ESE381 Embedded Microprocessor Systems Design II

Spring 2024, K. Short
revised April 10, 2024 9:18 pm

PRELIMINARY PRELIMINARY PRELIMINARY PRELIMINARY PRELIMINARY

Laboratory 10: Serial Alarm Real-Time Clock

To be performed the week starting April 14th.

Prerequisite Reading

1. Maxim Integrated DS3231 Extremely Accurate I2C - Integrated RTC/TCXO/Crystal data sheet.
2. Adafruit DS3231 Precision RTC Breakout data sheet.
3. Maxim Integrated *Tutorial 5791 Tips for Writing Bulletproof Real-Time Clock Control Code* (on Brightspace).
4. Voltage and Current Interface Compatibility Checklist (on Brightspace).

Overview

One approach to providing time of day and date information in an embedded system is to use a real-time clock (RTC) IC. In addition to providing time and date information for display on a user interface this information can be used to "time stamp" data measurements made by the embedded system. One popular real-time clock IC is Maxim's DS3231, which has an I2C serial interface.

The DS3231 uses a built-in crystal oscillator as it time base. The crystal is inside the chip. Next to the integrated crystal is a temperature sensor. The temperature sensor is used to compensate for temperature dependent frequency changes by adding or removing clock ticks, so that the time-keeping remains accurate. The 32.768 kHz oscillator signal is also available at an output pin for used by other ICs. A separate frequency selectable square wave output (1 Hz is an option) is also available at another pin.

Using its 32.768 kHz time base, the DS3231 counts seconds, minutes, hours, date of the month, month, day of the week, and year, with leap year compensation valid to 2100. These values can be read from the DS3231's internal registers. Time is accurate to ±2 minutes/year.

The DS3231 also has an interrupt request output that can be controlled by its two time-of-day alarms. These alarms are programmable on a combination of seconds, minutes, hours, day of the week, and day of the month.

A Lithium battery can be used with the DS3231 to provide power fail backup.

The interface between the DS3231 and the AVR128DB48 in this design must use module I2C0.

This laboratory requires the you carefully read the DS3231's data sheet to understand the hardware and software operation of the device. You will also be extensively using the MSO-X 3012A oscilloscope and Saleae Logic Pro 16 Logic Analyzer to verify the DS3231's operation. This is another opportunity for you to further develop your skills in using these instruments.

The following is a list of steps for "bringing up" a new IC device in an embedded system prototype:
1. Carefully and completely read the data sheet.
2. Draw a schematic for the proper circuit connections and configuration.
3. Verify that the logic levels, loading, and timing of the interfaces are compatible.
4. Wire the circuit and, if possible, verify the stand alone operation of the IC.
5. Connect the IC to the microcontroller.
6. Write the IC's driver functions and place them in a separate file.
7. Write a test program to verify that the microcontroller can communicate with the IC.
8. Using an oscilloscope or logic analyzer, verify that all of the timing parameters are met.

Design Tasks

This laboratory focuses on designing and implementing a hardware interface of a DS3231 to an AVR128DB48 and the development of software drivers to support the DS3231's use by C code running on the AVR128DB48. Since, like the AVR128DB48, the DS3231 comes in a surface mount IC package, we need a breakout board for it so that we can do our development work. The breakout board selected is the Adafruit DS3231 Precision RTC Breakout.

Design Task 1: DS3231 and AVR128DB48 I2C Interface

Design the circuit for an I2C interface of the DS3231 (Adafruit breakout) to the AVR128DB48. Battery back up is provided using a CR1220 Lithium coin battery.

Draw, on a separate page, a schematic showing the DS3231's wiring. Use offpage connectors to show the interface to the AVR128DB48.

Perform computations to verify that the logic levels and currents of the interface are compatible. Use the Voltage and Current Interface Compatibility Checklist to verify voltage and current compatibility.

You must submit your schematic page for the DS3231 and logic level and current compatibility analysis as part of your prelab.

Design Task 2: "Single-Byte" Read and Write Functions for the DS3231

Laboratory Task 1 verifies that the DS3231's 32.768 kHz oscillator works. Now it is time to determine if the I2C interface works.

The DS3231 contains registers for time and alarms, a control register, and a status register. Each of these registers can be written and read.

Data transfers to and from the DS3231 may be classified as either "single-byte" transfers or "multi-byte burst" transfers. This task focuses on "single-byte" transfers. What is meant by a single-byte transfer is the transfer of a single data byte. To accomplish one "single-byte" data transfer to or from the DS3231 actually requires a bus transaction that consists of three bytes on the I2C interface. The first byte is the 7-bit slave address of the DS3231 (0x68) and the direction bit, the second byte is the address of the DS3231 register the data is to be transferred to or from, and the third byte is the actual data. In order for a transaction to be successful, the AVR128DB48's control registers must be properly configured so that the transfer data rate is compatible with the DS3231. For this laboratory you must configure the I2C0 module so data transfers occur at the highest speed possible.

You need a simple test program to determine whether the AVR128DB48 I2C is configured properly to write data to and read data from the DS3231. The simplest test is to write values to some of the alarm registers in the RTC and read them back.

When this test program is run in the laboratory, the I2C signals can be monitored and timing between these signals measured, using the Saleae Logic Pro 16. This allows you to verify if proper configuration and timing have been achieved.

You must write three basic DS3231 driver functions and a simple test program that uses them. Abbreviated headers for the three basic driver functions are:

```
//*************************************************************************
// Function             : void I2C_rtc_DS3231_config(void)
// Date and version     : 041024, version 1.0
// Target MCU           : AVR128 @ 4MHz
// Author               : Ken Short
// DESCRIPTION
// This function configures an AVR128DB48 operated at
// 4 MHz to communicate with the DS323
// SCL must be operated at the maximum possible frequency for
// the DS3231.
//*************************************************************************
```

```
//*************************************************************************
// Function: void write_RTC (uint8_t slave, unit8_t reg_RTC, uint8_t data_RTC)
//
// Target MCU           : AVR128DB48 @ 4MHz
// Target Hardware      ;
// Author               : Ken Short
// DESCRIPTION
// This function writes data to a register in the RTC. To accomplish this, it
// must first write the DS3231's slave address, then the register's pointer
// address, and finally the data.
//*************************************************************************

//*************************************************************************
// Function: uint8_t read_RTC (uint8_t slave, uint8_t reg_RTC)
// Target MCU           : AVR128DB48 @ 4MHz
// Author               : Ken Short
// DESCRIPTION
// This function reads data from a register in the RTC. To accomplish this, it
// must first write the DS3231's slave address, then its pointer address, and
// finally read the data.
//*************************************************************************
```

Create a project named `DS3231_write_read_test`. Write a program that calls the function `write_RTC` to write a value into one of the alarm registers and then calls the function `read_RTC` to read that same register and verify the I2C communications between the AVR128DB48 and the DS3231. When this test program is run on the AVR128DB48 in the laboratory you will use the Logic Pro 16 and breakpoints for debugging.

In the infinite while loop, you will need to delay for a period of time, say 1ms between writing and reading alarm registers.

You must submit a copy of your source file that contains the functions specified and the function main(), and your calculations to determine the I2C baud prescalar value.

Design Task 3: Multi-byte Burst Transfer Write and Read Functions for the DS3231

The DS3231 has a **burst mode** for transferring a block of data to consecutive registers in the clock. Burst mode increases the rate of data transfer since only the address of the first data byte in the block is transmitted to the DS3231. This effectively doubles the throughput. This mode is similar to a single-data-byte read or write, except that multiple data bytes are sent until the end of the burst.

The clock registers may be written or read in burst mode. When accessing the clock registers in burst mode, the address pointer wraps around after reaching 0x12. During a multi-byte access,

when the address pointer reaches the end of the register space (0x12), it wraps around to location 0x00. On an I2C START or address pointer incrementing to location 0x00, the current time is transferred to a second set of registers. The time information is read from these secondary (buffer) registers, while the clock may continue to run.

A simple test program for the multi-byte burst mode is to write values to the seven time and date registers (equivalent to setting the clock) and then read them back. After the initial read back, subsequent reads should give the current time for that reading.

Write two functions that allow a block write and a block read using the DS3231's burst mode. Abbreviated headers for these two functions are:

```
//***************************************************************************
// Function Name        : "block_write_RTC"
// void block_write_RTC (uint8_t slave, volatile uint8_t *array_ptr,
//                       uint8_t strt_addr, uint8_t count)
// Target MCU           : AVR128DB48 @ 4MHz
// Author               : Ken Short
// DESCRIPTION
// This function writes a block of data from an array to the DS3231. strt_addr
// is the starting address in the DS3231. count is the number of data bytes to
// be transferred and array_ptr is the address of the source array in the AVR128.
//***************************************************************************

//***************************************************************************
// Function Name        : "block_read_RTC"
// void block_read_RTC (uint8_t slave, volatile uint8_t *array_ptr,
//                       uint8_t strt_addr, uint8_t count)
// Target MCU           : AVR128DB48 @ 4MHz
// Author               : Ken Short
// DESCRIPTION
// This function reads a block of data from the DS3231 and transfers it to an
// array. strt_addr is the starting address in the DS3231. count is the number
// of data bytes to be transferred and array_ptr is the address of the
// destination array in the AVR128.
//***************************************************************************
```

Create a new workspace and project named `DS3231_block_write_read_test`. Write a program by that same name that writes a block of data to the RTC and then reads it back to test the I2C communications between the AVR128DB48 and the DS3231.

Also, add to that file the following arrays, which you must use in the verifications:

```
volatile uint8_t RTC_time_date_write[7];
volatile uint8_t RTC_time_date_read[7];
```

Write a program named `block_write_read_RTC_test` that configures the I2C for commu-

nication with the DS3231, calls `block_read_RTC` to transfer the contents of the DS3231 clock registers to the array `RTC_time_date_read`. The contents of this array is then copied to the array `RTC_time_date_write`. Next the function `block_write_RTC` is called.

By putting a breakpoint on the call to `block_write_RTC` the current setting for the clock can be changed by modifying the contents of the `RTC_time_date_write` array.

The program then executes a while loop that loops forever and each time through the loop calls the `block_read_RTC` function to read the first seven bytes from the DS3231 data registers to the `RTC_time_date_read` array. By putting a breakpoint on the call to `block_read_RTC`, you can verify the setting. Each subsequent time you click on Continue you will see the current time as kept by the DS3231. Place this function in your file `DS3231_RTC_drivers`.

When this test program is run on the AVR128DB48 in the laboratory you will use the Logic Pro 16 and breakpoints for debugging. Therefore, this project only needs one file, this file contains the function `main()` and the other DS3231 driver functions. In the infinite while loop in this file, you only need to call `block_read_RTC` and the delay for a period of time, say 10 ms.

There is no prelab for Task 3.

Design Task 4: Display of Time

Create a new workspace and project named `display_time`. Add copies of the functions from your code from Tasks 2 and 3 to the project `disply_time`. Also, add copies of the functions needed to operate the LCD.

Create whatever additional functions you need and modify the code in `main()` so that the DS3231 generates an interrupt every second using Alarm 1. Use the DS3231's $\overline{\text{INT}}$ output as the interrupt request source, do not use the 1 Hz output. On the AVR128DB48, use PC2 as the interrupt input. Each time this interrupt occurs, the program must display the time in 24 hour format on the first line of the LCD display. Use the format `hours:minutes:seconds`.

```
Time: 14:48:27
```

You will have to use a breakpoint to enter values into the array `RTC_time_date_write[]` once at the beginning of the execution of your program to set the time.

Leave the infinite loop in main empty.

There is no prelab for Task 4.

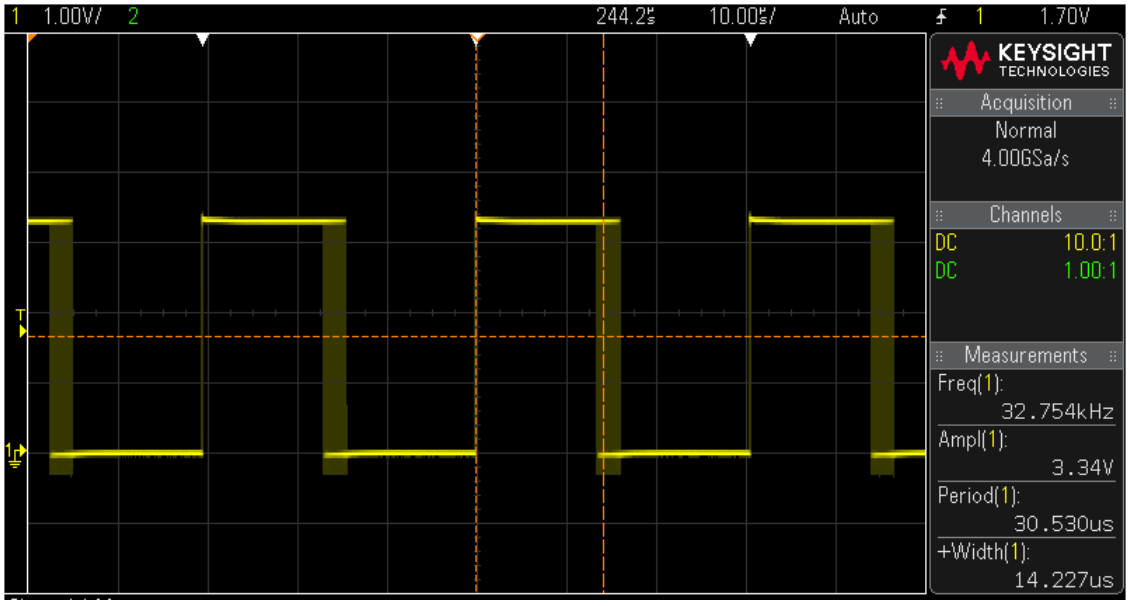
Laboratory Activity

Laboratory Task 1: DS3231 and AVR128DB48 I2C Interface

The objective of this task is to verify the basic electrical operation of the DS3231 with regard to its primary time base. Wire the basic DS3231 circuit, but do not connect it to the AVR128DB48. Also, do not place a battery in the battery holder.

Have a TA verify your wiring before applying power and proceeding.

Using the oscilloscope, verify the existence of the 32.768 kHz oscillator time base of the RTC. Measure the 32.768 kHz output of the DS3231. Use channel 1 of the oscilloscope. Turn OFF channel 2. Set the oscilloscope to measure and display the frequency. Do a PNG format screen capture to your thumb drive and print the image. See pages 277 to 278 of the Agilent *InfiniiVision 3000 X-Series Oscilloscopes User's Guide*.



If the 32.768kHz output is not present and correct, troubleshoot your circuit.

Demonstrate to a TA that your DS3231 32.768kHz oscillator is working and that you have captured the images.

Demonstrate to a TA your 32.768 kHz output and obtain that TA's signature.

Laboratory Task 2: "Single-Byte" Read and Write Functions for the DS3231

The objective of this laboratory task is to verify the logic and timing of “single-byte” transfers on the I2C bus.

Wire the connections of the DS3231 to the AVR128DB48 via I2C. Connect the Logic Pro 16 to the DS3231. Configure the Logic Pro 16 Protocol Analyzer for the I2C protocol. Display all of the DS3231’s I2C signals.

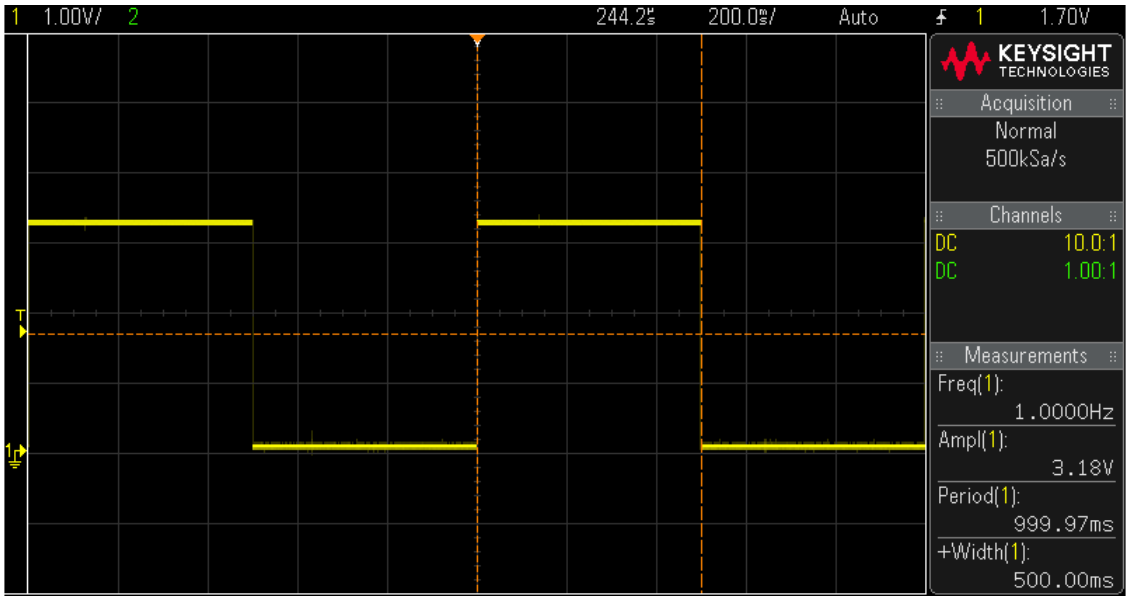
Load and make your `DS3231_write_read_test` project. Start the Logic Pro 16 and then run your program. The Logic Pro 16 should capture the waveforms.

Use a watch window to display the contents of `RTC_byte[]` to verify that the data was properly written to the DS3231 and read back and stored in `RTC_byte[]`.

Using Logic Pro 16 demonstrate to a TA that your writes to and reads from the DS3231 are properly occurring over the I2C bus and that the DS3231’s timing requirements are met.

You were not able to verify the 1Hz output in Task 1, do so now. Write a byte to the appropriate location in the DS3231 to enable its output SQW. What is the default output frequency at this pin?

Write a byte to the appropriate location in the DS3231 to enable its output SQW and set the ouput frequecy to 1 Hz. Measure the 1 Hz output at SQW of the DS3231 using channel 2 of the oscilloscope. Turn OFF channel 1. Set the oscilloscope to measure and display the frequency. Do a PNG format screen capture to your thumb drive and print the image.



If the 1 Hz output is not present and correct, troubleshoot and correct your circuit.

Obtain a TA’s signature.

Laboratory Task 3: Multi-byte Burst Transfer Write and Read Functions for the DS3231

The objective of this laboratory task is to verify the logic and timing of "multi-byte burst" transfer driver functions on the I2C bus.

Load and build your `DS3231_block_write_read_test` project. Slide the Lithium battery to the battery holder. The + side (shiny and wide) of the battery goes down. Use the oscilloscope to verify the output of the battery at the pins of the DS3231 before applying power to the system.

Set a breakpoint on the call the to `block_write_RTC()` that is outside of the infinite loop. Also, set a breakpoint on the call the to `block_read_RTC()` that is inside of the infinite loop. Put the arrays `RTC_time_date_write` and `RTC_time_date_read` into the Watch 1 window.

Run your program until it hits the breakpoint on `block_write_RTC()`. Set the appropriate time values into the locations of the `RTC_time_date_write` array. Start the Logic Pro 16 and then continue to run your program. The Logic Pro 16 should capture the waveforms. When the `block_read_RTC()` breakpoint is hit you should have the correct settings read back in `RTC_time_date_read` array. When you want to read the current time just hit go and the current time readings should be displayed in the `RTC_time_date_read` array.

Have the TA verify that your block write and read works and that you clock is keeping time. Obtain the TA's signature.

Laboratory Task 4: Display Time

Load and debug your program `display_time`.
Have the TA verify that your program properly displays the time in the required format. Obtain the TA's signature.

Leave the hardware that you have wired on the breadboard intact. This hardware may be used again in later laboratories