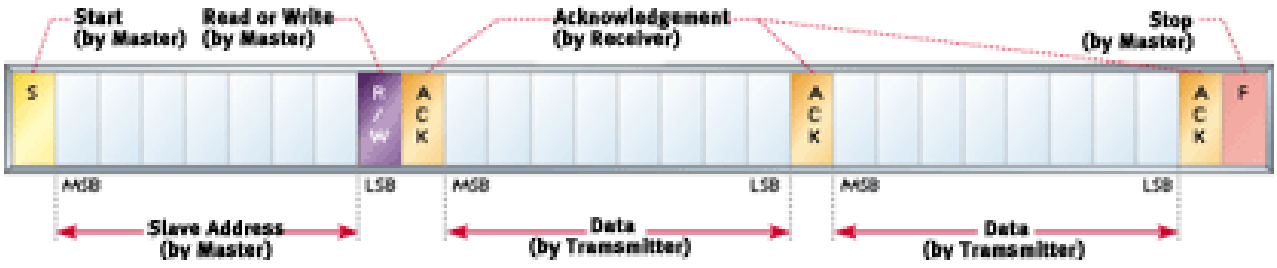
- Enunciado de la práctica (*este documento*)
- Material explicado en clase de teoría (*Bus-I2C.pdf*) y prácticas: "El bus I2C" (*23_24-PI_Lab3_Presentacion-I2C-V4.pdf*)
- Documentos varios relativos al funcionamiento del bus I2C
- Documentación del fabricante de la Memoria 24LC64 (datasheet-24LC64-21189f.pdf).
- Documentación (*http://arduino.cc/en/Reference/HomePage*) del entorno de programación del Arduino.
- Documentación (*BusI2C.pdf*) de la Práctica y del "*Arduino Mega*" (*https://store.arduino.cc/arduino-mega-2560-rev3*), y en especial:
    - Esquema de conexiones
    - Técnicas de acceso a los Puertos Digitales y canal serie

3. Conceptos previos

En las Clases de Teoría han sido explicados los conceptos previos sobre el bus I2C. Concretamente se han detallado los criterios de uso y características del bus, así como algunos ejemplos de uso que superan incluso a los que utilizaremos en esta práctica. En la presentación de la práctica, además se planteó un bosquejo de cómo se podía utilizar el hardware de la práctica 2 para controlar un dispositivo I2C. Esto está recogido en los documentos citados anteriormente.

Aquí se continuará, desde ese punto, detallando el esquema del circuito a utilizar en el desarrollo de la práctica 2. Previamente interesa dedicar espacio a hacer un repaso de los conceptos necesarios y relacionarlo con el detalle concreto del dispositivo que vamos a utilizar.

Es muy importante tener conocimiento claro y detallado del funcionamiento del bus I2C. Si repasamos el documento de la explicación en clase, encontramos un ejemplo típico de comunicación:

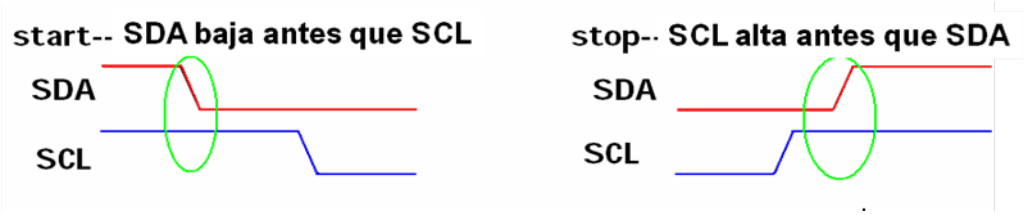


Recuérdese, que nosotros vamos a utilizar un entorno muy sencillo, donde solo existirá un "master" que será siempre nuestro Procesador Arduino. A partir de ese momento sabemos que nosotros vamos a iniciar la comunicación (generando una condición "start") y que también vamos a terminarla (generando la condición de "stop") para dejar el bus libre. No va a existir otro Maestro del bus, lo cual nos simplifica la gestión.

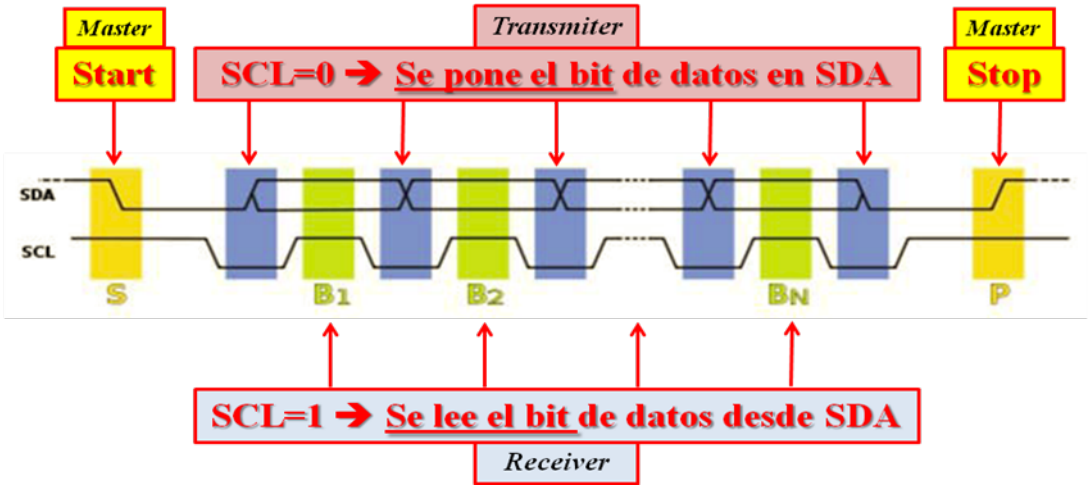
OPERACIONES DE START Y STOP

El Maestro adquiere el uso del bus enviando la "condición de START". Antes de hacerlo debería asegurarse de que el bus está libre: SDA="1" y SCL="1". Esto lo haría poniendo SDA,SCL="1,1" e inmediatamente leyendo SDA y SCL y comprobando que efectivamente están a 1,1 (¿SDA,SCL ="1,1" ?). Si se cumple esta condición el bus puede ser tomado por el Maestro enviando la condición de START.

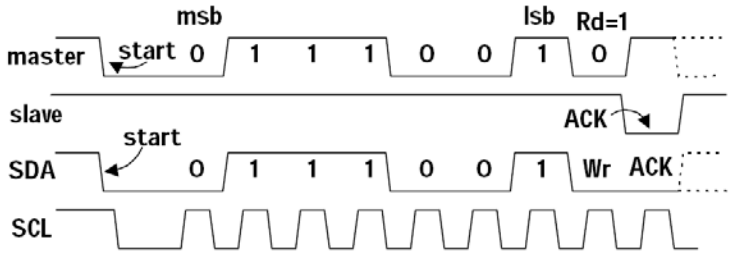
En el siguiente esquema vemos que la "condición de start" se produce cuando el "master" hace una transición de "1" a "0" de la señal SDA mientras SCL está a "1". La condición de stop es la opuesta, es decir, cuando el "master" hace una transición de "0" a "1" de la señal SDA mientras SCL está a "1". Esto es lo que se refleja en el siguiente gráfico:



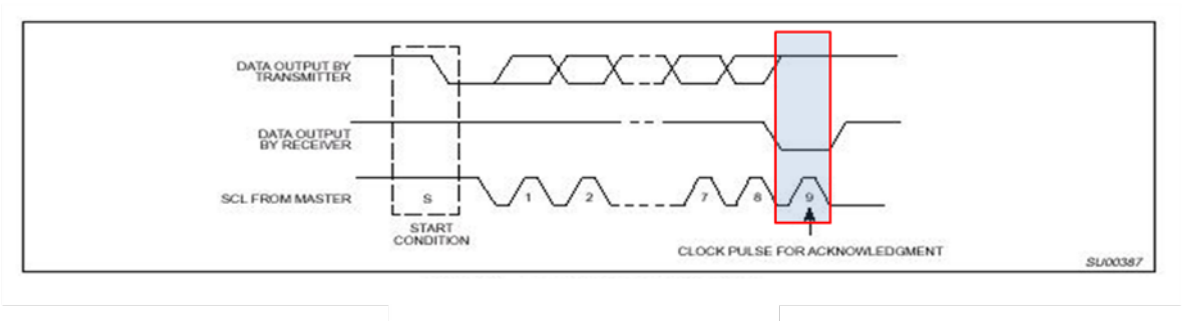
Téngase en cuenta que en circunstancias normales (distintas de "start", "stop" y otras condiciones atípicas) solo se producen cambios en la señal SDA mientras SCL está a "0" y solo se lee SDA cuando SCL está a "1", como se refleja en el siguiente gráfico:



Téngase muy presente que el master actúa como "transmitter" cuando escribe y como "Receiver" cuando lee. Además, recuérdese que **las líneas físicas SCL y SDA implementan un AND-implícito** entre todos los dispositivos conectados, de forma que si queremos leer un dato en SDA, debemos poner nuestra salida SDA (que es open-collector) a "1" durante todo el tiempo que estamos leyendo hasta que completamos la lectura del último bit, tras lo que ponemos un "0" para enviar el ACK. En la siguiente secuencia hay un ejemplo de inicio donde el master (transmitter) comienza enviando la dirección y todos los Slaves (receivers) están escuchando (SDA=1), hasta que quien ha sido direccionado contesta con el ACK (y en ese momento el master ha puesto el SDA=1 y el slave SDA=0, así, por el AND-implícito, el valor observable en la línea SDA será 0)



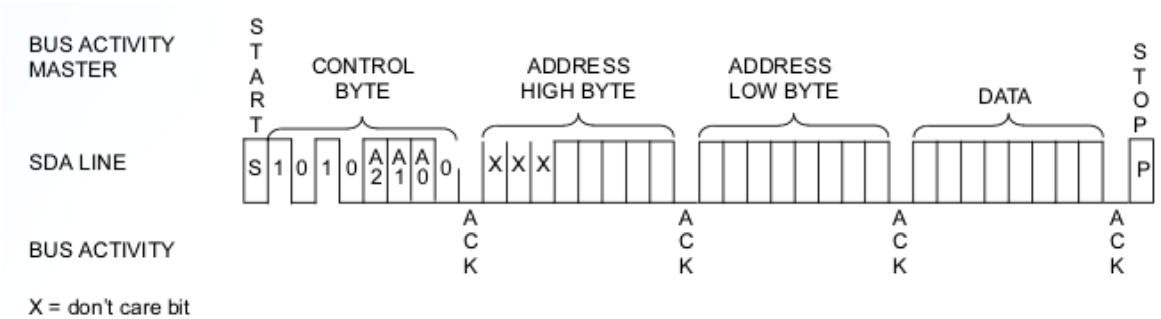
Y en general, en términos de transmisor - receptor, la gráfica sería la siguiente:

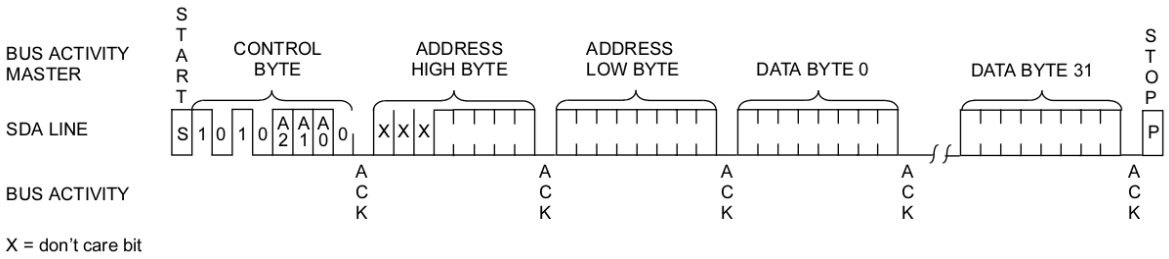


OPERACIÓN DE ESCRITURA

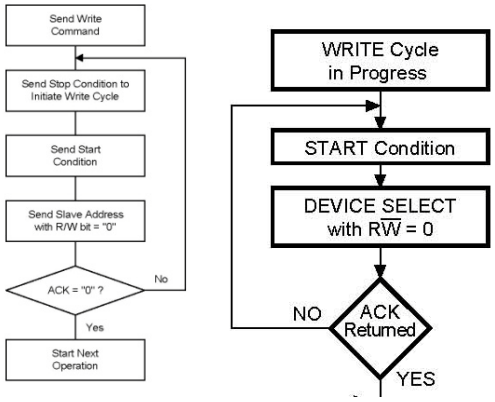
Vamos a ver un primer caso en que el maestro (transmitter todo el tiempo) escribe datos sobre un esclavo (receiver todo el tiempo). La secuencia de operaciones está esquematizada en la siguiente gráfica:

En el caso concreto de la memoria 24LC64, los diferentes modos de operación cuando es direccionada para escritura (con wc=0) están detallados en las siguientes gráficas:





Puede darse la circunstancia de que la memoria esté ocupada en un ciclo interno de escritura (que puede durar típicamente hasta 5 milisegundos, según los datos del fabricante, aunque en el Proteus este valor lo podemos cambiar según interese). En este caso, sucede que al enviar la dirección del dispositivo después de la condición de "start", la memoria no confirma la aceptación de la información enviada (no enviando "ack", o lo que es lo mismo enviando "NO-ACK") por lo que debemos establecer un bucle de espera para consultar (polling) tal condición de "ack", **repitiendo todo el "Control-Byte" desde el Start hasta el Ack**. Cuando se reciba el "ack" entonces podremos continuar con el resto del proceso para realizar la operación deseada. El siguiente esquema (optimizado y más detallado en la figura5.1 de de la hoja de datos del 24LS64 (y tmb otros chips, como *stmicroelectronics_24c01.pdf*) muestra el proceso descrito. Además, téngase en cuenta que el máximo número de bytes que podemos transferir en una sola operación de escritura (PAGE WRITE) es de **32** [1] debido que existe un contador interno de 5 bits para apuntar a la dirección baja dentro de una página del buffer. Por tanto, si el contador, en el modo PAGE, está a 11111 la siguiente escritura se realizará en la posición 31 de la página y el contador se autoincrementará pasando a la cuenta 00000. La escritura del siguiente byte en el buffer se realizaría sobreescribiendo ("machacando") en la posición cero del buffer y perdiendo lo que habíamos escrito en él.

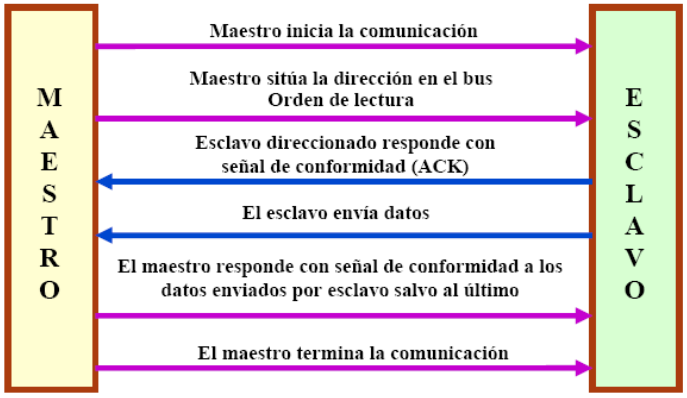


En esta figura una vez se envía "stop" y en el otro flujograma START??

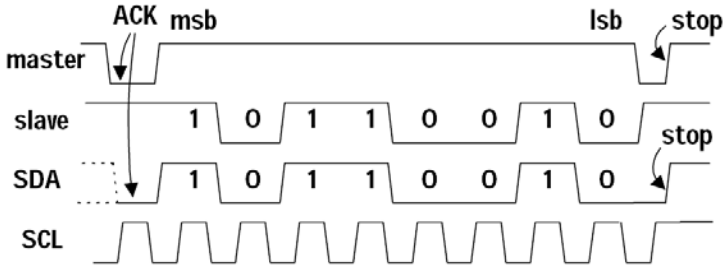
OPERACIÓN DE LECTURA

En la siguiente gráfica se recoge la secuencia general de pasos mediante la cual el Maestro lee información desde el Esclavo. Aquí, como se observa, tanto el Maestro como el Esclavo usan y se alternan los papeles de "transmitter" y "receiver".

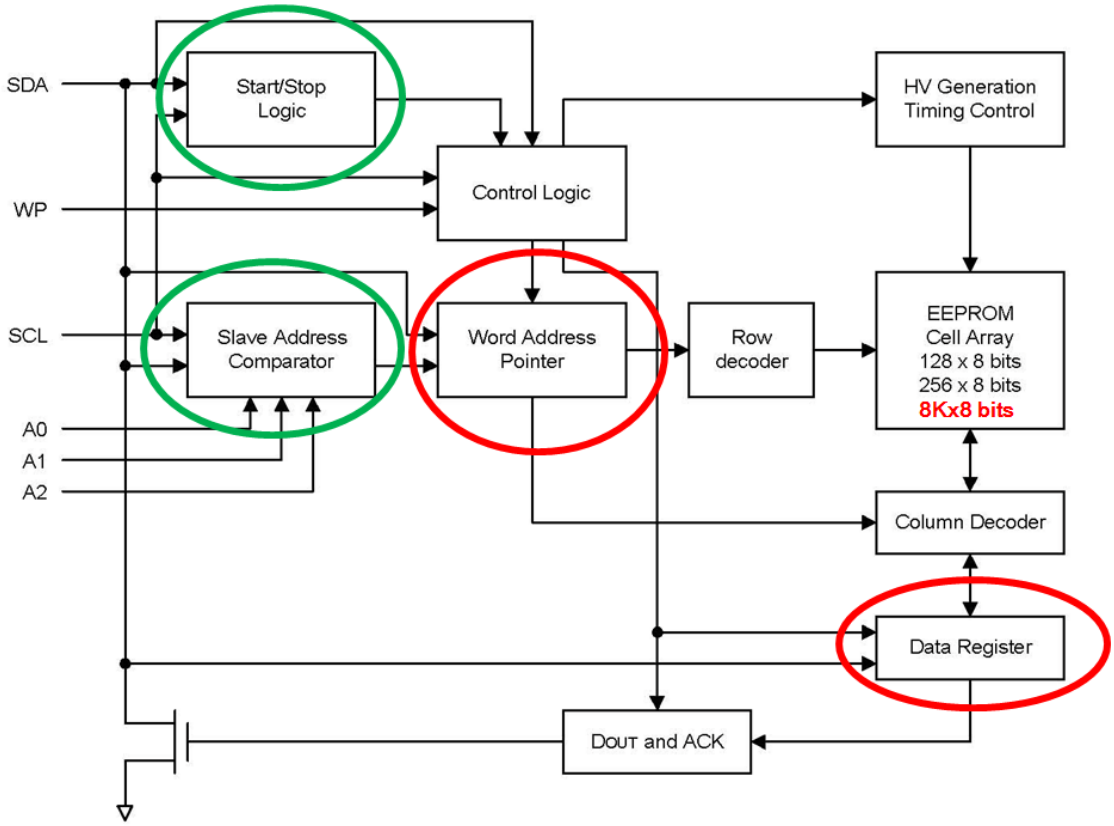
[1] Este dato 32 bytes está extraído de la documentación del fabricante. Sin embargo, en el pasado, en unas pocas de las memorias disponibles en el Laboratorio se han apreciado, empíricamente, discrepancias en el tamaño respecto a las especificaciones dadas por los fabricantes. Esto es posible detectarlo y solucionarlo por software, pero no será necesario dado que en el simulador no va a ocurrir.



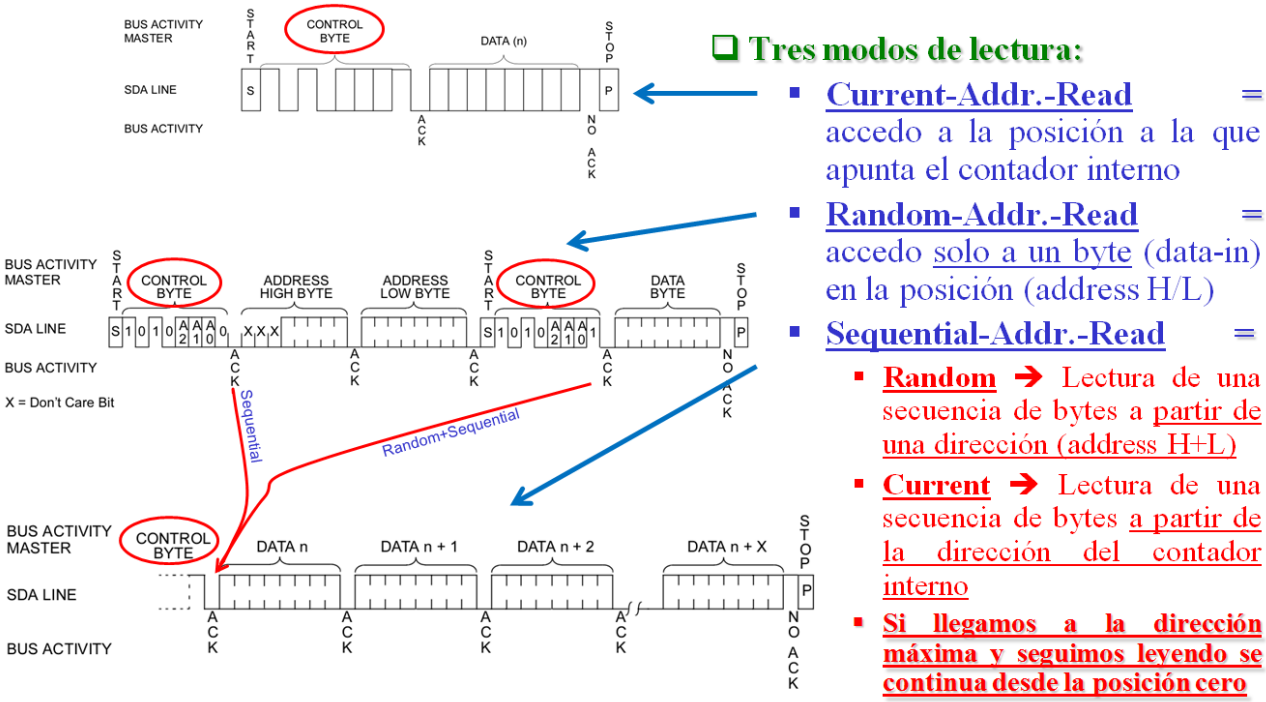
En la siguiente gráfica vemos un detalle de la señal SDA puesta por el Maestro (cuando actúa como receiver), la SDA puesta por el Esclavo (transmitter) y el SDA observable en la línea física y que es el resultante del AND-implícito.



En la siguiente figura se detalla los distintos bloques funcionales de la memoria 24LC64 (y otras similares) donde se aprecia el "Word Address Pointer" que señala la posición que va a ser accedida.



En el caso concreto de la memoria 24LC64, los diferentes modos de operación cuando es direccionada para lectura están detallados en la siguiente gráfica:



❑ **Tres modos de lectura:**

- **Current-Addr.-Read** = accedo a la posición a la que apunta el contador interno
- **Random-Addr.-Read** = accedo solo a un byte (data-in) en la posición (address H/L)
- **Sequential-Addr.-Read** =
  - **Random** ➔ Lectura de una secuencia de bytes a partir de una dirección (address H+L)
  - **Current** ➔ Lectura de una secuencia de bytes a partir de la dirección del contador interno
  - **Si llegamos a la dirección máxima y seguimos leyendo se continua desde la posición cero**

Como podemos apreciar existen cuatro (o tres, según se mire) modos de lectura. La memoria tiene un puntero o contador interno (ya descrito en el esquema general y que es común para la escritura) que apunta a la posición actual a ser leída. Cada vez que se realiza la lectura (o escritura) el contador se auto-incrementa.

- En el primer modo de lectura, el "**current address read**" (lectura de la posición actual) enviamos la condición de START, seleccionamos el dispositivo con R/W=1(Read) y, tras responder con Ack (si está disponible) este nos envía el dato almacenado en la posición de memoria actual (la señalada por el puntero) y además incrementa el puntero que quedará apuntando a la siguiente posición de memoria (salvo que fuese la última posición, en cuyo caso quedará apuntado a la posición cero). Para terminar, **el Maestro no envía el ACK (envía No-Ack=1,** es decir, **leemos un 1)** y seguidamente envía la condición de STOP.

- En el segundo modo de lectura, el "**random address read**" hacemos inicialmente un "dummy write" o intento de escritura para cargar la dirección que queremos leer en el contador interno. Para esto enviamos la condición de START, seleccionamos el dispositivo con R/W=0 (Write), enviamos la dirección y tras recibir el ACK enviamos la condición de START. Ahora volvemos a seleccionar el dispositivo para operación de lectura y éste nos envía el dato almacenado en la posición de memoria actual (la señalada por el puntero) y además incrementa el puntero para quedar apuntando a la siguiente posición de memoria (salvo que fuese la última posición, en cuyo caso quedará apuntado a la posición cero). Para terminar **el Maestro no envía el ACK (envía No-Ack=1)** y seguidamente envía la condición de STOP.

- El tercer y cuarto modo son los mismos que el primero y segundo, pero en modo "**Sequential**". Esto es debido a que el dispositivo seguirá enviándonos los datos o contenidos de las sucesivas posiciones de memoria mientras el Maestro responda enviando el ACK y siga enviando ciclos de SCL. Para terminar **el Maestro no envía el ACK (envía No-Ack=1)** tras el último dato y

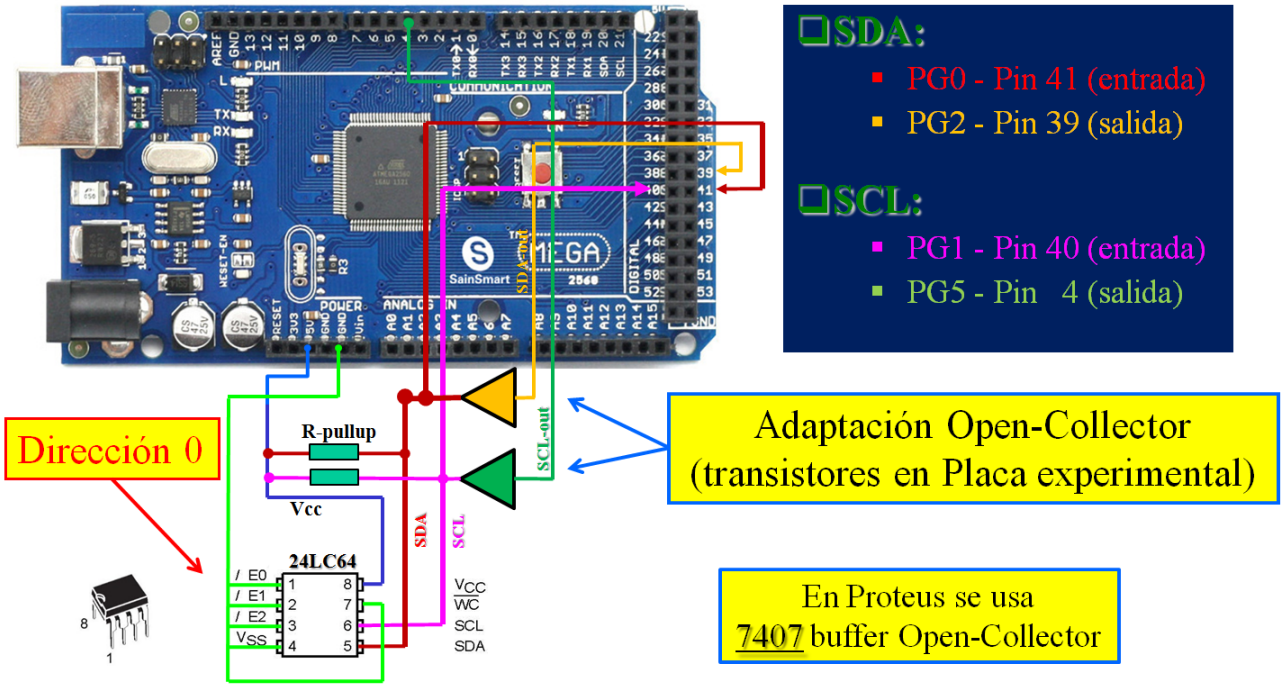
seguidamente envía la condición de STOP. Cuando el puntero interno alcanza el valor máximo y lo incrementamos, continúa desde cero.

Téngase presente que **en la lectura no se está usando un buffer intermedio**, por lo tanto **no es de aplicación la limitación de 32 bytes** en la lectura secuencial como existe en la escritura en modo página. También es importante señalar que tras leer los 8 bits del dato, **si se trata del último byte que se va a leer hay que enviar No-Ack, es decir un "1" en lugar de un "0", Ack,** porque si enviamos un "0" o Ack el esclavo continuaría enviando y en el siguiente pulso de reloj pondría en el SDA el bit 7 (MSB) del siguiente byte, cosa que podría crear un conflicto, ya que nosotros en ese mismo ciclo vamos a poner un STOP, sobre todo si ese bit7 fuese un cero.

4. Esquema de conexiones

Como explicamos en clase, el microcontrolador ATmega2560 posee internamente la capacidad de gestionar "por hardware" el protocolo del I2C mediante los terminales 20 y 21. Sin embargo el objetivo principal de esta práctica es saber realizar el control de un bus I2C utilizando líneas de entrada salida de propósito general y gestionar los protocolos de la comunicación en el bus I2C mediante software escrito totalmente por el estudiante.

Para llevarlo a cabo, utilizaremos en esquema de conexiones recogido en la figura siguiente:



En este caso tenemos el bus de reloj SCL (dibujado en color violeta) y la señal de datos SDA (en color rojo) accesibles en los terminales 40 (PG1) y 41 (PG0), respectivamente. **Es críticamente importante que estos terminales los programemos como entrada en la inicialización (setup)**; a través de ellos podremos leer (monitorizar) en todo momento el estado de las señales en el bus. Para poder escribir en el bus utilizaremos los terminales 4 (PG5) para actuar sobre el SCL y el terminal 39 (PG2) para actuar sobre el SDA (programándolos como salidas en el setup).

Seguidamente se muestra **un ejemplo** de la inicialización que se debe realizar en el "setup" en relación con las líneas de E/S para el SDA y el SCL, para llevar a cabo la práctica:

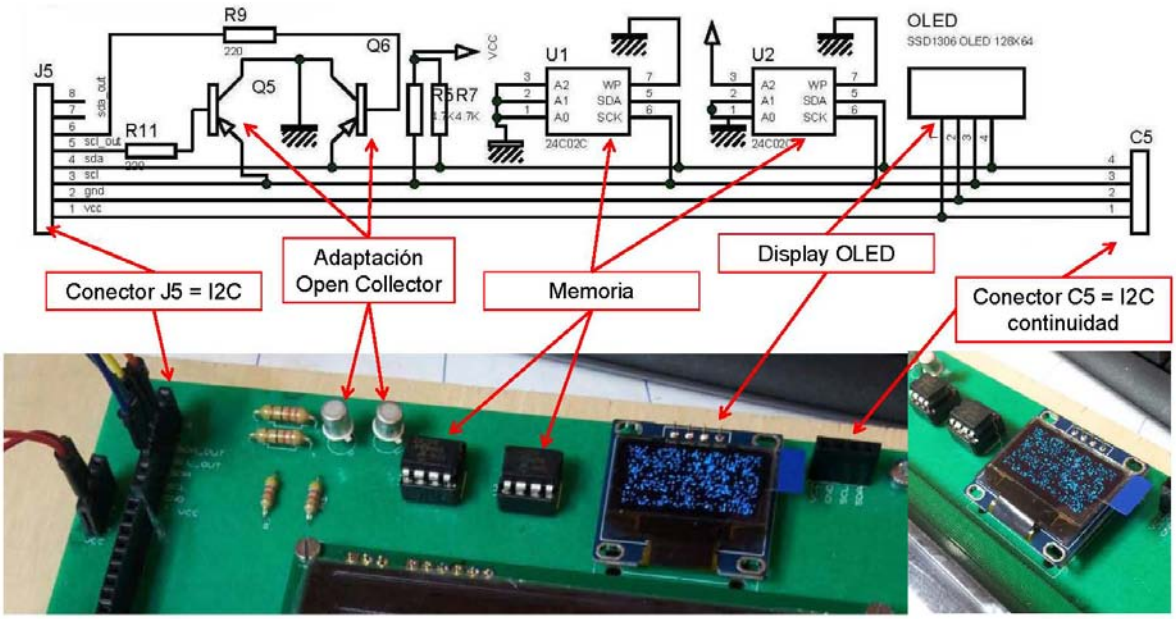
```
#define LEE_SCL   40 // puerto de entrada para leer el estado de la línea SCL
#define LEE_SDA   41 // puerto de entrada para leer el estado de la línea SDA
#define ESC_SCL    4 // puerto de salida para escribir el valor de la línea SCL-out
#define ESC_SDA   39 // puerto de salida para escribir el valor de la línea SDA-out

void setup(){

  // Inicialización del canal serie para comunicarse con el usuario
  Serial.begin(9600);

  // Inicialización de los terminales de entrada
  pinMode(LEE_SDA, INPUT);
  pinMode(LEE_SCL, INPUT);
 // Inicialización de los terminales de salida
  pinMode(ESC_SDA, OUTPUT);
  pinMode(ESC_SCL, OUTPUT);
 // Asegurarse de no intervenir el bus poniendo SDA y SCL a "1"....
  digitalWrite(ESC_SDA, HIGH);
  digitalWrite(ESC_SCL, HIGH);
}
```

También se puede apreciar que se ha conectado la señal **WC a "0"**, con lo que se **habilita permanentemente la escritura** en el dispositivo. Los terminales A0, A1 y A2 (E0, E1 y E2) han sido también conectados de forma que **la dirección del dispositivo dentro del bus I2C será "1010000" para acceder al chip U3 (U1 en la placa) y será "1010100" para acceder al chip U4 (U2 en la placas)**; ver el esquema en el proyecto del Proteus o de la placa experimental.



Como se ha explicado anteriormente, estos terminales, cuando los programamos como salida, no podemos garantizar que sean de tipo "open-collector" u "open-drain" como necesitamos para cumplir con las especificaciones del I2C. Por este motivo utilizamos un acoplamiento mediante transistores (PNP en nuestro diseño) para poder conectar adecuadamente cada una de las salidas del Arduino al bus I2C que

hemos implementado. También podría realizarse con un buffer estándar tipo open-collector (por ejemplo, el TTL 7407, usado en versiones anteriores de la práctica y **en el Proyecto Proteus**).

5. Realización práctica

Como ya se ha comentado, la realización de la presente práctica se pretende alcanzar el objetivo de ser capaz de realizar las operaciones de control básicas sobre el bus I2C, comprendiendo su funcionamiento y ganar experiencia en el manejo detallado del mismo. Asimismo, se ha de garantizar el correcto funcionamiento tanto del hardware como del software empleando los medios y procesos de depuración que procedan o estime oportunos. Para alcanzar estos objetivos se plantea la realización de un conjunto de tareas, de menor a mayor dificultad, empezando por dominar el acceso a la memoria 24LC64, es decir, probar sus distintos modos de acceso y operaciones elementales hasta implementar, con la experiencia ganada, una aplicación completa que facilite el uso de la memoria (y de algún otro dispositivo como mejora) a posibles usuarios.

5.1. Tarea 1. Implementación del protocolo I2C: funciones básicas

En primer lugar, tenemos que aprender a utilizar el bus I2C. Para facilitar la realización de la práctica, seguidamente se van a proporcionar una **serie de sugerencias** desde el punto de vista del implementador. Tienen como finalidad darles un punto de partida y unas indicaciones que pretenden facilitar la consecución de los objetivos. Evidentemente otros enfoques de resolución son posibles ya que lo que se presenta aquí es demasiado simplista (por un enfoque inicial docente, antes que de eficacia), y que para algunos incluso podrían ser más directos o sencillos. Siéntanse libres de elegir el procedimiento de implementación que le resulte de mayor comodidad o efectividad (siempre que cumplan con los objetivos de la práctica y que funcionen correctamente).

Operaciones elementales del Maestro:

Seguidamente establecemos una lista de las operaciones más elementales que tiene que realizar el Maestro, y por lo tanto que tenemos que implementar en nuestro software de control.

Tipos de acciones Elementales		
Actividad del Maestro	Acrónimo	Cometario
Condición de START	**START**	
Condición de STOP	**STOP**	
Envío un bit a "1"	**E-BIT-1**	
Envío un bit a "0"	**E-BIT-0**	
Envío el "ACK"	**E-ACK** (= E-BIT-0)	Es igual a Envío bit a "0"
Envío el "NO-ACK"	**E-NO-ACK** (= E-BIT-1)	Es igual a Envío bit a "1"
Recibo el "ACK"	**R-ACK** (= R-BIT)	Es igual a Recibo un bit
Recibo un bit	**R-BIT**	

En realidad vemos que en este modo simplificado, solo necesitamos cinco de estas operaciones/funciones elementales, con las que podremos implementar todas las demás (podrían ser cuatro funciones si E-BIT-0 y E-BIT-1 se funden en 1 a la que se le pasa el valor 0/1, depende del estilo que se prefiera) .

Un ejemplo muy básico (docente, i.e. fácilmente optimizables) de secuencia de señales y acciones que se pueden usar (sabiendo que son lentas) para generar se han resumido en las siguientes cinco tablas y se explican detalladamente en la sesión de presentación inicial de la práctica:

START				
	Sec. temporal/tipo			
	1	2	3	4
Señal	Esc	Lec	Esc	Esc
SDA	1	¿1?	0	0
SCL	1	¿1?	1	0

STOP				
	Sec. temporal/tipo			
	1	2	3	4
Señal	Esc	Esc	Esc	Esc
SDA	0	0	1	1
SCL	0	1	1	1

E-BIT-1				
	Sec. temporal/tipo			
	1	2	3	4
Señal	Esc	Esc	Esc	Esc
SDA	1	1	1	1
SCL	0	1	1	0

E-BIT-0 = E-ACK				
	Sec. temporal/tipo			
	1	2	3	4
Señal	Esc	Esc	Esc	Esc
SDA	0	0	0	0
SCL	0	1	1	0

R-BIT = R-ACK				
	Sec. temporal/tipo			
	1	2	3	4
Señal	Esc	Esc	Lec	Esc
SDA	1	1	¿*?	1
SCL	0	1	¿1?	0
	* = Valor leído 0,1			

Los valores leídos, que han sido resaltados en amarillo, (en START, R-BIT y R-ACK) serán gestionados adecuadamente por el software del usuario; por ejemplo, cuando se va a acceder al bus, en el caso de

START, si no se lee SDA=1 y SCL=1 hay que permanecer esperando a que el bus se libere (aunque en esta práctica en el simulador no debe suceder, salvo error en las conexiones o que se añada otro maestro).

Utilizando lo anterior como guía o ejemplo de test, podremos hacer una secuencia que implemente las distintas operaciones necesarias, que serán deducidas de la información proporcionada por el fabricante. Por ejemplo, la secuencia para realizar un "BYTE WRITE" sería como se especifica en el siguiente cuadro:

"BYTE WRITE"	
Secuencia	Acción
START	
E-BIT-1	Los 7 bits de dirección del dispositivo 1010 000
E-BIT-0	
E-BIT-1	
E-BIT-0	
E-BIT-0	
E-BIT-0	
E-BIT-0	
E-BIT-0	R/W = 0 (W)
R-ACK	Leer ack del dispositivo; si ok, sigo
E-BIT-0	Pongo la dirección del byte Hight = XXX0 0000 ==> 0000 0000 Envío 8 ceros (esto pondrá la parte Hight del contador interno a 0000 0000)
E-BIT-0	
E-BIT-0	
E-BIT-0	
E-BIT-0	
E-BIT-0	
E-BIT-0	
E-BIT-0	
R-ACK	Leer ack del dispositivo; si ok, sigo
E-BIT-0	Pongo la dirección del byte Low= 0000 0000 Envío 8 ceros (esto pondrá la parte Low del contador interno a 0000 0000)
E-BIT-0	
E-BIT-0	
E-BIT-0	
E-BIT-0	
E-BIT-0	
E-BIT-0	
E-BIT-0	
R-ACK	Leer ack del dispositivo; si ok, sigo
E-BIT-0	Envío el dato, por ej.: 0101 0101 = 0x55
E-BIT-1	
E-BIT-0	
E-BIT-1	
E-BIT-0	
E-BIT-1	
E-BIT-0	
E-BIT-1	
R-ACK	Leer ack del dispositivo; si ok, sigo
STOP	==> Se produce el almacenamiento (5 msg)

Consultar las correspondientes gráficas presentes en la documentación para implementar la secuencia similar a la anterior para las otras operaciones que se necesita utilizar. Cuando logramos que esto funcione,

ya podemos desarrollar las capas de más alto nivel que nos proporcione un manejo sencillo y eficiente de acceso a los dispositivos I2C

5.2. Tarea 2. Menú básico de usuario

Una vez implementadas las funciones básicas de bus I2C, procederemos en esta tarea a la implementación de un menú, con 6 opciones básicas, que permita al usuario la verificación y el uso de la memoria.

Se puede seguir un procedimiento de menos a más como, por ejemplo:

1. Comprobar el funcionamiento escribiendo un valor a una posición fija y leyéndolo después.
    a. En el Proteus podemos ver lo que contiene la memoria 24LC64, pulsando "Pause" y activándolo en el menú "Debug" si es necesario (primera vez o fase inical).
    b. Además, en Proteus se puede usar el Osciloscopio o el "I2C Debugger", muy útiles al principio o cuando surge algún problema. Se explicará su uso aparte, a interesados.
2. Comprobar la memoria completa escribiendo una secuencia de bytes y luego leer los bytes escritos (o también verlos en el Debbuger).

Utilizando la experiencia anterior, procederemos a la implementación de la aplicación de usuario consistente en un menú, a mostrar en el "Terminal Virtual" de Proteus, y que permitirá el uso de la memoria a través de sus diferentes opciones.

1. Guardar un dato (de 0 a 255) en cualquier dirección de memoria del dispositivo 24LC64. Tanto el dato como la dirección se han de solicitar al usuario. *(pidiéndolos por teclado/pantalla)*
2. Leer una posición (de 0 a 8191) del 24LC64 *(pidiendo la posición por pantalla)*
3. Inicializar un bloque de 256 bytes contiguos de la memoria 24LC64 a un valor *(la dirección del primer elemento y el valor a escribir se solicitan por pantalla; medir el tiempo el tiempo empleado en acceso a la memoria)*
4. Mostrar el contenido de un bloque de 256 bytes contiguos del 24LC64, comenzando en una dirección especificada *(en una matriz de 8 0 16 columnas, en hexadecimal; medir el tiempo que consume el acceso a la memoria)*
5. Inicializar usando "**Page Write**" un bloque de 256 bytes contiguos del 24LC64 a un valor *(la dirección del primer elemento y el valor a escribir se solicitan por pantalla; medir el tiempo que consume el acceso a la memoria)*
6. Mostrar el contenido de un bloque de 256 bytes del 24LC64 (usando **Sequential Read**), comenzando en una dirección especificada *(en una matriz de 8 o 16 columnas, en hexadecimal; medir el tiempo que consume el acceso a la memoria)*

Cuando se ejecute el programa, se mostrará el menú anterior y permanecerá a la espera de que se le solicite una de las opciones. Cuando se haya completado la toma de datos y/o visualización de resultados, se volverá al principio, es decir, mostrar el menú y permanecer a la espera. En la lectura de valores, se comprobará siempre la validez de los datos proporcionados por el usuario, y se implementará el tratamiento de errores necesario.

Para lo anterior resultará útil crear procedimientos que sirvan de vínculo para programar más cómodamente. Por ejemplo, crear, al menos, un procedimiento que podríamos denominar "Escribe-en-

Mem-I2C(*dir*,*valor*)" para escribir un "valor" en una determinada dirección "dir"; y otro procedimiento para "Lee-de-Mem-I2C(*dir*)", para leer una posición "dir" de la memoria. Para implementar procedimientos que transfieran bloques mayores, pero recuérdese que el máximo número de bytes que podemos transferir en una sola operación de escritura (PAGE WRITE) es de 32 bytes (**en otros chips puede ser diferente**).

Ejemplo de listado del contenido de 128 bytes de memoria en 8 columnas (Se pide mostrar 256 bytes, duplicando el número de filas o el de columnas):

	0	1	2	3	4	5	6	7
0	33	33	33	33	33	33	33	33
8	33	33	33	33	33	33	33	33
16	33	33	33	33	33	33	33	33
24	33	33	33	33	33	33	33	33
32	33	33	33	33	33	33	33	33
40	33	33	33	33	33	33	33	33
48	33	33	33	33	33	33	33	33
56	33	33	33	33	33	33	33	33
64	33	33	33	33	33	33	33	33
72	33	33	33	33	33	33	33	33
80	33	33	33	33	33	33	33	33
88	33	33	33	33	33	33	33	33
96	33	33	33	33	33	33	33	33
104	33	33	33	33	33	33	33	33
112	33	33	33	33	33	33	33	33
120	33	33	33	33	33	33	33	33

5.3. Tarea 3. Dispositivo de Reloj de Tiempo Real (RTC) (opcional, mejora)

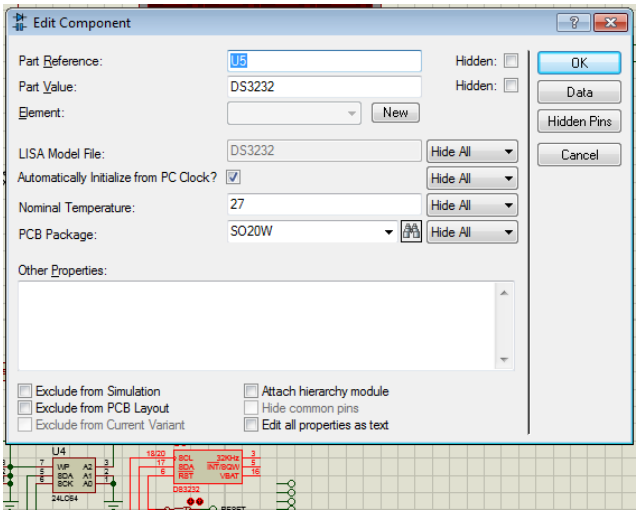
Como mejoras a la práctica básica se propone el uso de otro dispositivo I2C, como puede ser un Reloj de Tiempo Real (RTC) que nos permite disponer de la fecha y hora en un sistema microcomputador como el nuestro, entre otras funciones.

Para la implementación de esta tarea, se añadirán dos nuevas opciones al menú básico: opción 7 y opción 8.

Opción 7 (0.5 puntos): Mostrar en pantalla la fecha y hora, tomándola del RTC; en el Anexo se muestra resumen de los registros del RTC, pero lo mejor es consultar la documentación del fabricante del DS3232 en el Moodle (Hay dos versiones).

Opción 8 (0.5 puntos): Mostrar en Pantalla la Temperatura (la pueden cambiar en propiedades) leyendo del termómetro interno del RTC.

La información del RTC DS3232 (su **"Slave Address"** es **"1101 000")**, también la tienen accesible a través de Proteus, abriendo propiedades y pulsando el botón "Data":



6. Entrega de la práctica

Una vez desarrollada la aplicación y comprobado su correcto funcionamiento es, absolutamente necesario, subir el informe de la práctica al Campus Virtual de la asignatura, en tiempo y forma, a través del enlace disponible en la sección de prácticas. El informe consistirá en el proyeto Proteus (*.pdsprj) de la aplicación con los programas adecuadamente comentados. En principio, solo será necesario subir un fichero cuyo nombre seguirá la siguiente nomenclatura:

23-24_plab3_inciales nombre y apellidos.pdsprj

Ejemplo:

23-24_plab3_amf.pdsprj

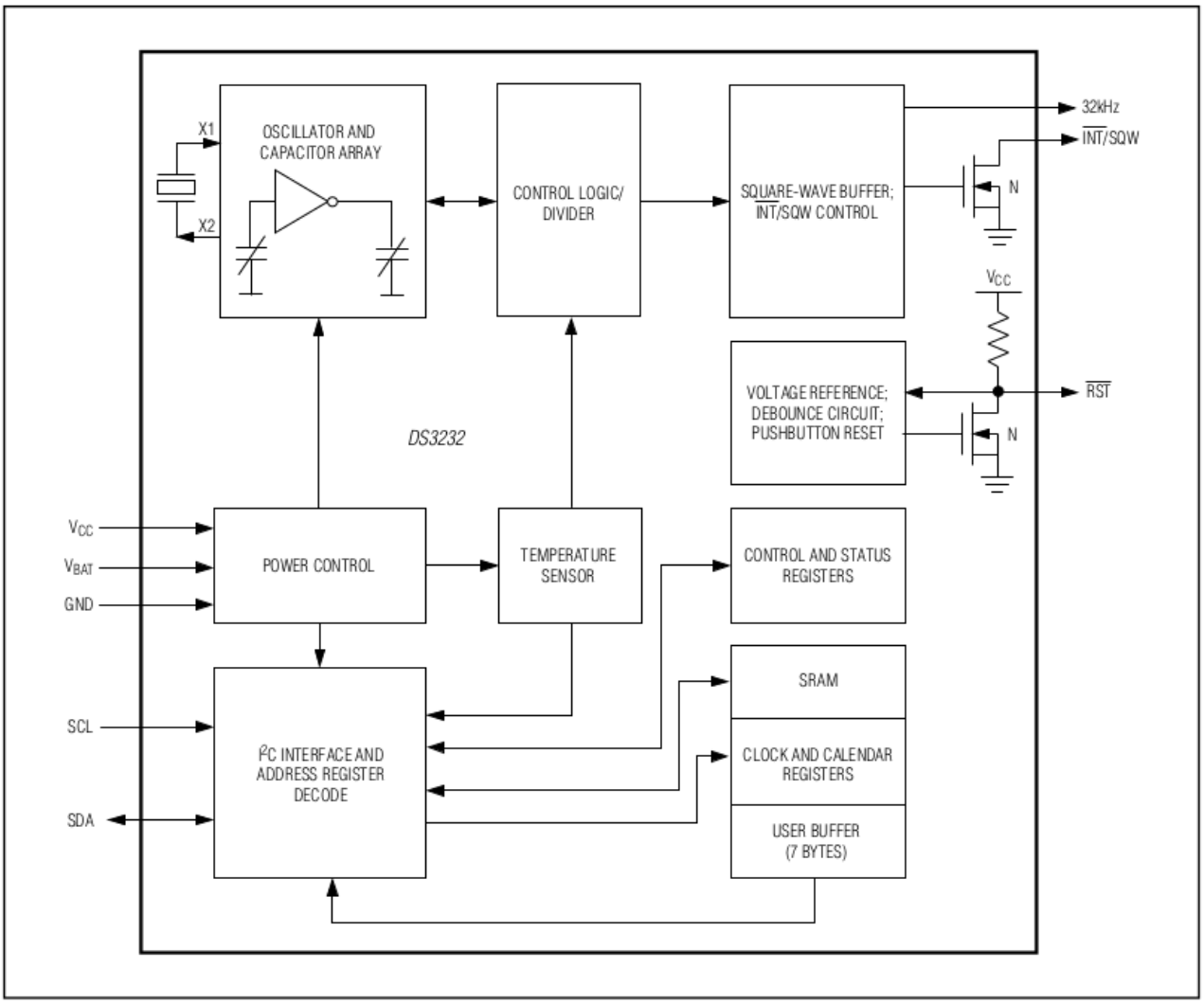
Fichero correspondiente al proyecto Proteus de la práctica 3 en laboratorio entregado por Anabel Medina Falcón.

7. Anexo: Resumen del Reloj de Tiempo Real (RTC) DS3232

DS3232

Extremely Accurate I²C RTC with Integrated Crystal and SRAM

Block Diagram



Seguidamente se detallan los 256 registros/SRAM de 8 bits internos. La mecánica de funcionamiento es muy similar a la vista anteriormente. Tiene un puntero que hay que inicializar y que luego se autoincrementa tras la operación. Tiene un bloque relacionado con la hora y fecha, seguido de dos alarmas, dos registros de control/estado, dos registros para leer la temperatura interna y el resto de direcciones de SRAM. En la siguiente tabla se detallan los campos concretos:

Figure 1. Address Map for DS3232 Timekeeping Registers and SRAM

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	1–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{AM}$/PM 00–23
0Ah	A1M4	DY/$\overline{DT}$	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{AM}$/PM 00–23
0Dh	A2M4	DY/$\overline{DT}$	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	BB32kHz	CRATE1	CRATE0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—
13h	0	0	0	0	0	0	0	0	Not used	Reserved for test
14h–0FFh	x	x	x	x	x	x	x	x	SRAM	00h–0FFh

Note: *Unless otherwise specified, the registers' state is not defined when power is first applied.*

Table 2. Alarm Mask Bits

DY/$\overline{DT}$	ALARM 1 REGISTER MASK BITS (BIT 7)				ALARM RATE
	A1M4	A1M3	A1M2	A1M1	
X	1	1	1	1	Alarm once per second
X	1	1	1	0	Alarm when seconds match
X	1	1	0	0	Alarm when minutes and seconds match
X	1	0	0	0	Alarm when hours, minutes, and seconds match
0	0	0	0	0	Alarm when date, hours, minutes, and seconds match
1	0	0	0	0	Alarm when day, hours, minutes, and seconds match

DY/$\overline{DT}$	ALARM 2 REGISTER MASK BITS (BIT 7)			ALARM RATE
	A2M4	A2M3	A2M2	
X	1	1	1	Alarm once per minute (00 seconds of every minute)
X	1	1	0	Alarm when minutes match
X	1	0	0	Alarm when hours and minutes match
0	0	0	0	Alarm when date, hours, and minutes match
1	0	0	0	Alarm when day, hours, and minutes match

Control Register (0Eh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE
POR*:	0	0	0	1	1	1	0	0

**POR is defined as the first application of power to the device, either V_{BAT} or V_{CC}.*

Special-Purpose Registers

The DS3232 has two additional registers (control and control/status) that control the real-time clock, alarms, and square-wave output.

Control Register (0Eh)

Bit 7: Enable Oscillator ($\overline{EOSC}$). When set to logic 0, the oscillator is started. When set to logic 1, the oscillator is stopped when the DS3232 switches to battery power. This bit is clear (logic 0) when power is first applied. When the DS3232 is powered by V_{CC}, the oscillator is always on regardless of the status of the $\overline{EOSC}$ bit. When $\overline{EOSC}$ is disabled, all register data is static.

Bit 6: Battery-Backed Square-Wave Enable (BBSQW). When set to logic 1 with INTCN = 0 and V_{CC} < V_{PF}, this bit enables the square wave. When BBSQW is logic 0, the $\overline{INT}$/SQW pin goes high impedance when V_{CC} < V_{PF}. This bit is disabled (logic 0) when power is first applied.

Bit 5: Convert Temperature (CONV). Setting this bit to 1 forces the temperature sensor to convert the temperature into digital code and execute the TCXO algorithm to update the capacitance array to the oscillator. This can only happen when a conversion is not already in progress. The user should check the status bit BSY before forcing the controller to start a new TCXO execution. A user-initiated temperature conversion does not affect the internal 64-second (default interval) update cycle.

A user-initiated temperature conversion does not affect the BSY bit for approximately 2ms. The CONV bit remains at a 1 from the time it is written until the conversion is finished, at which time both CONV and BSY go to 0. The CONV bit should be used when monitoring the status of a user-initiated conversion.

Bits 4 and 3: Rate Select (RS2 and RS1). These bits control the frequency of the square-wave output when the square wave has been enabled. The following table shows the square-wave frequencies that can be selected with the RS bits. These bits are both set to logic 1 (8.192kHz) when power is first applied.

SQUARE-WAVE OUTPUT FREQUENCY

RS2	RS1	SQUARE-WAVE OUTPUT FREQUENCY
0	0	1Hz
0	1	1.024kHz
1	0	4.096kHz
1	1	8.192kHz

Bit 2: Interrupt Control (INTCN). This bit controls the $\overline{INT}$/SQW signal. When the INTCN bit is set to logic 0, a square wave is output on the $\overline{INT}$/SQW pin. When the INTCN bit is set to logic 1, a match between the timekeeping registers and either of the alarm registers activates the $\overline{INT}$/SQW (if the alarm is also enabled). The corresponding alarm flag is always set regardless of the state of the INTCN bit. The INTCN bit is set to logic 1 when power is first applied.

Bit 1: Alarm 2 Interrupt Enable (A2IE). When set to logic 1, this bit permits the alarm 2 flag (A2F) bit in the status register to assert $\overline{INT}$/SQW (when INTCN = 1). When the A2IE bit is set to logic 0 or INTCN is set to logic 0, the A2F bit does not initiate an interrupt signal. The A2IE bit is disabled (logic 0) when power is first applied.

Bit 0: Alarm 1 Interrupt Enable (A1IE). When set to logic 1, this bit permits the alarm 1 flag (A1F) bit in the status register to assert $\overline{INT}$/SQW (when INTCN = 1). When the A1IE bit is set to logic 0 or INTCN is set to logic 0, the A1F bit does not initiate the $\overline{INT}$/SQW signal. The A1IE bit is disabled (logic 0) when power is first applied.

Control/Status Register (0Fh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	OSF	BB32kHz	CRATE1	CRATE0	EN32kHz	BSY	A2F	A1F
POR*:	1	1	0	0	1	0	0	0

**POR is defined as the first application of power to the device, either V_{BAT} or V_{CC}.*

Control/Status Register (0Fh)

Bit 7: Oscillator Stop Flag (OSF). A logic 1 in this bit indicates that the oscillator either is stopped or was stopped for some period and may be used to judge the validity of the timekeeping data. This bit is set to logic 1 any time that the oscillator stops. The following are examples of conditions that can cause the OSF bit to be set:

1) The first time power is applied.
2) The voltages present on both V_{CC} and V_{BAT} are insufficient to support oscillation.
3) The $\overline{EOSC}$ bit is turned off in battery-backed mode.
4) External influences on the crystal (i.e., noise, leakage, etc.).

This bit remains at logic 1 until written to logic 0.

Bit 6: Battery-Backed 32kHz Output (BB32kHz). This bit enables the 32kHz output when powered from V_{BAT} (provided EN32kHz is enabled). If BB32kHz = 0, the 32kHz output is low when the part is powered by V_{BAT}.

Bits 5 and 4: Conversion Rate (CRATE1 and CRATE0). These two bits control the sample rate of the TCXO. The sample rate determines how often the temperature sensor makes a conversion and applies compensation to the oscillator. Decreasing the sample rate decreases the overall power consumption by decreasing the frequency at which the temperature sensor operates. However, significant temperature changes that occur between samples may not be completely compensated for, which reduce overall accuracy. When a new conversion rate is written to the register, it may take up to the new conversion rate time before the conversions occur at the new rate.

CRATE1	CRATE0	SAMPLE RATE (seconds)
0	0	64
0	1	128
1	0	256
1	1	512

Bit 3: Enable 32kHz Output (EN32kHz). This bit indicates the status of the 32kHz pin. When set to logic 1, the 32kHz pin is enabled and outputs a 32.768kHz square-wave signal. When set to logic 0, the 32kHz pin goes low. The initial power-up state of this bit is logic 1, and a 32.768kHz square-wave signal appears at the 32kHz pin after a power source is applied to the DS3232 (if the oscillator is running).

Bit 2: Busy (BSY). This bit indicates the device is busy executing TCXO functions. It goes to logic 1 when the conversion signal to the temperature sensor is asserted and then is cleared when the conversion is complete.

Bit 1: Alarm 2 Flag (A2F). A logic 1 in the alarm 2 flag bit indicates that the time matched the alarm 2 registers. If the A2IE bit is logic 1 and the INTCN bit is set to logic 1, the $\overline{INT}$/SQW pin is also asserted. A2F is cleared when written to logic 0. This bit can only be written to logic 0. Attempting to write to logic 1 leaves the value unchanged.

Bit 0: Alarm 1 Flag (A1F). A logic 1 in the alarm 1 flag bit indicates that the time matched the alarm 1 registers. If the A1IE bit is logic 1 and the INTCN bit is set to logic 1, the $\overline{INT}$/SQW pin is also asserted. A1F is cleared when written to logic 0. This bit can only be written to logic 0. Attempting to write to logic 1 leaves the value unchanged.

Aging Offset (10h)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA
POR*:	0	0	0	0	0	0	0	0

No es necesario modificar nada para conseguir la exactitud normal.

Temperature Register (Upper Byte) (11h)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA
POR*:	0	0	0	0	0	0	0	0

Parte entera de la Temperatura (8bits con signo).

Temperature Register (Lower Byte) (12h)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	DATA	DATA	0	0	0	0	0	0
POR*:	0	0	0	0	0	0	0	0

Parte fraccionaria de la Temperatura. La precisión máxima es de 0,25º Centígrados, por lo tanto los bits 6 y 7 codifican las 4 posibilidades de fracción:

Bit7	Bit6	Temperatura
Dato	Dato	Fracción ºC
0	0	0,00
0	1	0,25
1	0	0,50
1	1	0,75

SRAM (14h–FFh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	D7	D6	D5	D4	D3	D2	D1	D0
POR*:	X	X	X	X	X	X	X	X

**POR is defined as the first application of power to the device, either V_{BAT} or V_{CC}.*

SRAM

The DS3232 provides 236 bytes of general-purpose battery-backed read/write memory. The I^2C address ranges from 14h to 0FFh. The SRAM can be written or read whenever V_{CC} or V_{BAT} is greater than the minimum operating voltage.

La **"Slave Address" es "1101 000"**. La escritura y lectura es similar a las ya explicadas; la única diferencia destacable es , que al ser solo 256 registros, la dirección del puntero es de 8 bits (word address) y por lo tanto se tiene que enviar un solo byte en lugar de dos bytes que es necesario enviar el caso de la memoria 24LC64, ya que esta es de 8Kbytes de tamaño. También, en la escritura no hay buffer intermedio, y por lo tanto es mucho más simple ya que no se producirán ciclos de espera por motivo de los retardos de escritura.

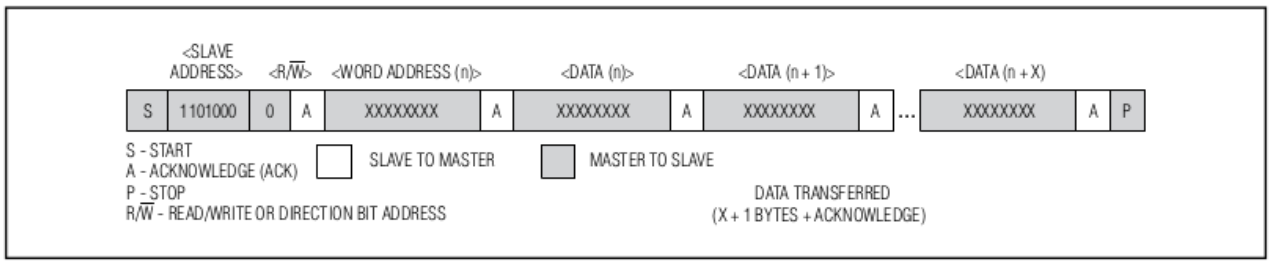


Figure 3. Data Write—Slave Receiver Mode

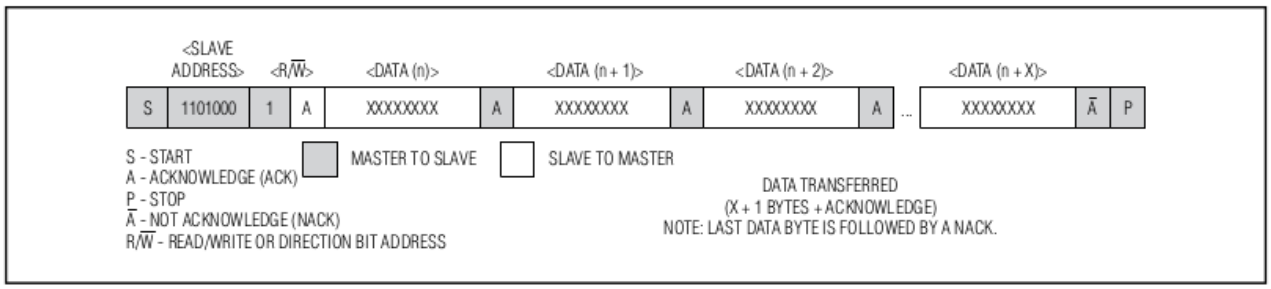


Figure 4. Data Read—Slave Transmitter Mode

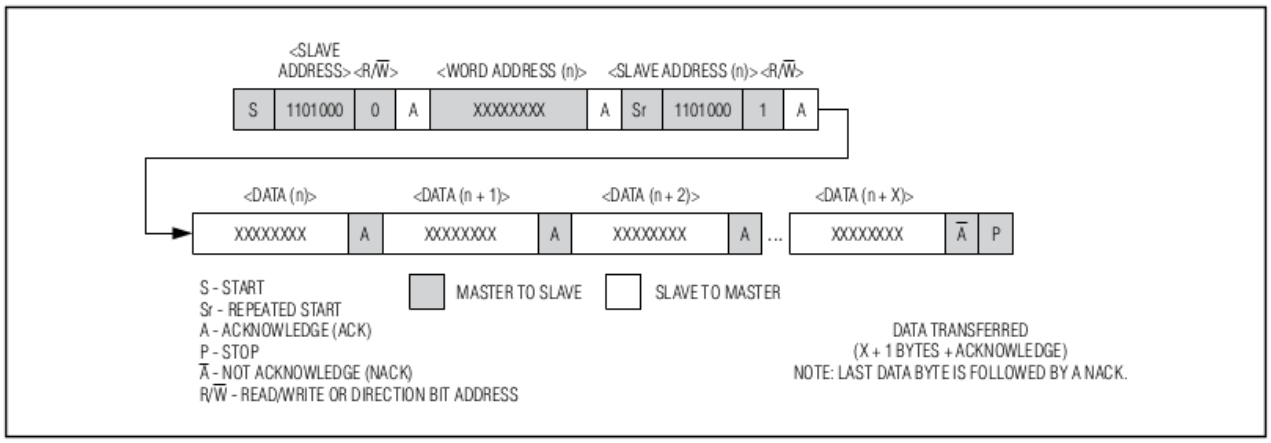


Figure 5. Data Write/Read (Write Pointer, Then Read)—Slave Receive and Transmit