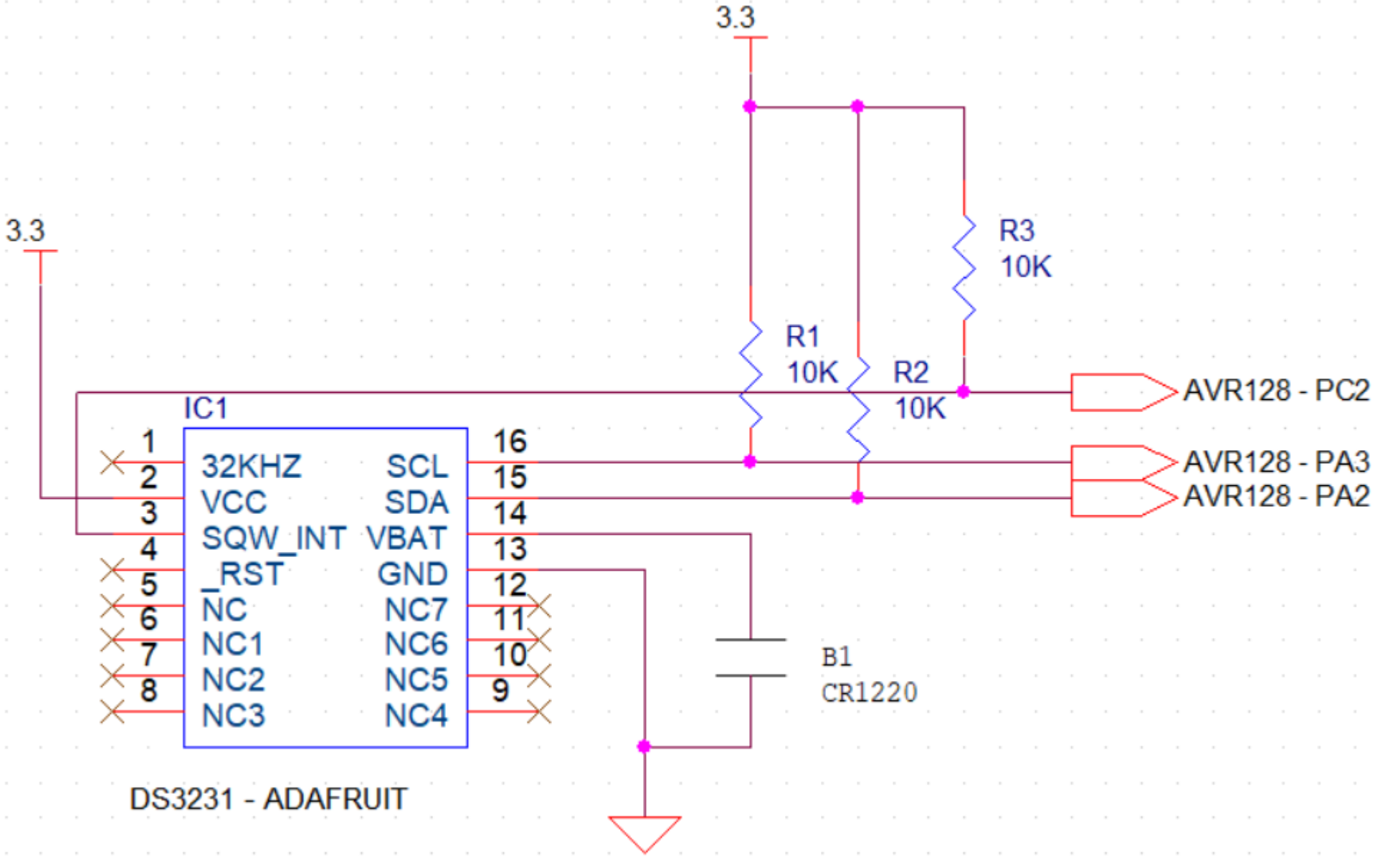
Laboratory 10: Serial Alarm Real-Time Clock

CHRISTOPHER NIELSEN + CHRISTOPHER SHAMAH
KENNETH SHORT
Bench No: 17
ID#114211318
ID#112229076
Lab Section L01


3.3
3.3
R3
10K
R1
10K
R2
10K
AVR128 - PC2
AVR128 - PA3
AVR128 - PA2
IC1
1
2
3
4
5
6
7
8
32KHZ
VCC
SQW_INT
_RST
NC
NC1
NC2
NC3
SCL
SDA
VBAT
GND
NC7
NC6
NC5
NC4
16
15
14
13
12
11
10
9
B1
CR1220
DS3231 - ADAFRUIT

```c
/*
 * display_time.c
 *
 * Created: 4/14/2024 4:09:25 PM
 * Author : MysticOwl
 */
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 4000000 //freq
#include <util/delay.h>
#include <stdio.h>
#include <string.h>
#include <stdint.h>

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



//Function prototypes
void TWI0_LM75_init(void);  //Initialize TWI0 module to talk to LM75

//Write a byte to the specified I2C slave. Parameters are slave address,
//address of register in slave to be written, and data to be written.
int TWI0_LM75_write(unsigned char saddr, unsigned char raddr, unsigned char data);
//void LM75_TWI0_init(void) ; //Initialize LM75
uint16_t TWI0_LM75_read(unsigned char saddr);



// Display buffer for DOG LCD using sprintf()
char dsp_buff1[17];
char dsp_buff2[17];
char dsp_buff3[17];

void lcd_spi_transmit_CMD (char cmd);  //macro for multiple write spi functions for a
  setup command
void lcd_spi_transmit_DATA (char cmd); //macro for multiple write spi functions for
  any data to be sent
void init_spi_lcd (void); //init spi0 settings for avr
void init_lcd_dog (void); //finish init commands for dog
void update_lcd_dog(void); //send buffer for line data

void init_spi_lcd (){

    PORTA.DIR |= PIN4_bm; /* Set MOSI pin direction to output */
    PORTA.DIR &= ~PIN5_bm; /* Set MISO pin direction to input */
    PORTA.DIR |= PIN6_bm; /* Set SCK pin direction to output */
    PORTA.DIR |= PIN7_bm; /* Set SS pin direction to output */
    PORTA.OUT |= PIN7_bm; /* Set SS pin direction to output */

    PORTC.DIR |= PIN0_bm; //Reg select output to the display memory
```

```
    PORTC.OUT &= ~PIN0_bm;

    SPI0.CTRLB |= (SPI_SSD_bm | 0x03 ); // mode 3 as per the dog waveforms

    SPI0.CTRLA = SPI_ENABLE_bm | SPI_MASTER_bm;

    init_lcd_dog();
}


void lcd_spi_transmit_CMD (char data){

    PORTA_OUT &= ~PIN7_bm; //Slave select ON
    PORTC.OUT &= ~PIN0_bm; // register select 0, command setting
    SPI0.DATA = data;

    while (!(SPI0.INTFLAGS & SPI_IF_bm))  /* waits until data is exchanged*/
    {
        asm volatile ("nop");
    }
    volatile uint8_t dummy;
    dummy = SPI0_DATA;


    PORTF_OUT = PIN7_bm; //Slave select OFF

}

void lcd_spi_transmit_DATA (char data){

    PORTA_OUT &= PIN7_bm; //Slave select ON
    PORTC.OUT = PIN0_bm;  // register select 1, data setting
    SPI0.DATA = data;

    while (!(SPI0.INTFLAGS & SPI_IF_bm))  /* waits until data is exchanged*/
    {
        asm volatile ("nop");
    }


    PORTF_OUT = PIN7_bm; //Slave select OFF

}



void init_lcd_dog(){

    //start_dly_40ms:
    _delay_ms(90);    //startup delay.

    //func_set1:
```

```
    lcd_spi_transmit_CMD(0x39);   // send function set #1 //tell for 3 lines and data
      interface at 8 bits
    _delay_us(30);  //delay for command to be processed


    //func_set2:
    lcd_spi_transmit_CMD(0x39); //send function set #2 // again??
    _delay_us(30);  //delay for command to be processed


    //bias_set:
    lcd_spi_transmit_CMD(0x1E); //set bias value.
    _delay_us(30);  //delay for command to be processed


    //power_ctrl:
    lcd_spi_transmit_CMD(0x55); //~ 0x50 nominal for 5V
    //~ 0x55 for 3.3V (delicate adjustment).
    _delay_us(30);  //delay for command to be processed


    //follower_ctrl:
    lcd_spi_transmit_CMD(0x6C); //follower mode on...
    _delay_ms(220); //delay for command to be processed SPECIAL CASE


    //contrast_set:
    lcd_spi_transmit_CMD(0x7F); //~ 77 for 5V, ~ 7F for 3.3V
    _delay_us(30);  //delay for command to be processed


    //display_on:
    lcd_spi_transmit_CMD(0x0c); //display on, cursor off, blink off
    _delay_us(30);  //delay for command to be processed


    //clr_display:
    lcd_spi_transmit_CMD(0x01); //clear display, cursor home
    _delay_us(420); //delay for command to be processed


    //entry_mode:
    lcd_spi_transmit_CMD(0x06); //clear display, cursor home
    _delay_us(30);  //delay for command to be processed


}



// Updates the LCD display lines 1, 2, and 3, using the
// contents of dsp_buff_1, dsp_buff_2, and dsp_buff_3, respectively.
```

```
void update_lcd_dog(void) {

    init_spi_lcd();     //init SPI port for LCD.



    // send line 1 to the LCD module.
    lcd_spi_transmit_CMD(0x80); //init DDRAM addr-ctr
    _delay_us(30);  //delay for command to be processed
    for (int i = 0; i < 16; i++) {
        lcd_spi_transmit_DATA(dsp_buff1[i]);
        _delay_us(30);  //delay for command to be processed
    }

    // send line 2 to the LCD module.
    lcd_spi_transmit_CMD(0x90); //init DDRAM addr-ctr
    _delay_us(30);  //delay for command to be processed
    for (int i = 0; i < 16; i++) {
        lcd_spi_transmit_DATA(dsp_buff2[i]);
        _delay_us(30);  //delay for command to be processed
    }

    // send line 3 to the LCD module.
    lcd_spi_transmit_CMD(0xA0); //init DDRAM addr-ctr
    _delay_us(30);  //delay for command to be processed
    for (int i = 0; i < 16; i++) {
        lcd_spi_transmit_DATA(dsp_buff3[i]);
        _delay_us(30);  //delay for command to be processed
    }
}



//***************************************************************************
// 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.
//***************************************************************************

void I2C_rtc_DS3231_config(void){

    TWI0.MBAUD = 0x01;  //Want 400kHz, but to get it BAUD value would be negative.
      4MHz main clock -> ~400KHz I2C clock
    TWI0.MCTRLA = 0x01; //Enable TWI master bit0
    //Smart mode enable SMEN is bit1, it is 0, so smart mode not enabled.
    //Since SMEN = 0, MCMD field in MCTRLB must be written for each byte
```

```
    //received by master to create an acknowledge action followed by an operation.
    TWI0.DBGCTRL = 0x01;
    TWI0.MSTATUS = 0x01; //Force bus state to idle



}

//***************************************************************************
// 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.
//***************************************************************************
void block_write_RTC(uint8_t slave, volatile uint8_t *array_ptr, uint8_t strt_addr,
  uint8_t count) {
    while((TWI0.MSTATUS & 0x03) != 0x01); // wait until idle

        TWI0.MADDR = slave << 1;         // send base address for write
        while((TWI0.MSTATUS & 0x40) == 0); // WIF flag, wait until saddr sent

        TWI0.MDATA = (strt_addr);    // send memory address
        while((TWI0.MSTATUS & 0x40) == 0); // WIF flag, wait until raddr sent

for (uint8_t i = 0; i < count; i++) {
        TWI0.MDATA = array_ptr[i];       // send data
        while((TWI0.MSTATUS & 0x40) == 0); // WIF flag, wait until data sent
    }

    TWI0.MCTRLB |= 0x03;               // issue a stop
    return;
}


//***************************************************************************
// 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.
//***************************************************************************
```

```
void block_read_RTC(uint8_t slave, volatile uint8_t *array_ptr, uint8_t strt_addr,
  uint8_t count) {
    // Store the data in this: volatile uint8_t RTC_time_date_read[7];

    while((TWI0.MSTATUS & 0x03) != 0x01); // wait until idle


        TWI0.MADDR = slave << 1;         /* send address for write */ // 0 VERSION
        while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until saddr sent */

        //The next write clears the WIF flag
        TWI0.MDATA = strt_addr;              /* send memory address    WRITE COMMAND
          */
        while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until raddr sent */

        TWI0.MADDR = (slave << 1) | (0b00000001) ;         /* send address for write
          */ // 1 VERSION

    for (uint8_t i = 0; i < count-1; i++) {

        while((TWI0.MSTATUS & 0x80) == 0); // RIF flag, wait until byte is received
        array_ptr[i] = TWI0.MDATA; // Store the received byte in the destination array

        TWI0.MCTRLB = 0x02; // MCMD - issue ack followed by a byte read operation


    }


    while((TWI0.MSTATUS & 0x80) == 0); // RIF flag, wait until byte is received
        array_ptr[count-1] = TWI0.MDATA; // Store the received byte in the destination
           array

        TWI0.MCTRLB = TWI_ACKACT_NACK_gc | TWI_MCMD_STOP_gc;    //MCMD issue nack
          followed by a stop


    return; // read data from received data buffer
}

void display_new_time(){


    cli();
    // Variables for the read function
    uint8_t slave_address = 0x68; // Slave address of the DS3231 device
    uint8_t start_address = 0x00; // Starting address in the DS3231 to read from
    uint8_t read_count = 3; // Number of bytes to read
    uint8_t seconds;
    uint8_t minutes;
    uint8_t hours;
    uint8_t tens_seconds;
```

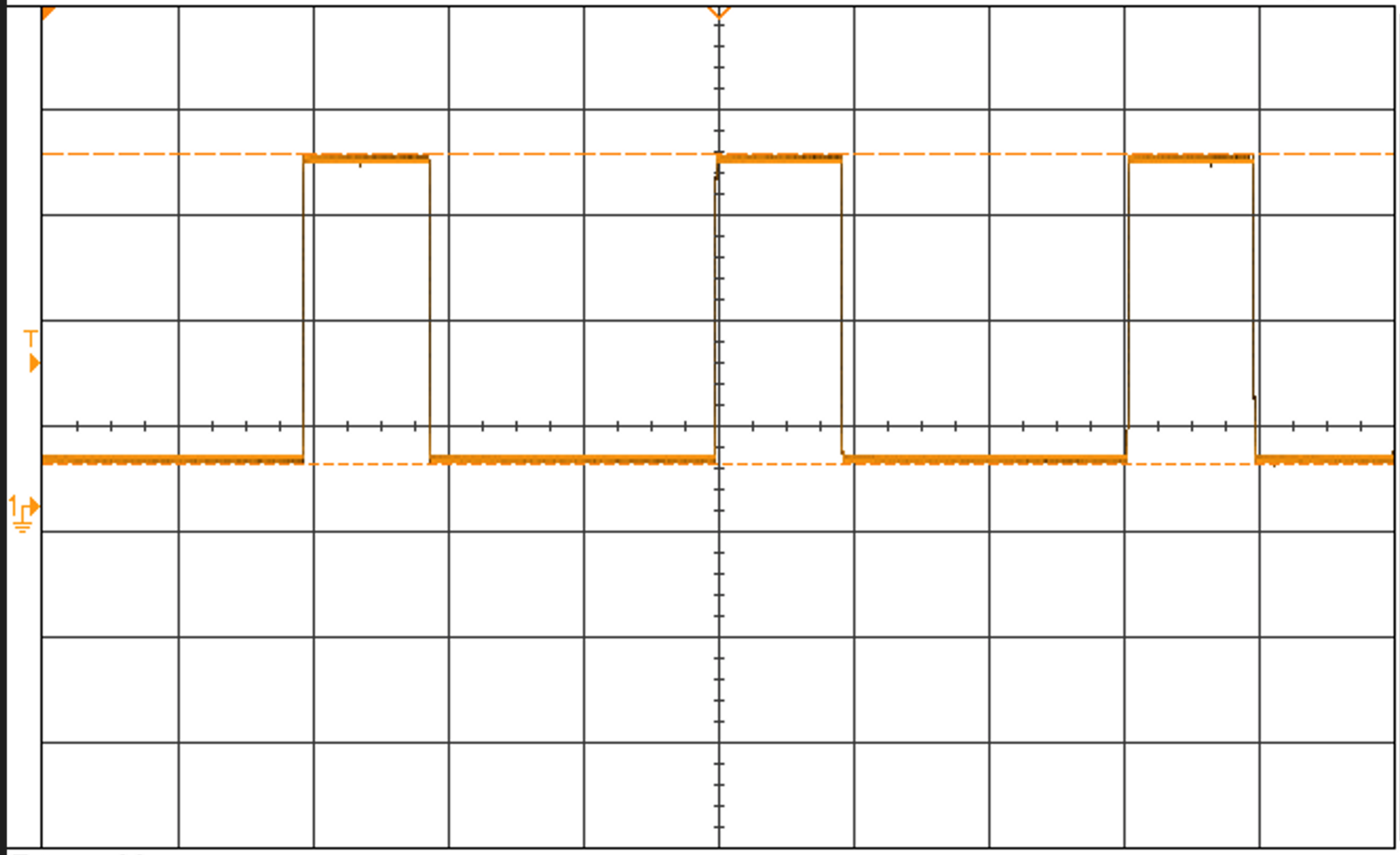
```
    uint8_t tens_minutes;
    uint8_t tens_hours;

    //rest of code
    block_read_RTC(slave_address, RTC_time_date_read, start_address, read_count);


    tens_seconds = (RTC_time_date_read[0] & 0x70) >> 4 ;
    tens_minutes = (RTC_time_date_read[1] & 0x70) >> 4 ; // take 10s place of all 3
      numbers
    seconds = RTC_time_date_read[0] & 0x0F ;
    minutes = RTC_time_date_read[1] & 0x0F ; // mask out data we dont need and its
      already oriented as we need
    hours   = RTC_time_date_read[2] & 0x0F ;


    tens_hours   = (RTC_time_date_read[2] & 0x70) >> 4 ;
    if(tens_hours & 0x04){
        if(tens_hours & 0x02){
            tens_hours = tens_hours & 0x01;
            sprintf(dsp_buff1,  "Time: %u%u:%u%u:%u%uPM",
              tens_hours,hours,tens_minutes,minutes,tens_seconds,seconds );
        }else{
            tens_hours = tens_hours & 0x01;
            sprintf(dsp_buff1,  "Time: %u%u:%u%u:%u%uAM",
              tens_hours,hours,tens_minutes,minutes,tens_seconds,seconds );
        }
    }else{
        tens_hours = tens_hours & 0x03;
        sprintf(dsp_buff1,  "Time: %u%u:%u%u:%u%u",
          tens_hours,hours,tens_minutes,minutes,tens_seconds,seconds );
    }



    update_lcd_dog(); // update display with the new time

    //  Pin interrupt flag n is cleared by writing a 1 to it
     PORTC.INTFLAGS |= 0x04; // falling edge interrupt clear.



}

ISR (PORTC_PORT_vect){

    display_new_time();

}
```

```
//*****************************************************************************
// 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.
//*****************************************************************************

void write_RTC (uint8_t slave, uint8_t reg_RTC, uint8_t data_RTC){

        while((TWI0.MSTATUS & 0x03) != 0x01) ; /* wait until idle */

        TWI0.MADDR = slave << 1;         /* send address for write */
        while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until saddr sent */

        //The next write clears the WIF flag
        TWI0.MDATA = reg_RTC;              /* send memory address */
        while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until raddr sent */

        //The next write clears the WIF flag
        TWI0.MDATA = data_RTC;               /* send data */
        while((TWI0.MSTATUS & 0x40) == 0); /* WIF flag, wait until data sent */

        //The next write clears the WIF flag
        TWI0.MCTRLB |= 0x03;          /* issue a stop */

        return;

}



int main(void)
{
    uint8_t DS3231addr = 0x68;
//  uint8_t REGaddr = 0x07; //alarm seconds reg ADDRESS
//  uint8_t configdata = 0x80; //set alarm to once a second

    // Variables for the read function
    uint8_t slave_address = 0x68; // Slave address of the DS3231 device
    uint8_t start_address = 0x00; // Starting address in the DS3231 to read from
    uint8_t read_count = 7; // Number of bytes to read

    // Variables for the write function
    uint8_t write_start_address = 0x00; // Starting address in the DS3231 to write to
```

```
uint8_t REGaddr = 0x0E; //CONTROL reg ADDRESS
uint8_t configdata = 0x00;
uint8_t statusAddr = 0x0F;
uint8_t statusData = 0x08;

    init_spi_lcd();

    I2C_rtc_DS3231_config();


    RTC_time_date_write[0] = 0x00; //0 seconds
    RTC_time_date_write[1] = 0x00; //0 minutes
    RTC_time_date_write[2] = 0x12; //12 hours
    RTC_time_date_write[3] = 0x18; //date
    RTC_time_date_write[4] = 0x04; //month
    RTC_time_date_write[5] = 0x24; //year



    uint8_t write_count = 6; // Number of bytes to write

    block_write_RTC(slave_address, RTC_time_date_write, write_start_address,
      write_count); //enter the time for it to be set to initially ( January 1st 5AD
      12pm)

   //setup PC2 as a interrupt low alarm from DS3231
   PORTC.DIR &= 0b11111011; //only PC2 is brought low, making it an input.
   PORTC.PIN2CTRL = 0x03; // falling edge interrupt




    //config DS3231 to do alarms every second
    write_RTC(DS3231addr, statusAddr, statusData);
    write_RTC(DS3231addr, REGaddr, configdata);




    /* Replace with your application code */
    while (1)
    {
    //display_new_time();
sei();
    asm volatile("nop");

    }
    return (0);
}
```

DSO-X 3012A, MY56040980: Fri Apr 19 04:04:53 2024

1 1.00V/ 2 450.0ms 10.00ks/ Stop 1 1.35V



KEYSIGHT TECHNOLOGIES

Acquisition

Normal

4.00GSa/s

Channels

DC 10.0:1

DC 1.00:1

Measurements

Freq(1):

32.769kHz

Pk-Pk(1):

2.93V

Trigger Menu

Trigger Type Edge

Source 1

Slope