

[UMA](#) / [CV](#) / [Másteres Oficiales de Posgrado](#) / [Mis asignaturas en este Centro](#) / [Curso académico 2025-2026](#)[/ Máster Universitario en Sistemas Electrónicos para Entornos Inteligentes. Plan 2014 \(2025-26\)](#)[/ Del Mundo Físico al Controlador: Sensores, Interfaces y Comunicaciones \(2025-26, Todos los grupos\)](#)[/ BLOQUE 1: Interfaces de conexión de sensores al microcontrolador](#) / [Cuestionario del ejercicio de Evaluación continua EC11a \(1p\)](#)

<b>Comenzado el</b>	viernes, 2 de enero de 2026, 09:16
<b>Estado</b>	Finalizado
<b>Finalizado en</b>	viernes, 2 de enero de 2026, 09:25
<b>Tiempo empleado</b>	8 minutos 51 s
<b>Calificación</b>	Sin calificar aún

**Pregunta 1**

Correcta

Se puntuá 0,10 sobre 0,10

Teniendo en cuenta las características particulares del protocolo de comunicaciones del sensor de humedad SHT11, cómo podría realizarse la conexión y programación de dicho sensor SHT11 al microcontrolador MSP430F5529. Indique de las opciones que se muestran a continuación cuál o cuáles son verdaderas:

Selecciona una o más de una:

- a. Si se quiere hacer uso del módulo que implementa la interfaz I2C en el microcontrolador (USCI en modo I2C), y programar el protocolo de comunicaciones utilizando la librería directamente, es necesario buscar otro sensor en el que el fabricante indique expresamente que soporta las especificaciones I2C. Por ejemplo, el sensor SHT31-DIS del mismo fabricante SENSIRION, sí posee interfaz I2C. ✓ Esta opción es correcta. Es evidente que es la solución más inmediata y sencilla, buscar un sensor que se ajuste al estándar I2C.
- b. Conectando las señales SCK y DATA a terminales GPIO que emulen el comportamiento en colector/drenador abierto y con la conexión de una resistencia de pull-up de unos 10Kohmios para fijar el nivel alto en las líneas. Este comportamiento emulado puede implementarse utilizando la salida triestado de los puertos GPIO del MSP430 (donde el terminal se configuraría como salida para fijar un 0 "fuerte" y se configuraría como entrada para establecer un 1 "débil" mediante la resistencia de pull-up). El protocolo de comunicaciones a nivel de bit se programaría mediante el control de los terminales GPIO (*bit banging*). De forma muy resumida, la programación del protocolo de comunicaciones consistiría en programar el terminal que se conecta al pin DATA del sensor según marque el protocolo en cada momento, bien como entrada o bien como salida. Cuando el terminal sea configurado como entrada se realizarán las lecturas de la línea DATA marcadas por los flancos de subida de la señal de reloj SCK. Cuando el terminal se programe como salida realizará las escrituras de los bits con los flancos de bajada de la señal de reloj SCK. La señal SCK también es controlada por la programación directa de los niveles del terminal GPIO de propósito general configurado como salida en la que se irán programando los niveles que componen el ciclo de reloj conforme al protocolo del sensor. ✓ Esta opción es correcta. La opción *bit banging* es la más habitual cuando la interfaz de conexión no se ajusta en gran medida a las interfaz estándar. En este caso además de no ajustarse las condiciones de arranque y parada, también se realizaba un control de flujo que no estaba basado en la bajada de la señal de reloj por parte del dispositivo esclavo, sino que era el maestro el que mantenía la línea de reloj a nivel bajo hasta que detecta un flanco de bajada en la línea de datos. Esta forma de funcionamiento no sigue la especificación I2C, por lo que la opción más sencilla es implementar mediante la programación de las líneas GPIO las especificaciones de la interfaz del sensor.
- c. Conectando las señales DATA y SCK a las señales SDA y SCL del módulo I2C (USCI en modo I2C) del microcontrolador respectivamente. Tras la conexión, la programación del protocolo de comunicación del I2C se puede realizar utilizando directamente las funciones de la librería del módulo I2C.
- d. Conectando las señales DATA y SCK a las señales SDA y SCL del módulo I2C (USCI en modo I2C) del microcontrolador respectivamente y realizando una programación mixta. En esta programación mixta, las condiciones de arranque y parada, que no cumplen con las especificaciones del I2C, se programarían a nivel de bit configurando los terminales GPIO asociados al I2C como propósito general y programando a nivel de bits los niveles requeridos para cumplir con estas condiciones según especifique el sensor. Mientras que el resto del protocolo de comunicación con el sensor SHT11 se podría programar haciendo uso de las funciones de la librería del módulo I2C.
- e. Conectando las señales SCK y DATA a terminales GPIO que emulen el comportamiento en colector/drenador abierto y con la conexión de una resistencia de pull-up de unos 10Kohmios para fijar el nivel alto en las líneas. Este comportamiento emulado puede implementarse utilizando la salida triestado de los puertos GPIO del MSP430 (donde el terminal se configuraría como salida para fijar un 0 "fuerte" y se configuraría como entrada para establecer un 1 "débil" mediante la resistencia de pull-up). El protocolo de comunicaciones a nivel de bit se programaría mediante el control de los terminales GPIO (*bit banging*). De forma muy resumida, la programación del protocolo de comunicaciones consistiría en programar el terminal que se conecta al pin DATA del sensor según marque el protocolo en cada momento, bien como entrada o bien como salida. Cuando el terminal sea configurado como entrada se realizarán las lecturas de la línea DATA marcadas por los flancos de subida de la señal de reloj SCK. Cuando el terminal se programe como salida realizará las escrituras de los bits con los flancos de bajada de la señal de reloj SCK. La señal SCK podría ser generada a través de un terminal del microcontrolador con salida PWM, ya que se necesita que tenga una frecuencia determinada y un ciclo de trabajo del 50%. Otra opción para la señal SCK, sería utilizar, configurándola debidamente, alguna de las señales de reloj del microcontrolador (ACLK, SMCLK).
- f. Respuesta en blanco -> SIN PUNTUACIÓN

Respuesta correcta

El sensor SHT11 no cumple con las especificaciones del estándar I2C, por lo tanto la solución más sencilla es implementar el protocolo definido por el sensor a través de la programación de las líneas GPIO, a este tipo de solución se la denomina (*bit banging*).

En caso de querer hacer uso de la implementación hardware del bus I2C en el microcontrolador, así como de la librería de operación de este bus, es necesario cambiar de sensor. Una posible alternativa es el sensor SHT31

**Pregunta 2**

Correcta

Se puntuá 0,05 sobre 0,05

Atendiendo a las especificaciones de las direcciones I2C y teniendo en cuenta que la operación de escritura de un byte en sensor es la que aparece en la *Imagen 1*. Indique el rango de direcciones que admite el sensor.

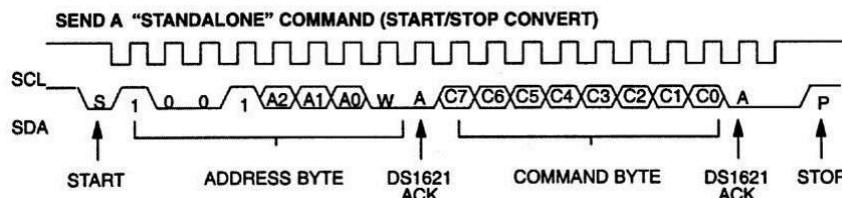


Imagen 1. Operación de escritura

Selecciona una:

- a. 0x90, 0x92, 0x94, 0x96, 0x98, 0x9A, 0x9C, 0x9E
- b. 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F ✓ Respuesta correcta
- c. 0x84, 0x94, 0xA4, 0xB5, 0xC4, 0xD4, 0xE4, 0xF4
- d. Respuesta en blanco -> SIN PUNTUACIÓN
- e. 0x91, 0x93, 0x95, 0x97, 0x99, 0x9B, 0x9D, 0x9F
- f. 0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97

Respuesta correcta

El rango de direcciones que puede tener el dispositivo va desde 0x48 a 0x4F, un total de 8 direcciones distintas.

Tal y como se ve en la Imagen 1, el primer byte que manda el maestro corresponde, con los 7 bits de dirección del dispositivo esclavo y el bit de operación lectura/escritura. La dirección de los dispositivos esclavos en I2C tiene 7 bits, por lo tanto la dirección será 100 1A<sub>2</sub> A<sub>1</sub> A<sub>0</sub>. Expresando la dirección en un byte queda 0100 1A<sub>2</sub> A<sub>1</sub> A<sub>0</sub>, lo que define el conjunto de direcciones: 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F

**Pregunta 3**

Correcta

Se puntuá 0,05 sobre 0,05

Para asignar una dirección I2C al sensor DS1621 es necesario realizar la conexión de los terminales de dirección ( $A_2$ ,  $A_1$  y  $A_0$ ). Si estos terminales se conectan tal y como se ve en la *Imagen 2*, qué dirección se le asigna al sensor DS1621 (valor del define dado por la etiqueta SLAVE\_ADDRESS\_DS1621).

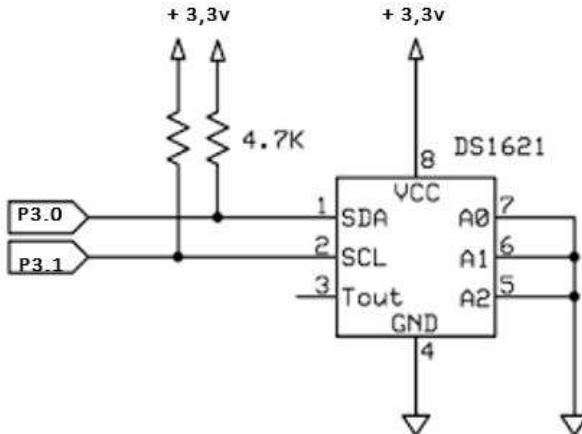


Imagen 2. Asignación de dirección del DS1621 mediante la conexión de los terminales ( $A_2$ ,  $A_1$  y  $A_0$ ) a GND

Selecciona una:

- a. 0x09
- b. Respuesta en blanco -> SIN PUNTUACIÓN
- c. 0x48 ✓ Respuesta correcta
- d. 0x84
- e. 0x19
- f. 0x91
- g. 0x90

Respuesta correcta

El byte de direccionamiento (el primer byte que manda el maestro tras la condición de parada) consta de los 7 bits de dirección del dispositivo esclavo y del bit de operación lectura/escritura. La dirección de los dispositivos esclavos en I2C tiene 7 bits, por lo tanto la dirección será 100 1A2 A1 A0, Expresando la dirección en un byte queda 0100 1A2 A1 A0, como los terminales de dirección están todos a 0, el byte de dirección es 0100 1000, que se corresponde con el valor hexadecimal 0x48.

**Pregunta 4**

Correcta

Se puntuá 0,10 sobre 0,10

Los terminales SDA y SCL del sensor DS1621 se han conectado a los terminales del módulo I2C USCI\_B0 del microcontrolador MSP430F5529 P3.0 y P3.1 respectivamente. Como hemos visto se podrían conectar a otro módulo I2C del microcontrolador. Para esta otra USCI indique las etiquetas que se utilizarán en las funciones de configuración de los terminales SDA y SCL del microcontrolador y en la que programa al microcontrolador en modo maestro :

```
GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_PZ, GPIO_PINX + GPIO_PINY);  
GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_PZ, GPIO_PINX + GPIO_PINY);  
USCI_B_I2C_initMaster(USCI_TW_BASE, &param);
```

GPIO\_PINX terminal de la línea SDA

GPIO\_PINY terminal de la línea SCL

Completa las funciones identificando los valores Z, X, Y, T y W en las etiquetas de las funciones anteriores:

**IMPORTANTE:** TIENE QUE INTRODUCIR LA ETIQUETA NO SOLO EL VALOR DE LAS INCÓGNITAS, se trata de hacer funciones que compilen.

```
GPIO_setAsInputPinWithPullUpResistor( GPIO_PORT_P4 ✓ , GPIO_PIN1 ✓ + GPIO_PIN2 ✓ );  
GPIO_setAsPeripheralModuleFunctionInputPin( GPIO_PORT_P4 ✓ , GPIO_PIN1 ✓ + GPIO_PIN2 ✓ );  
USCI_B_I2C_initMaster( USCI_B1_BASE ✓ , &param);
```

La USCI alternativa que se podría usar es la USC1\_B1, los terminales asociados son los terminales P4.1 para la señal SDA y P4.2 para la señal SCL.

Haciendo uso de las etiquetas las funciones quedarían programadas de la siguiente forma:

```
GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4, GPIO_PIN1 + GPIO_PIN2);  
GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P4,GPIO_PIN1 + GPIO_PIN2);  
USCI_B_I2C_initMaster(USCI_B1_BASE, &param);
```

**Pregunta 5**

Correcta

Se puntuá 0,10 sobre 0,10

En las especificaciones del bus I2C se indica que las etapas de salida de los dispositivos deben estar conectadas al bus I2C mediante una etapa en colector abierto o drenador abierto, de forma que estas etapas fijen el nivel bajo, mientras el nivel alto se fije a través de una resistencia de pull-up conectada a  $V_{DD}$  y al bus I2C. En principio el valor de la resistencia debe ser tal que permita fijar el nivel bajo respetando las especificaciones eléctricas de salida ( $I_{OL}$  máxima corriente que puede absorber la salida manteniendo el nivel lógico 0).

Un error que se suele cometer es colocar una resistencia de pull-up por cada dispositivo, **¿qué efecto puede tener si una vez el bus está funcionando añadimos nuevos dispositivos, añadiendo también sus resistencias de pull-up? Tenga en cuenta que, tal y como se ha indicado en el Documento "Interfaces serie síncronos: I2C y SPI", el bus I2C se puede ver como un circuito RC, en el que el tiempo de conmutación del nivel 0 al nivel 1 en las líneas del bus depende de la resistencia y de la capacidad de este. Marque aquellas afirmaciones que considere que son ciertas.**

Selecciona una o más de una:

- |   |   |
|---|---|
| <input checked="" type="checkbox"/> a. El bus se vuelve más rápido ya que el tiempo de subida en las líneas (SDA y SCL) (paso del nivel bajo al nivel alto) es menor al ser la resistencia menor  | <span style="color: green;">✓</span> Esta afirmación es correcta.<br>El bus se puede ver como un filtro RC, por ello el tiempo de subida depende de la constante RC, si la resistencia del bus es mas pequeña, el tiempo de subida será también menor y por ello el bus se vuelve más rápido.   |
| <input checked="" type="checkbox"/> b. La resistencia resultante es menor y la corriente que el dispositivo tiene que absorber es mayor para fijar el nivel bajo. Si esa corriente es mayor que $I_{OL}$ podría no fijar el nivel bajo adecuadamente. | <span style="color: green;">✓</span> La resistencia resultante es menor, el consumo a nivel bajo es mayor, ya que se necesita más corriente para fijar un nivel 0 en la línea. Además, también hay que tener en cuenta la máxima corriente que puede absorber la salida manteniendo el nivel lógico 0 ( $I_{OL}$ ), si se supera este valor de corriente, porque la resistencia se haga más pequeña, puede no fijarse el nivel bajo adecuadamente, lo que hará que no se cumplan las especificaciones I2C |
| <input checked="" type="checkbox"/> c. La resistencia resultante es menor y por lo tanto el consumo a nivel bajo es mayor   | <span style="color: green;">✓</span> Esta afirmación es correcta.<br>La resistencia resultante es menor y por lo tanto el consumo a nivel bajo es mayor, ya que se necesita más corriente para fijar un nivel 0 en la línea   |
| <input type="checkbox"/> d. El bus se vuelve más lento ya que el tiempo de subida en las líneas (SDA y SCL) (paso del nivel bajo al nivel alto) es mayor al ser la resistencia resultante mayor   |   |
| <input type="checkbox"/> e. La resistencia resultante es mayor y por lo tanto el consumo a nivel bajo es menor  |   |
| <input type="checkbox"/> f. No afecta   |   |
| <input type="checkbox"/> g. Respuesta en blanco -> SIN PUNTUACIÓN   |   |

Respuesta correcta

Cuando se colocan más resistencias de pull-up, la resistencia resultante de una línea (SDA o SCL) es el paralelo de las resistencias existentes y por lo tanto el valor de esa resistencia resultante es menor. El tiempo de conmutación del nivel 0 al nivel 1 en las líneas del bus depende de la resistencia y de la capacidad de este. De esta forma si la resistencia o la capacidad disminuye también disminuye el tiempo de subida y por lo tanto el bus se hace más rápido. Por otra parte, como la resistencia resultante es menor, el consumo a nivel bajo es mayor, ya que se necesita más corriente para fijar un nivel 0 en la línea. Además, también hay que tener en cuenta la máxima corriente que puede absorber la salida manteniendo el nivel lógico 0 (  $I_{OL}$ ), si se supera este valor de corriente, porque la resistencia se haga más pequeña, puede no fijarse el nivel bajo adecuadamente, lo que hará que no se cumplan las especificaciones I2C ( se creerá que se está fijando un 0 en la línea para marcar cualquier condición del bus y en realidad se estará fijando un nivel 1)

**Pregunta 6**

Correcta

Se puntuá 0,10 sobre 0,10

Indique a qué velocidad se ha programado la señal de reloj del bus I2C (SCL) mediante la función `USCI_B_I2C_initMaster(USCI_B0_BASE, &param)`, del código `DS1621_usci_b_i2c.c`. Indique igualmente qué velocidad máxima admite la señal SCL del módulo I2C del microcontrolador y qué velocidad admite la señal de reloj SCL del sensor I2C DS1621.

Velocidad programada por la función `USCI_B_I2C_initMaster()` para SCL = 100 ✓ KHz

Velocidad máxima de SCL soportada por el I2C del microcontrolador = 400 ✓ KHz

Velocidad máxima de SCL soportada por el sensor I2C DS1621 = 400 ✓ KHz

La frecuencia a la que está programada el reloj del maestro I2C es la que se define en la estructura de tipo `USCI_B_I2C_initMasterParam`. Esta estructura tiene diferentes campos que hay que llenar y que están relacionados con la tasa de la señal SCL. Entre los distintos campos que se deben asignar se encuentra la definición de la fuente de reloj utilizada por el generador de reloj del I2C, que en este caso es la señal de reloj SMCLK, así como el valor a la que se ha programado, que en este caso es su valor por defecto, esto es, aproximadamente un 1Mhz. En el código el valor al campo `i2cClk` se asigna con la función `UCS_getSMCLK()`, que lo hace es leer el valor de los bits del registro que programa el reloj SMCLK para ver a qué frecuencia se encuentra.

La etiqueta `USCI_B_I2C_SET_DATA_RATE_100KBPS` define que la frecuencia de la señal SCL sea 100KHz. Dividiendo la frecuencia de la señal de reloj SMCLK por el valor de esta etiqueta se obtiene el valor del divisor que actúa en el generador de reloj del I2C para obtener la señal SCL de aproximadamente 100KHz

```
//Initialize Master I2C
USCI_B_I2C_initMasterParam param = {0};

param.selectClockSource = USCI_B_I2C_CLOCKSOURCE_SMCLK;
param.i2cClk = UCS_getSMCLK();

param.dataRate = USCI_B_I2C_SET_DATA_RATE_100KBPS;

USCI_B_I2C_initMaster(USCI_B0_BASE, &param);
```

En el caso del microcontrolador MSP430F5529, la frecuencia máxima que soporta el I2C implementado aparece detallada en el datasheet del microcontrolador en la página 39, en la que aparece como máximo los 400KHz, que corresponde con el modo FAST del I2C

### 5.34 USCI (I<sup>2</sup>C Mode)

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted) (see Figure 5-15)

PARAMETER	TEST CONDITIONS	V <sub>CC</sub>	MIN	MAX	UNIT	
f <sub>USCI</sub>	USCI input clock frequency Internal: SMCLK, ACLK External: UCLK Duty cycle = 50% ± 10%			f <sub>SYSTEM</sub>	MHz	
f <sub>SCL</sub>	SCL clock frequency	2.2 V, 3 V	0	400	KHz	
t <sub>HOLD,STA</sub>	Hold time (repeated) START	f <sub>SCL</sub> ≤ 100 kHz f <sub>SCL</sub> > 100 kHz	2.2 V, 3 V	4.0 0.6	μs	
t <sub>SU,STA</sub>	Setup time for a repeated START	f <sub>SCL</sub> ≤ 100 kHz f <sub>SCL</sub> > 100 kHz	2.2 V, 3 V	4.7 0.6	μs	
t <sub>HOLD,DAT</sub>	Data hold time		2.2 V, 3 V	0	ns	
t <sub>SU,DAT</sub>	Data setup time		2.2 V, 3 V	250	ns	
t <sub>SU,STOP</sub>	Setup time for STOP	f <sub>SCL</sub> ≤ 100 kHz f <sub>SCL</sub> > 100 kHz	2.2 V, 3 V	4.0 0.6	μs	
t <sub>SP</sub>	Pulse duration of spikes suppressed by input filter		2.2 V 3 V	50 50	600 600	ns

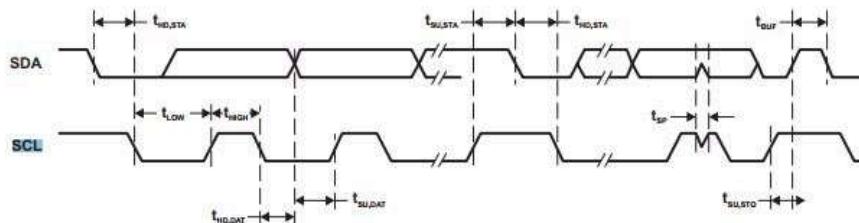


Figure 5-15. I<sup>2</sup>C Mode Timing

En el caso del sensor DS1621 en su datasheet en la página 15 podemos encontrar que la máxima frecuencia de reloj que soporta para la comunicación I2C es 400KHz. El sensor está preparado para soportar los dos modos ( standard y fast)

<b>AC ELECTRICAL CHARACTERISTICS</b>		(-55°C to +125°C; V <sub>DD</sub> = 2.7V to 5.5V)					
PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Temperature Conversion Time	T <sub>TC</sub>				750	ms	
NV Write Cycle Time	t <sub>WR</sub>	0°C to 70°C		4	10	ms	10
SCL Clock Frequency	f <sub>SCL</sub>	Fast Mode Standard Mode	0 0		400 100	KHz	

**Pregunta 7**

Correcta

Se puntuá 0,10 sobre 0,10

Atendiendo al *Código 1* que se corresponde con el del fichero DS1621\_usci\_b\_i2c.c se puede observar que en primer lugar se realiza la configuración del DS1621 mediante el envío por parte del maestro de un comando ACCESS CONFIG que permite la configuración del registro de configuración. Esta operación de configuración conlleva la escritura de dos bytes, uno que corresponde al comando [1], y otro al dato que se desea escribir en el registro de estado/configuración [2]. Tal y como se ha comentado **se quiere programar la conversión continua de temperatura por parte del sensor**. Tras la configuración se manda el comando START CONVERT T que se encarga de disparar la conversión de temperatura.

Busca las funciones de la **librería usci\_b\_i2c** que se utilizan para implementar las operaciones antes comentadas y observa la *Imagen 3* que se corresponde con el protocolo serie que sigue el sensor DS1621, así como la *Imagen 4* que corresponde con la captura del bus I2C con el analizador lógico.

Indica el valor de los comandos y el valor con el que se debe configurar el registro de estado/configuración. Rellena los valores en formato hexadecimal de las siguientes etiquetas:

- K\_DS1621\_ACCEES\_CONFIG=0x AC ✓
- K\_DS1621\_CONTINUOUS\_CONFIG=0x 00 ✓
- K\_DS1621\_START\_CONVERT=0x EE ✓

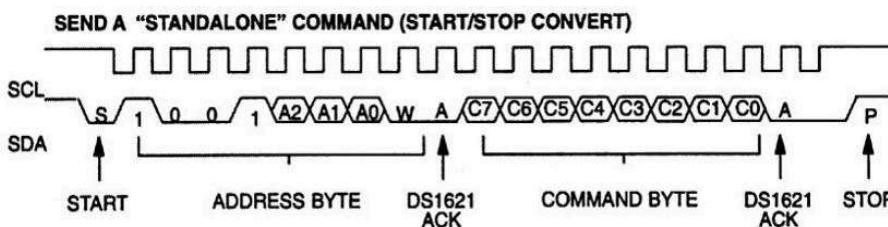
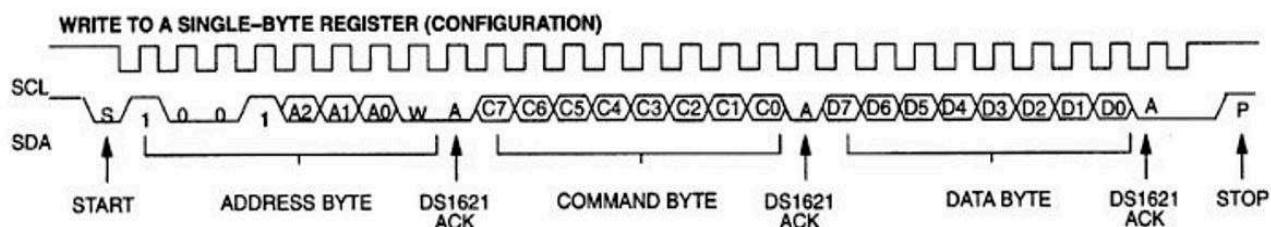


Imagen 3. Protocolo serie de los comandos ACCESS CONFIG y START CONVERT T

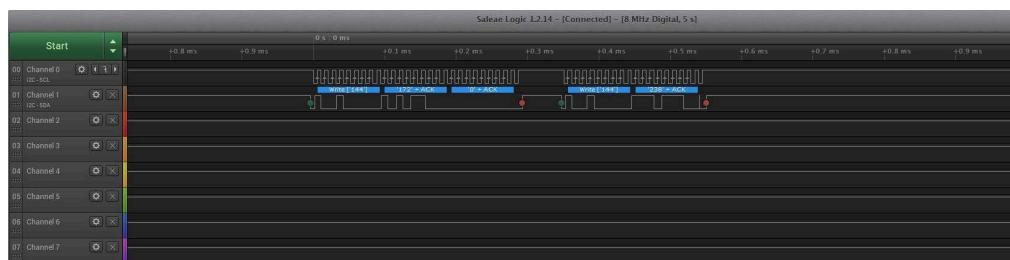


Imagen 4. Captura del bus I2C con el analizador lógico. Configuración y comando START\_COMMAND

[1] Otra manera de ver al comando es como la dirección del registro sobre el que se va a escribir

[2] La escritura sobre el registro de estado/configuración solo tiene efecto en los bits que pueden ser programados, esto es, bit 0 y bit 1.

Los valores de las etiquetas y el valor de configuración del registro se puede extraer directamente de las capturas, o bien acudiendo al datasheet del sensor.

Para el valor de la etiqueta K\_DS1621\_ACCEES\_CONFIG se puede observar en la captura que se envía primero el byte de dirección con operación de escritura que en binario es 10010000 (144 en decimal) y a continuación se manda el comando ACCES\_CONFIG que en decimal es 172 y en hexadecimal es 0xAC. Este comando también se pude ver en la página 11 del datasheet del sensor.

```
#define K_DS1621_ACCEES_CONFIG 0xAC
```

Para el valor de la etiqueta K\_DS1621\_CONTINUOUS\_CONFIG, en este caso es necesario codificar en el registro de estado/configuración el disparo continuo y por ello hay que poner el bit 0 a 0, el resto como 0 no se utiliza o la escritura no tiene impacto los ponemos también a 0. Por lo tanto:

```
K_DS1621_CONTINUOUS_CONFIG=0x00
```

Por último para el valor de la etiqueta K\_DS1621\_START\_CONVERT se puede observar en la captura que se envía primero el byte de dirección con operación de escritura 144 en decimal y a continuación se manda el comando START\_CONVERT\_T que en decimal es 238 y en hexadecimal es 0xEE. Este comando también se pude ver en la página 11 del datasheet del sensor.

**Pregunta 8**

Correcta

Se puntúa 0,10 sobre 0,10

Indique el valor aproximado (etiqueta TEMP\_CONVERSION\_TIME) en ciclos de reloj que se le debe pasar a la función `__delay_cycles(TEMP_CONVERSION_TIME)` para que antes de realizar la lectura se garantice que ha transcurrido el tiempo de conversión de la temperatura que especifica el fabricante. Para determinar este tiempo consulta las características eléctricas en el *datasheet* del sensor.

Respuesta: 786432



Si se accede al datasheet del sensor DS1621 se puede observar que el tiempo máximo que tarda el sensor en realizar una conversión son 750ms. Este tiempo máximo es el que hay que considerar porque supone el caso peor del conversor. Por lo tanto, sabiendo que el sensor está a 1MHz aproximadamente, en concreto está a 1048576 Hz, para garantizar que el dato está convertido antes de leerlo tendríamos que poner 786432 ciclos.

En esta pregunta se ha dado un margen de 100000 ciclos por arriba y por abajo para que la contestación sea correcta, por lo tanto esta pregunta considera correcto cualquier valor entre 686432 y 886432.

Es evidente que esperar más de 786432, no tiene sentido porque el máximo es 750ms.

También es evidente que al poner 786432 ciclos la espera será un poco mayor, ya que antes de leer la temperatura hay instrucciones en el programa que tardan un tiempo en ejecutarse.

DS1621

AC ELECTRICAL CHARACTERISTICS		(-55°C to +125°C; V <sub>DD</sub> = 2.7V to 5.5V)					
PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Temperature Conversion Time	T <sub>TC</sub>				750	ms	
NV Write Cycle Time	t <sub>WR</sub>	0°C to 70°C		4	10	ms	10
SCL Clock Frequency	f <sub>SCL</sub>	Fast Mode Standard Mode	0		400 100	KHz	

**Pregunta 9**

Correcta

Se puntuá 0,15 sobre 0,15

Las operaciones que se indican en el datasheet del sensor para obtener la temperatura (2 bytes de los cuales solo los 9 más significativos tienen información) y los parámetros Count\_Remain y Count\_Per\_C que permiten calcular la temperatura con mayor resolución son los que se muestran en la *Imagen 5*.

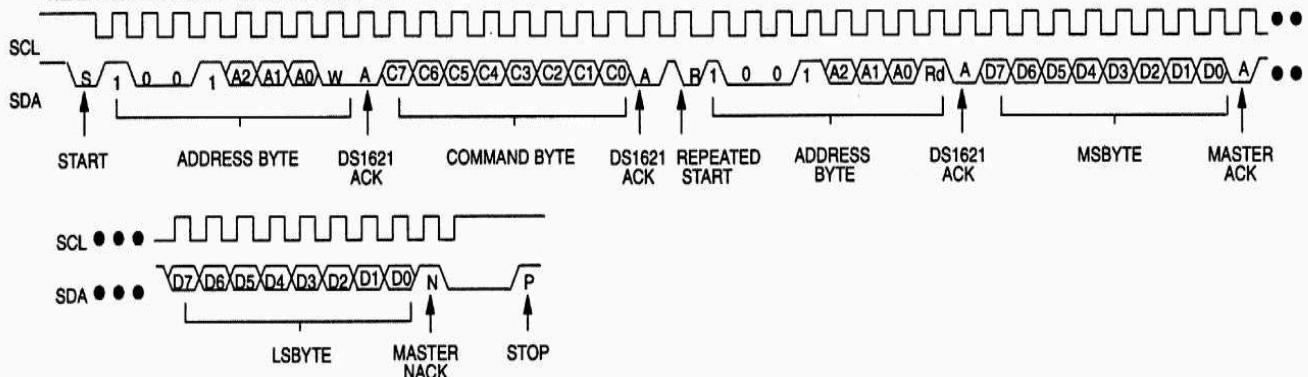
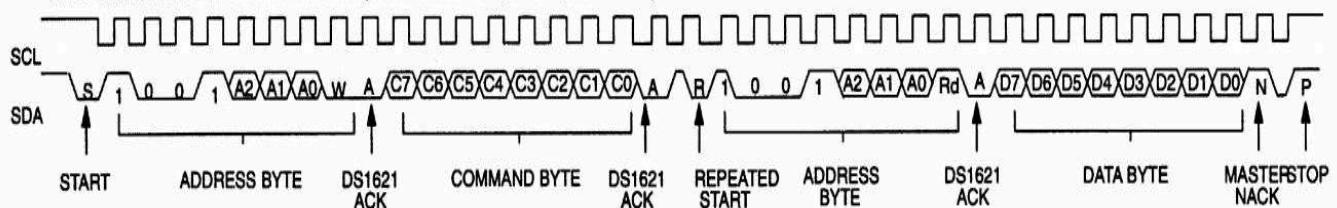
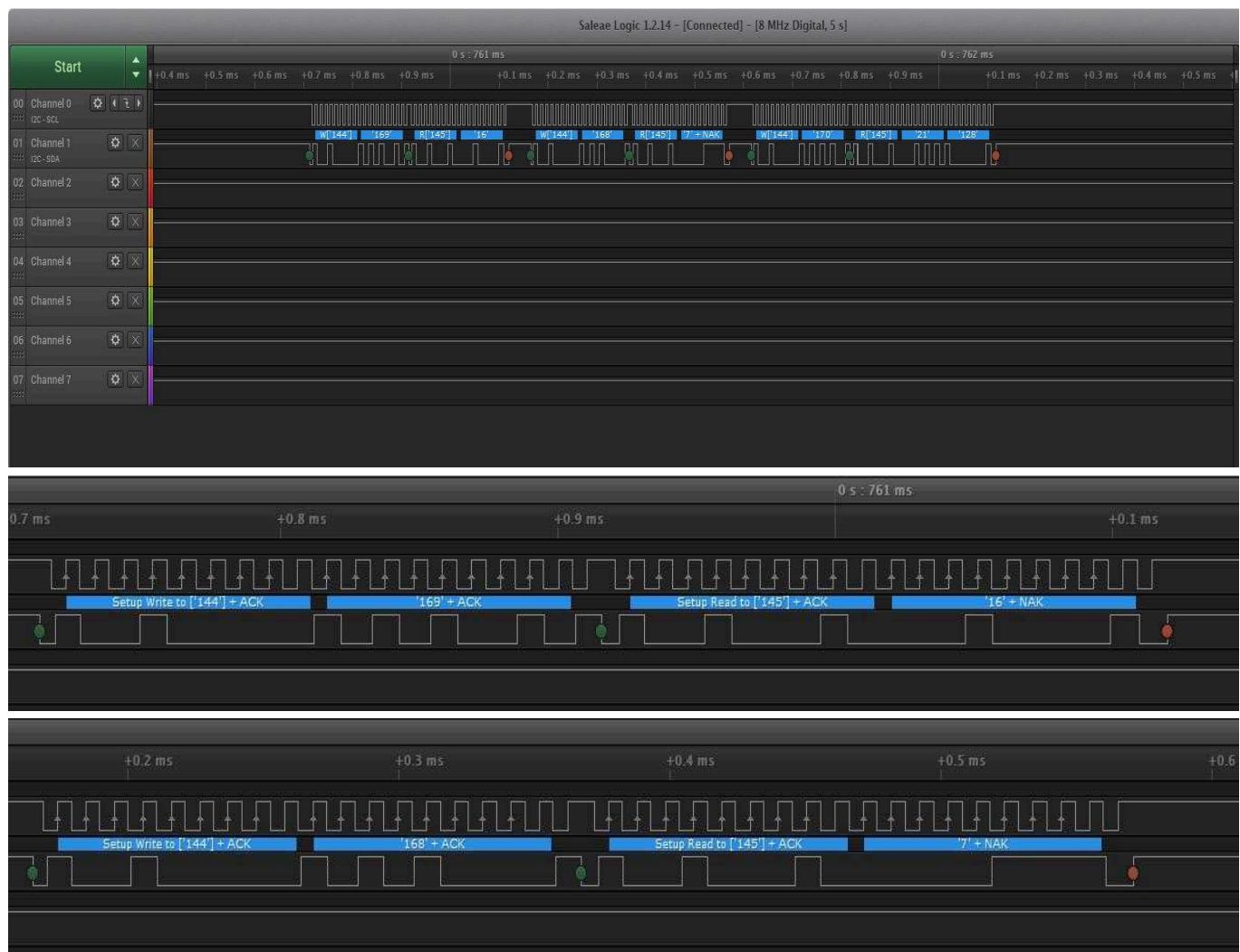
**READ FROM A TWO-BYTE REGISTER ( $T_H, T_L$ , TEMPERATURE)****READ FROM A SINGLE-BYTE REGISTER (CONFIGURATION, SLOPE, COUNTER)**

Imagen 5. Protocolo serie de los comandos READ TEMPERATURE, READ COUNT y READ SLOPE



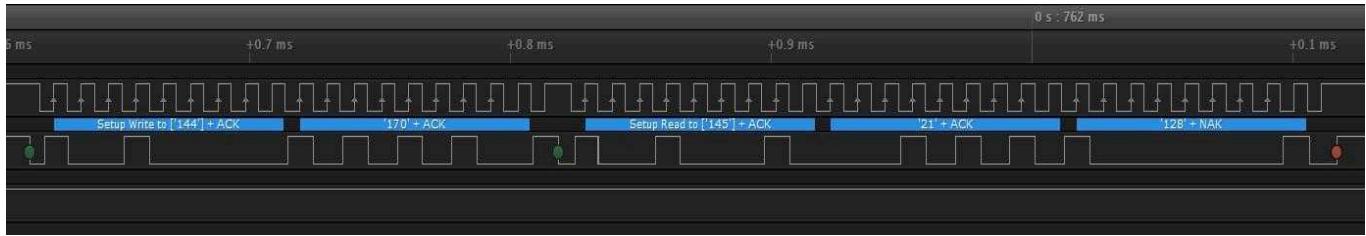


Imagen 6. Captura del bus I2C con el analizador lógico. Comandos para el cálculo de la temperatura de mayor resolución

En la Imagen 6 se presenta la captura del bus I2C cuando se han enviado estos comandos tal y como se indican en el código del fichero DS1621\_usci\_b\_i2c.c.

Analizar las funciones de la de la **librería usci\_b\_i2c** bajo las que se implementan estas operaciones de escritura y **rellenar los valores en formato hexadecimal de las siguientes etiquetas:**

- K\_DS1621\_READ\_TEMP=0x AA ✓
- K\_DS1621\_READ\_COUNTER=0x A8 ✓
- K\_DS1621\_READ\_SLOPE=0x A9 ✓

**Calcula el valor de la temperatura y de la temperatura de mayor resolución que se obtendría según los datos leídos en la Imagen 6 (variables temp\_DS1621 y temp del código DS1621\_usci\_b\_i2c.c).**

temp\_DS1621= 21.5 ✓ °C

temp= 21.3125 ✓ °C

Los valores de las etiquetas se pueden extraer directamente de las capturas, o bien acudiendo al datasheet del sensor. En el caso de las temperaturas, para poder obtener sus valores hay que interpretar los datos de la captura, y además, consultar el datasheet o el código DS1621\_usci\_b\_i2c.c para determinar cómo se calculan dichas temperaturas.

De la primera imagen de la captura se observa que hay tres operaciones de escritura-lectura. Una para leer un byte correspondiente al valor del parámetro Count\_Per\_C , otra para leer byte correspondiente al parámetro Count\_Remain, y por último la operación de lectura de la temperatura con dos bytes. A su vez, cada una de estas operaciones comienza con el byte de dirección con operación de escritura que en binario es 10010000 (144 en decimal) y a continuación se manda el comando que determina en cada caso qué se quiere leer, estos son READ SLOPE, READ COUNTER y READ TEMPERATURE respectivamente. Por ello, los valores de las etiquetas planteadas serán:

K\_DS1621\_READ\_SLOPE= 169 que en hexadecimal es 0xA9

K\_DS1621\_READ\_COUNTER=168 que en hexadecimal es 0xA8

K\_DS1621\_READ\_TEMP=170 que en hexadecimal es 0xAA

Estos valores están también recogidos en el datasheet del sensor en la página 11.

En el caso de la temperatura, como se ha comentado más arriba, es necesario observar los dos bytes que se reciben tras el comando de READ TEMPERATURE. En la imagen que captura la lectura de la temperatura, se ve que el primer byte (byte más significativo) que se recibe tiene el siguiente valor en binario 00010101, mientras que el valor en binario del segundo byte es 1000 0000. Tal y como se recoge en el datasheet del sensor, la temperatura se expresa con 9 bits que se corresponden con la temperatura de la siguiente forma: el valor del byte más alto representa el valor de la temperatura en °C, mientras que el byte más bajo, del que solo es significativo el bit más alto, el bit 7, expresa la resolución de 0,5°C de temperatura. De esta forma el valor de la temperatura sería de 21°C + 0,5°C (el bit 7 del byte menos significativo está a 1) . Por lo tanto, temp\_DS1621=21,5°C

En el caso de la TEMPERATURA de mayor resolución es necesario obtener los valores de los parámetros Count\_Per\_C , y Count\_Remain, que en la captura se puede observar que son 16 y 7 respectivamente. Con la fórmula que da el fabricante en el datasheet del sensor para calcular la temperatura de mayor resolución en base a estos dos parámetros es:

temp=ucRegData[0]-0.25+((float)(Count\_Per\_C-Count\_Remain))/(float)Count\_Per\_C;

donde la ucRegData[0] es el valor de la temperatura sin considerar el bit 7 del byte menos significativo, con ello se obtiene un valor de 21,3125°C.

Si se considera el valor de la temperatura sin truncar, esto es:

temp=temp\_DS1621-0.25+((float)(Count\_Per\_C-Count\_Remain))/(float)Count\_Per\_C;

se obtendría un valor de 21,8125°C.

El ejercicio da ambas soluciones como válidas pero la correcta es la que parte de la temperatura sin considerar el bit 7 del byte menos significativo, ya que lo que pretende el cálculo de la temperatura de mayor resolución es definir de forma más precisa la temperatura en los 0,5°C que define el bit 7 del byte menos significativo.

**Pregunta 10**

Finalizado

Valor: 0,15

La lectura de los parámetros Count\_Remain y Count\_Per\_C se realiza, entre otras, con la función USCI\_B\_I2C\_masterReceiveSingle(). La función original es la que aparece en la *Imagen 7* y la modificada para que el programa funcione correctamente es la que aparece en la *Imagen 8*.

```
uint8_t USCI_B_I2C_masterReceiveSingle (uint16_t baseAddress)
{
    //Polling RXIFG0 if RXIE is not enabled
    if(!(HWREG8(baseAddress + OFS_UCBxIE) & UCRXIE)) {
        while(!(HWREG8(baseAddress + OFS_UCBxIFG) & UCRXIFG));
    }

    //Read a byte.
    return (HWREG8(baseAddress + OFS_UCBxRXBUF));
}
```

Imagen 7. Función USCI\_B\_I2C\_masterReceiveSingle() original

```
uint8_t USCI_B_I2C_masterReceiveSingle (uint16_t baseAddress)
{
    //Polling RXIFG0 if RXIE is not enabled
    if(!(HWREG8(baseAddress + OFS_UCBxIE) & UCRXIE)) {
        while(((!(HWREG8(baseAddress + OFS_UCBxIFG) & UCRXIFG))&& (!(HWREG8(baseAddress +
OFS_UCBxIFG) & UCNACKIFG)));
    }

    //Read a byte.
    return (HWREG8(baseAddress + OFS_UCBxRXBUF));
}
```

Imagen 8 Función USCI\_B\_I2C\_masterReceiveSingle() modificada

De igual forma, aunque no en las funciones de la librería, esta modificación es introducida en la línea 101 del Código 1, después de la función USCI\_B\_I2C\_masterReceiveMultiByteStart() dentro del bucle de lectura de la temperatura. La motivación/necesidad de esta línea es la misma que la de la USCI\_B\_I2C\_masterReceiveSingle(). El tratamiento de esta línea se hace con el if-else que hay en las líneas de la 103 a la 123 del Código 1.

**¿Qué intenta solventar esta modificación? ¿En qué circunstancias es necesaria? Intenta indicar una situación en la que si no se pone esta condición el bus se quedaría bloqueado.**

Para poder responder a esta pregunta se van a presentar una serie de capturas realizadas con el analizador lógico que pueden hacer entender por qué es necesaria la modificación indicada anteriormente. En las siguientes capturas que aparecen en la Imagen 9, se puede observar como la comunicación I2C con el sensor se rompe, quedando el sistema bloqueado en la función **USCI\_B\_I2C\_masterReceiveSingle()**.

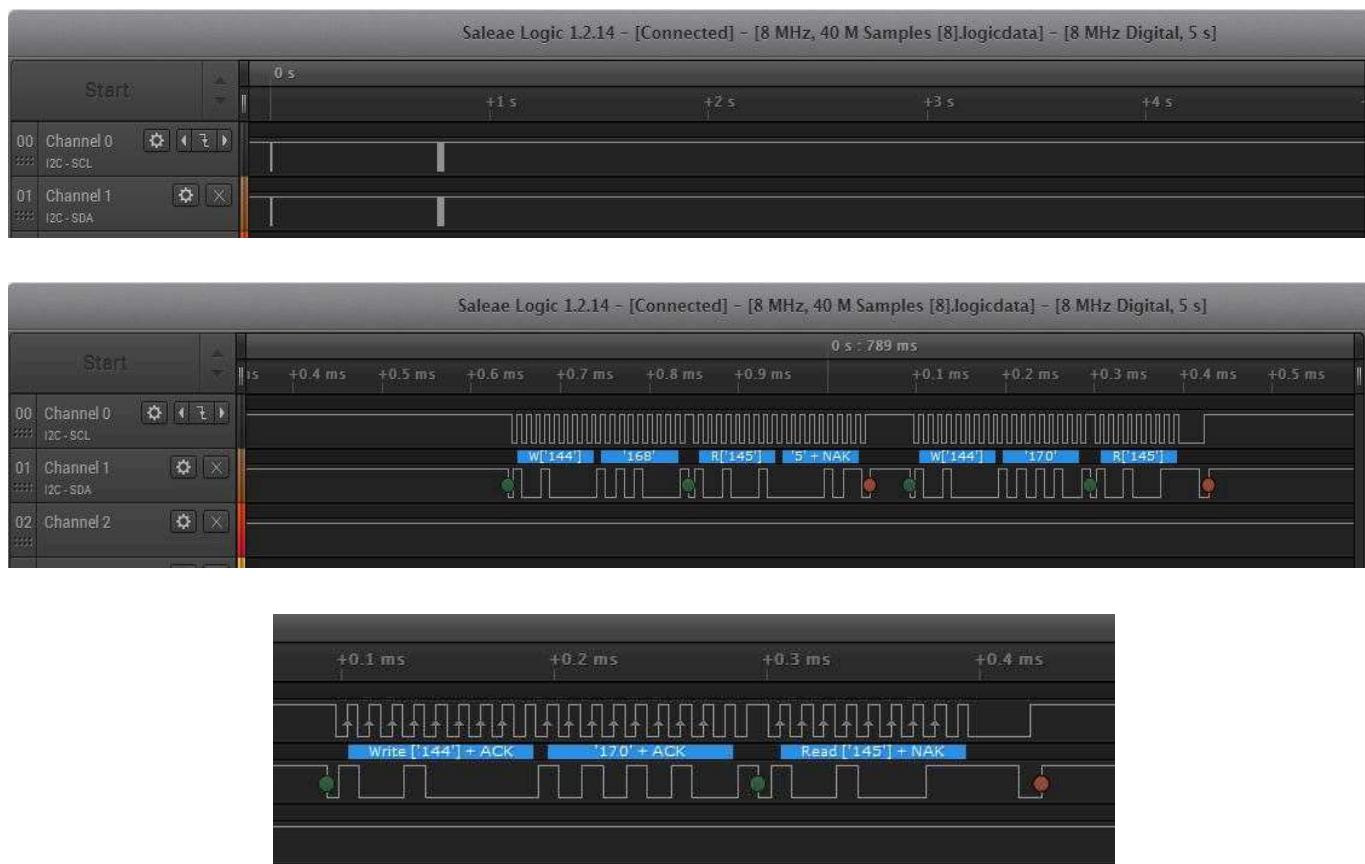


Imagen 9. Capturas del bus I2C que muestran el efecto de no modificar la función USCI\_B\_I2C\_masterReceiveSingle(), ni incluir la línea 101 tras la función USCI\_B\_I2C\_masterReceiveMultiByteStart()

Intenta evitar el bloqueo por software (hang) del microcontrolador. Sin esta modificación, el código asume ciegamente que el hardware siempre recibirá un byte. La versión modificada permite que el código continúe su ejecución incluso si hay un fallo en la comunicación, permitiendo al programa detectar el error (mediante el if-else mencionado en el Código 1) y tomar medidas correspondientes, como reiniciar el bus o avisar de un fallo de lectura.

◀ Captura 2 sensor DS1621

Saltar a...

Configuración del módulo SPI del MSP430F5529 para su conexión a los dispositivos MCP4261 y MCP3008 ►