EEE3096S: Practical 2

Daniel Jones (JNSDAN013)

Mishay Naidoo (NDXMIS011)

1.

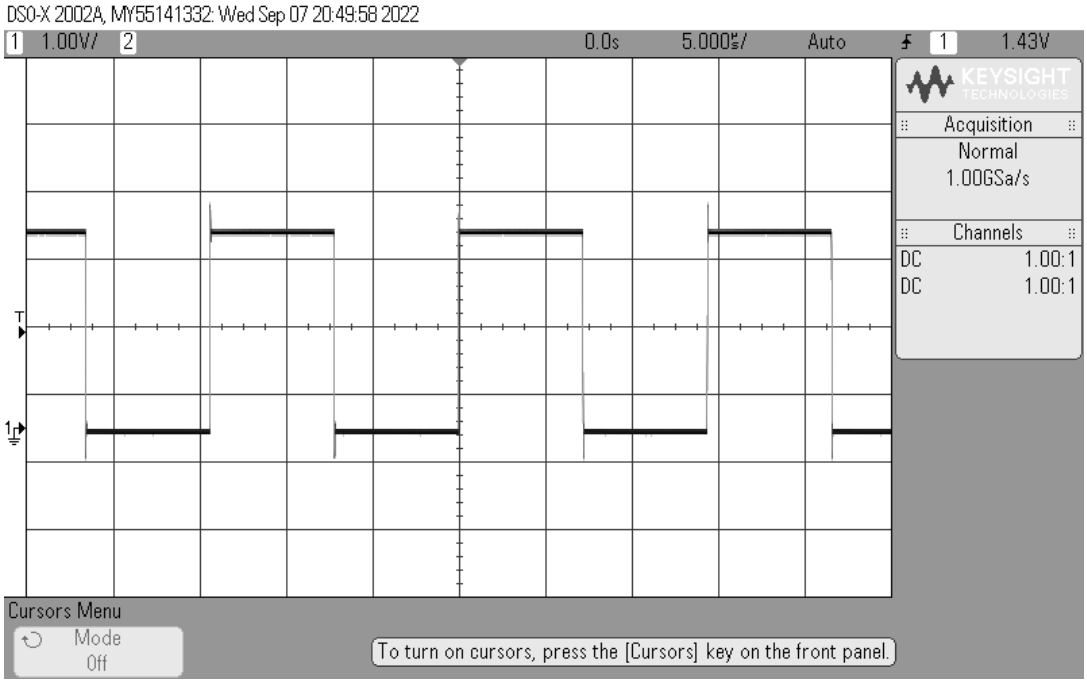

Figure 1: Oscilloscope Output for Task 1

2.

To create the delay function, nested for-loops were used with the outer loop corresponding to a delay of 1 second and the inner for-loop acting as a multiplier. When attempting to achieve a delay of 60 seconds, the delay only lasted 54 seconds. This can be attributed to how the STM32 runs for-loops. With some minor adjustments, this value was improved to 59.73 seconds. Thus our delay function has an accuracy of approximately $\pm 0.27\ seconds$ which is fine for long delays, but will not be acceptable if an accurate small delay is needed.

3.

4.

```
/* USER CODE BEGIN Header */
/**
**********************************
Info:        STM32 I2C with DS3231 HAL
Author:             Mishay Naidoo, Daniel Jones
**********************************
In this practical you will learn to use I2C on the STM32 using the HAL. Here, we
will
be interfacing with a DS3231 RTC. We also create functions to convert the data
between Binary
Coded Decimal (BCD) and decimal.

Code is also provided to send data from the STM32 to other devices using UART
protocol
by using HAL. You will need Putty or a Python script to read from the serial port
on your PC.

UART Connections are as follows: red->5V black->GND white(TX)->PA2
green(RX;unused)->PA3.
Open device manager and go to Ports. Plug in the USB connector with the STM
powered on. Check the port number (COMx).
Open up Putty and create a new Serial session on that COMx with baud rate of 9600.

https://www.youtube.com/watch?v=EEsI9MxndbU&list=PLfIJKC1ud8ghc4eFhI84z_3p3Ap2MCMV
-&index=4

RTC Connections: (+)->5V (-)->GND D->PB7 (I2C1_SDA) C->PB6 (I2C1_SCL)
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stdio.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
typedef struct {
	uint8_t seconds;
	uint8_t minutes;
	uint8_t hour;
	uint8_t dayofweek;
	uint8_t dayofmonth;
	uint8_t month;
	uint8_t year;
} TIME;

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
//TO DO:
```

```c
//TASK 2
//Give DELAY1 and DELAY2 sensible values
#define DELAY1 21700
#define DELAY2 64

//TO DO:
//TASK 4
//Define the RTC slave address
#define DS3231_ADDRESS 0xD0

#define EPOCH_2022 1640988000
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

UART_HandleTypeDef huart2;
DMA_HandleTypeDef hdma_usart2_tx;

/* USER CODE BEGIN PV */
char buffer[14];
uint8_t data [] = "Hello from STM32!\r\n";
TIME time;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
/* USER CODE BEGIN PFP */
void HAL_UART_TxCpltCllback(UART_HandleTypeDef *huart);
void pause_sec(float x);

uint8_t decToBcd(int val);
int bcdToDec(uint8_t val);
void setTime (uint8_t sec, uint8_t min, uint8_t hour, uint8_t dow, uint8_t dom,
uint8_t month, uint8_t year);
void getTime (void);
int epochFromTime(TIME time);

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void){
```

```c
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  //SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_DMA_Init();
  MX_USART2_UART_Init();

  /* USER CODE BEGIN 2 */


  //TO DO
  //TASK 6
  //YOUR CODE HERE
  //setTime(00, 20, 16, 3, 7, 9, 22);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
      //TO DO:
      //TASK 1
      //First run this with nothing else in the loop and scope pin PC8 on an
oscilloscope
      HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_8);

      //TO DO:
      //TASK 6

      //sprintf(buffer, "%s \r\n", "hello");


      getTime();
      sprintf (buffer, "%02d:%02d:%02d\r\n\r\n", time.hour, time.minutes,
time.seconds);
      HAL_UART_Transmit(&huart2, buffer, sizeof(buffer), 1000);
```

```
      sprintf (buffer, "%02d-%02d-20%02d\r\n\r\n", time.dayofmonth, time.month,
time.year);
      HAL_UART_Transmit(&huart2, buffer, sizeof(buffer), 1000);


      sprintf (buffer, "%d\r\n\r\n", epochFromTime(time));
      HAL_UART_Transmit(&huart2, buffer, sizeof(buffer), 1000);



      //This creates a string "55555555555555" with a pointer called buffer

      //Transmit data via UART
      //Blocking! fine for small buffers
      //HAL_UART_Transmit(&huart2, buffer, sizeof(buffer), 1000);



      //YOUR CODE HERE
      pause_sec(0);


    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL12;
  RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
```

```c
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_I2C1;
  PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_HSI;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x2000090E;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */
```

```c
  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 9600;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel4_5_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel4_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel4_5_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
```

```c
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, LD4_Pin|LD3_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD4_Pin LD3_Pin */
  GPIO_InitStruct.Pin = LD4_Pin|LD3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
void pause_sec(float x)
{
	/* Delay program execution for x seconds */
	//TO DO:
	//TASK 2
	//Make sure you've defined DELAY1 and DELAY2 in the private define section

	//YOUR CODE HERE
	for(float i = 1; i < DELAY1; i++){
		for(float j = 1; j < DELAY2*x; j++){}
	}
}

uint8_t decToBcd(int val)
{
    /* Convert normal decimal numbers to binary coded decimal*/
	//TO DO:
	//TASK 3

	//YOUR CODE HERE
	return (uint8_t)((val/10 * 16) + (val%10));
}

int bcdToDec(uint8_t val)
{
    /* Convert binary coded decimal to normal decimal numbers */
	//TO DO:
	//TASK 3
	//Complete the BCD to decimal function
```

```
      //YOUR CODE HERE
      //return (int)((val/16 *10) + (val%16));
      return ((val>>4)*10+(val&0x0F));

}

void setTime (uint8_t sec, uint8_t min, uint8_t hour, uint8_t dow, uint8_t dom,
uint8_t month, uint8_t year)
{
    /* Write the time to the RTC using I2C */
      //TO DO:
      //TASK 4

      uint8_t set_time[7];

      //YOUR CODE HERE

      //fill in the address of the RTC, the address of the first register to
write and the size of each register
      //The function and RTC supports multiwrite. That means we can give the
function a buffer and first address
      //and it will write 1 byte of data, increment the register address, write
another byte and so on

      set_time[0] = decToBcd(sec);
      set_time[1] = decToBcd(min);
      set_time[2] = decToBcd(hour);
      set_time[3] = decToBcd(dow);
      set_time[4] = decToBcd(dom);
      set_time[5] = decToBcd(month);
      set_time[6] = decToBcd(year);


      HAL_I2C_Mem_Write(&hi2c1, DS3231_ADDRESS, 0x00, 1, set_time, 7, 1000);
      //HAL_I2C_Mem_Write(&hi2c1, 0b1101000, FIRST_REG, REG_SIZE, set_time, 7,
1000);

}

void getTime (void)
{
    /* Get the time from the RTC using I2C */
      //TO DO:
      //TASK 4
      //Update the global TIME time structure

      uint8_t get_time[7];

      //fill in the address of the RTC, the address of the first register to
write anmd the size of each register
      //The function and RTC supports multiread. That means we can give the
function a buffer and first address
      //and it will read 1 byte of data, increment the register address, write
another byte and so on
      HAL_I2C_Mem_Read(&hi2c1, DS3231_ADDRESS, 0x00, 1, get_time, 7, 1000);


      //YOUR CODE HERE
```

```
        time.seconds = bcdToDec(get_time[0]);
        time.minutes = bcdToDec(get_time[1]);
        time.hour = bcdToDec(get_time[2]);
        time.dayofweek = bcdToDec(get_time[3]);
        time.dayofmonth = bcdToDec(get_time[4]);
        time.month = bcdToDec(get_time[5]);
        time.year = bcdToDec(get_time[6]);
}

int epochFromTime(TIME time){
    /* Convert time to UNIX epoch time */
        //TO DO:
        //TASK 5
        //You have been given the epoch time for Saturday, January 1, 2022 12:00:00
AM GMT+02:00
        //It is define above as EPOCH_2022. You can work from that and ignore the
effects of leap years/seconds

        //YOUR CODE HERE
        int months = time.month;
        int day = time.dayofmonth;
        int EPOCH = EPOCH_2022;
        switch(months){

        case 2:

                EPOCH += 31*24*60*60;
        break;

        case 3:

                EPOCH += 31 * 24 * 60 * 60 + 28 * 24 * 60 *60;

        break;

        case 4:

                EPOCH += 31 * 24 * 60 * 60 + 28 * 24 * 60 * 60 + 31*24*60*60;
        break;

        case 5:

                EPOCH += 31 * 24 * 60 * 60 + 28 * 24 * 60 * 60 + 31*24*60*60 +
30*86400;
        break;

        case 6:

                EPOCH += 31 * 24 * 60 * 60 + 28 * 24 * 60 * 60 + 31*24*60*60 +
30*86400 + 31*86400;
        break;

        case 7:
                EPOCH += 31 * 24 * 60 * 60 + 28 * 24 * 60 * 60 + 31*24*60*60 +
30*86400 + 31*86400 + 30*86400;
        break;

        case 8:
```

```
			EPOCH += 31 * 24 * 60 * 60 + 28 * 24 * 60 * 60 + 31*24*60*60 +
30*86400 + 31*86400 + 30*86400 + 31*86400;
		break;

		case 9:
			EPOCH += 31 * 24 * 60 * 60 + 28 * 24 * 60 * 60 + 31*24*60*60 +
30*86400 + 31*86400 + 30*86400 + 31*86400 + 31*86400;
		break;

		case 10:
			EPOCH += 31 * 24 * 60 * 60 + 28 * 24 * 60 * 60 + 31*24*60*60 +
30*86400 + 31*86400 + 30*86400 + 31*86400 + 31*86400 + 30*86400;
		break;

		case 11:
			EPOCH += 31 * 24 * 60 * 60 + 28 * 24 * 60 * 60 + 31*24*60*60 +
30*86400 + 31*86400 + 30*86400 + 31*86400 + 31*86400 + 30*86400 + 31*86400;
		break;

		case 12:
			EPOCH += 31 * 24 * 60 * 60 + 28 * 24 * 60 * 60 + 31*24*60*60 +
30*86400 + 31*86400 + 30*86400 + 31*86400 + 31*86400 + 30*86400 + 31*86400 +
30*86400;
		break;



		default:
			day = day;
		}

		EPOCH += (day-1)*24*60*60;

		return EPOCH;
}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
```

```
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```