PetCare

Arvind Balaji Narayan
Gopik Anand

Devpost: PetCare

Final Project, Fall 2021
ESE519/IPD519: Real-Time and Embedded Systems
University of Pennsylvania

Table of Contents

Abstract **3**

Motivation **3**

Goals **4**

Milestone 1 4

Final Demo 4

Methodology **5**

Results **8**

Conclusion **9**

References **10**

Appendix A **11**

Last edited on 2020/08/02 18:18 EDT

Abstract

PetCare is a completely automated pet food and water dispenser that is built upon the Arduino ATmega328P. It attempts to compensate for human inefficiencies in feeding their loved ones owing to their busy work schedules.

The dispenser comes as a compact package that features a Real-time clock module to schedule meal-time intervals, a consolidated setup consisting of a submersible water pump to pump out water, a relay that controls it and a water level sensor to indicate whether a refill is required. The food dispenser is equipped with a top-grade servo motor coupled with a shaft that acts as the lid to dispense food. The dispenser also offers manual dispensing capabilities via the Blynk IoT setup to allow users to feed their pets at the comfort of their smartphones.

PetCare was able to satisfy all the necessary expectations with respect to both the dispensing process and the two control modes, thus working in a clean, organized and coherent fashion.

Motivation

We've always loved pets. So much so that we're the type of people that can't resist going "OMG! That doggo is so cute. 🐶" or to refrain ourselves from petting them. It's also obvious that we might not always get the love back. However, when we bring up the idea of having a cat or dog for ourselves to our parents, it is always immediately shot down. The main reason most parents would have was, "You'll just play with it and forget the fact it has to be fed and kept clean. We'll just end up cleaning up the mess after you two". Kind of a harsh thing to say to a child, but then if you think about it, it's a fair assessment of what could possibly happen.

Now that we're obviously older and more mature (hopefully?!!) and we think back, our parents' argument actually makes even more sense. We're always dealing with a horribly tight class schedule with assignments, lectures and other stuff that we usually go to sleep at ungodly hours in the day only to realize that we won't be getting much sleep since we forgot to have dinner. If we can't feed ourselves at the right time, how will we remember to feed a pet? The situation becomes even more dire if we leave home for a couple of days and set up camp in a lab to finish homework (Happened a couple of times this semester alone 🥲) without anyone to watch over the pets. Considering all these scenarios, it would be great if we had a fully automatic Pet Food and Water Dispenser. It

can feed and provide water to our pets at regular intervals or whenever they feel hungry or thirsty. The easy way would be to just get one from Amazon or Walmart, set it up and call it a day. However, our ego of being graduate students at an IVY League school studying embedded systems got the better of us, and we decided to make it ourselves.

Goals

A. Milestone 1

• We had broken down our entire project work into multiple short-term goals that we managed to achieve over the course of the past few weeks. By Milestone 1, we were able to achieve the following results listed below:

1) Interface RTC and all the other components individually with the Arduino
2) The food and Water dispensing parts worked perfectly as individual systems
3) Completed the Blynk IoT setup through which the user can communicate with the dispenser and control the dispensing process.

• We achieved this in a step by step process by Milestone 1 (12/03/12) as given below:

a) Parts Selection & Power Management (10th - 15th Nov)
b) Water Dispenser Setup (14th - 20th Nov)
c) Food Dispenser Setup (20th - 27th Nov)
d) Complete Blynk IoT Setup (28th Nov - 3rd Dec)

B. Final Demo

After Milestone 1, we integrated both the food and water dispensing subsystems into one complete setup. At this stage, the device worked perfectly and satisfied all of its initial objectives of dispensing food and water at pre-scheduled intervals throughout the day.

We checked this out by setting the time for dispensing as 10 am in the morning and 3 pm in the afternoon, and the dispensing process started exactly at the scheduled times. We also verified that the water dispensing process (that gets initiated when the pet comes very close to the dispenser) works as expected. The safety module keeps track of the volume of water remaining in the storage container and alerts us suitably for refills.

Last edited on 2020/08/02 18:18 EDT

We were also able to manually access the dispenser and initiate the dispensing process (in case we wanted to initiate it at a particular time) using the Blynk IoT application setup.

In conclusion, we were able to fulfill all the objectives that we set forth to achieve as a team at the start of the project.

Methodology

A) Dispensing Water

Overview:

The basic functioning of the Water Dispenser module involves dispensing water for the pet at regular intervals throughout the day. However, it is practically impossible for us to predict when our pet would feel thirsty. Therefore, we provide the added functionality of dispensing water whenever the pet stands in front of the dispenser. In both modes of operation, we make sure that measured amounts of water are dispensed every time.

Algorithm:

• The process of dispensing water can be categorized into two modes. The **first** mode dispenses water along with food at regular pre-scheduled meal intervals for the day. To achieve this, we need time data according to which the dispenser would work, and we account for this by using the **Real-Time Clock (RTC) - DS1307.**

• We interface the RTC module to the Arduino ATmega328P so that we can get timely data from the RTC at which the water and food have to be dispensed. Apart from connecting the 5V and GND pins, the other connections are to the SDA and the SCL pins of the Arduino because we follow I^2C Interfacing. The appropriate libraries are used while we program the function to communicate with the RTC and measure time. In this manner, we can measure real-time with the RTC and can hence program the Arduino to start the food dispensing process at predetermined time slots. We also add in a buzzer that sounds for 10s to alert our pet that its meal is ready. We can also train our pets to understand that if the buzzer sounds, it's mealtime. 😋. So, for example, we could set the system once at the start and then it works in such a way that it dispenses food and water at 9 am, 2 pm and 8 pm of a day, every day until we decide to change this schedule.

• The second mode is to dispense water as and when the pet feels thirsty and stands in front of the pet food dispenser. This accounts for the fact that we can never predict when our pet would feel thirsty. We use an ultrasonic sensor that is connected to the ICP1 pin

(PB0) of the Arduino to measure the width of the output pulse of the sensor when an object is detected. The sensor is programmed in such a way that when the pet stands in front of it, the distance is measured. If this distance is less than the particular threshold that we decide upon, the process of dispensing water from the storage starts and water is poured into the bowl kept below for a prefixed amount of time so that our pet can drink from it. The time limit ensures that only a fixed volume of water is dispensed and no water is wasted.

• For both modes of operation, we need a way to physically take water from the storage and dispense it into the bowl. We will be using **FIT0563** - Mini Immersible Water Pump for our application to pump out water from the storage container and dispense it onto the bowl. We have used a power supply of 16V to power the pump and ensure that the pump pumps out water with sufficient force onto the water bowl.

• Since the pump starts operation whenever power is supplied to it, we need a way to control the power supply to the pump and thereby control pump operation. The pump is connected to and controlled by the Arduino via the HiLetgo 5V One Channel **Relay Module** (B00LW15A4W). We'll be using the normally open mode of operation of the relay. We can program the Arduino so that the pump is turned on whenever the Arduino signals it to.

• PVC Vinyl Tubes are connected to the outlet of the pump, and the entire structure is submerged inside water. The inlet draws water from the storage container, and the outlet discharges it into the bowl from which the pet can drink water.

• Finally, there is also the possibility that the water stored in the storage is used up and needs a refill. To ensure that we do not forget and water is refilled correctly, we make use of the (28090) Mini **Liquid Level Sensor,** which measures the depth of the water present in the storage and outputs a corresponding voltage which can be measured by the Arduino. The VCC and GND pins of the sensor are connected to the 5V and GND pins of the Arduino. The S pin is the analog output pin that is connected to the ADC pin (A0) of the Arduino.

• Using the lookup table where we get the water level in the form of their corresponding voltages, we can get an idea of the water level remaining, and a red LED glows to remind the user to refill water at the earliest.

Last edited on 2020/08/02 18:18 EDT

B) Dispensing Food

Overview:

The basic functioning of the Food Dispenser module involves dispensing food into the bowl at regular predetermined intervals of time throughout the day. We can load the food container with food for a couple of days or for a week and then use a setup consisting of a Servo Motor interfaced with the Arduino to dispense appropriate quantities of food from the storage box into the bowl.

Algorithm:

• We will be interfacing the Food Dispenser along with the Mode 1 operation of the Water Dispenser, where food and water will be dispensed at pre-scheduled intervals of the day. The timings are set using the RTC that is interfaced with the Arduino, and the pet can be alerted when the meal is ready using the buzzer, as discussed earlier.

• The container in which food is stored is positioned above the output shaft that has an opening in it. The positioning has been done in such a way that the shaft position before dispensing is kept in a manner that blocks the food from falling out. The **SG90 Servo Motor** is attached to the shaft, and the angle it needs to rotate the shaft b,y is tuned appropriately. The GND and 5V ends are suitably connected, and the 3rd pin is connected to one of the PWM pins of the Arduino.

• We program the Arduino to use PWM signals to rotate the Servo Motor and thereby move the shaft such that food is dispensed. Since we require a fixed pre-measured quantity of food to be dispensed (i.e. neither deficit nor surplus), we'll be giving a small delay after the servo motor rotates the shaft so that the correct amount of food is dispensed into the bowl.

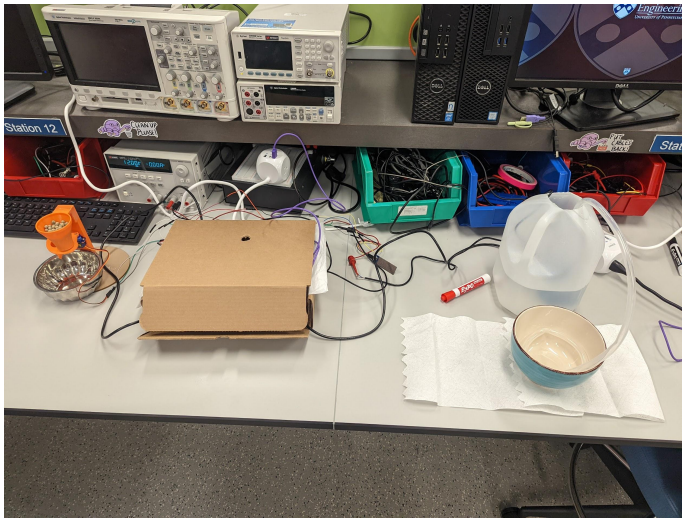

Figure 1: Image of PetCare

Last edited on 2020/08/02 18:18 EDT

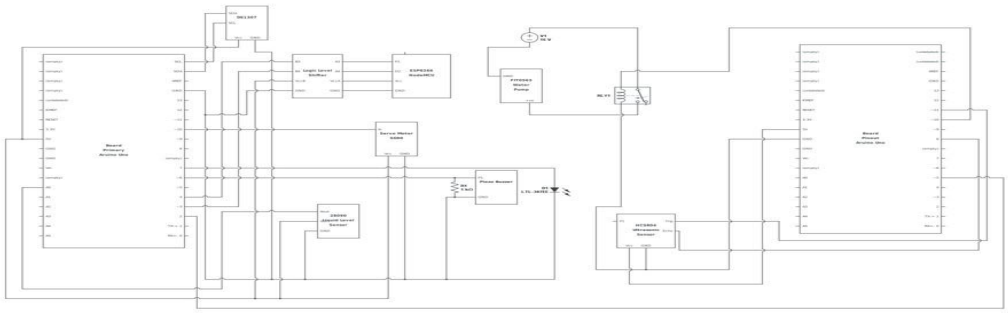

Figure 2: Circuit Diagram of PetCare

CircuitLab Implementation Diagram Link:
https://drive.google.com/file/d/1kjLfzhiDyZn90uKRbcLF9bw9m9MThRIp/view?usp=sharing

---

Results

The final design of the system as shown in the video demonstration, has two subsystems integrated together to work as one. We had our food dispenser 3D printed with the help and guidance of Kim. The food was dispensed onto a bowl placed underneath the opening of the dispenser. For the water dispenser, we used two 1 gallon bottles where one was used as the storage container while the other one was cut into the shape of a bowl from which our pet could sip water, and if it felt thirsty, it always has the option to stand in front of the dispenser (train them to learn this), and water would get dispensed.
As per the methodology discussed in the previous section, PetCare did a splendid job in dispensing water and food accurately at the time it was scheduled to start. We were able to dispense water not only during meal times but also at situations when the pet felt thirsty and stood in front of the dispenser for water.

Along with this, the second control mode which involves controlling the dispensing process using the Blynk IoT application also worked properly by dispensing food and water whenever we initiated the process with just a few touches on our smartphone. We had two button controls so that the user could choose what they wanted to dispense at any point in time.

Conclusion

This project work was instrumental in helping us understand and implement crucial concepts in Embedded Systems like power management, serial communication, timers, ADC and interrupts. We also had the opportunity to work with new sensors and components and in that attempt learnt their functioning and interfacing process with the Arduino. We are extremely proud of what we accomplished because of the use case that our product offers. We will finally be able to feed our pets without fretting about missing out on feeding them due to our tight schedule. The entire system is also portable allowing it to be carried without much distress.

Like every other project, we also faced a few challenges that we overcame as a team and this helped us to get the experience of ideating, strategizing and coming up with solutions to tackle such challenges. We had to efficiently plan our power management strategies because when we initially powered the integrated system, the relay module was drawing too much power and this largely affected the functioning of the dispenser as a whole.

Both the ultrasonic sensor (used for water dispensing: Mode 2) and the servo motor used the Timer 1 of the Arduino and this prevented us from using them simultaneously. This prompted us to use a secondary Arduino for the ultrasonic sensor with all the other setup controlled by the primary Arduino. Moreover, when we attempted to communicate information from the secondary Arduino to the primary one directly, we faced random errors which we later understood to be spurious outputs being produced. This was something we didn't anticipate. However, to combat this, we used debouncing to ensure that the transfer of information was smooth throughout. These were some of the challenges we encountered and tackled successfully.

The next step for PetCare would be to deploy it in our homes and use it forever :). We firmly believe that this product can be used to automatically dispense food and water once the time is set. They could also control the system through their smartphones using the Blynk IoT application. Apart from this, we believe that by putting it out in Devpost, anyone can take up this project and also add cool new features if they have any in mind.

Last edited on 2020/08/02 18:18 EDT

References

1) ATmega328P - Microcontroller
https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

2) DS3231 - Real Time Clock (RTC)
https://datasheets.maximintegrated.com/en/ds/DS3231.pdf

3) ESP8266 - Wi-Fi MCU module
https://cdn-shop.adafruit.com/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf

4) HC-SR04 - Ultrasonic Sensor
https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

5) FIT0563 - Submersible Water Pump

https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/FIT0563_Web.pdf

6) HiLetgo 2pcs 5V One Channel Relay Module Relay Switch with OPTO Isolation High Low-Level Trigger - Relay

http://www.hiletgo.com/ProductDetail/1958543.html

7) 426-SER0043 - Servo Motor

https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/SER0043_Web.pdf

8) 619-28090 - Water Level Sensor

https://www.mouser.com/pdfDocs/28090-Mini-Liquid-Level-Sensor-User-Manual.pdf

9) EZ-FLO 98562 PVC Clear Vinyl Tubing, 5/16 inch OD, 20 Ft - PVC Vinyl Tube

https://www.amazon.com/EZ-FLO-98562-Clear-Vinyl-Tubing/dp/B08SPSRJL4

Last edited on 2020/08/02 18:18 EDT

Appendix A

1) The code files for the entire project work have been added to the GitHub repository.
Link: https://github.com/upenn-embedded-courses/final-project-ese519f21-group26.git

2) The Video Demonstration has been submitted and also attached to Devpost Video Link:
https://drive.google.com/file/d/1U8uvn1JSCNEhLMqoLt98XlDQo-9CqSP5/view?usp=sharing

3) YouTube Video Link: https://youtu.be/ijcpfq7BCww

4) The External Libraries used and their Description has been added below

I. I2C communication library: twimaster.c and i2cmaster.h

A) i2c_init()

Functionality: It initializes the I^2C bus interface for transmission and needs to be called once. It initializes the TWI clock with a 100kHz frequency clock with no prescaling i.e. prescaler = 1.

Code Explanation:

- **TWSR = 0;**
  TWSR is set to 0 so that there is no prescaling.

- **TWBR = ((F_CPU/SCL_CLOCK)-16)/2;**
  TWBR is the TWI Bit rate register and its value should be greater than 10 for stable operation. In this implementation, the value is set to 72. The value of F_CPU is 16000000UL and that of SCL_CLOCK is 100000L.

B) i2c_start_wait()

Functionality: The function issues a start condition and sends the address and transfer direction for communication. In the scenario where the receiving device is busy, it uses ACK polling to wait until the receiving device is ready for communication.

Last edited on 2020/08/02 18:18 EDT

Code Explanation:

All of the following codes are wrapped around an infinite while loop so that the function waits till the address and transfer direction is sent.

- **TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);**
  This is done to transmit a start condition. It is important that TWINT is set to 1. Otherwise, the transmission process won't start.

- **while(!(TWCR & (1<<TWINT)));**
  After the start condition, the function waits till TWINT is cleared which signifies that the transmission process has been completed.

- **twst = TW_STATUS & 0xF8;**
  if ((twst != TW_START) && (twst != TW_REP_START)) continue;
  The function checks whether the START condition was successfully transmitted. If not the remaining instructions are bypassed using the continue instruction and the START condition is issued again at the beginning of the loop.

- **TWDR = address;**
  TWCR = (1<<TWINT) | (1<<TWEN);
  The address is stored into TWDR and then TWINT and TWEN is set to start the transmission of the address.

- **while(!(TWCR & (1<<TWINT)));**
  Again, the function waits till TWINT is cleared which signifies that the transmission process has been completed.

- **twst = TW_STATUS & 0xF8;**
  if ((twst == TW_MT_SLA_NACK)||(twst ==TW_MR_DATA_NACK))
  {
  /* device busy, send stop condition to terminate write operation */
  TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

  // wait until stop condition is executed and bus released
  while(TWCR & (1<<TWSTO));

Last edited on 2020/08/02 18:18 EDT

continue;

}

Now the function checks if the address was successfully transmitted. If that isn't the case, a STOP condition is transmitted to stop I^2C operation. Then the function waits till the STOP condition is executed and bus is released and uses the continue instruction to start the process from scratch in the next loop by bypassing the instructions below.

- **break;**
  The function only reaches this step if the address has been successfully transmitted. In that case, there is no need to stay inside the while loop and the break condition is used to exit the loop and return program control back to the main code.

C) i2c_write()

Functionality: The function sends one byte of data to the I^2C device. IF the transmission is successful, it returns a 0 and if the transmission fails it returns 1.

Code Explanation:

- **TWDR = data;**
  TWCR = (1<<TWINT) | (1<<TWEN);
  The data is stored into TWDR and TWINT and TWEN is set to start the transmission of the address.

- **while(!(TWCR & (1<<TWINT)));**
  The function waits till TWINT is cleared which signifies that the transmission process has been completed.

- **twst = TW_STATUS & 0xF8;**
  if(twst != TW_MT_DATA_ACK) return 1;
  return 0;
  The function checks whether the data was successfully transmitted by checking if the ACK bit is received. If the ACK hasn't been received, the function return 1 signifying the transmission failed. Otherwise, it goes to the next instruction and returns 1 signifying the transmission has successfully completed.

Last edited on 2020/08/02 18:18 EDT

D) i2c_stop()

Functionality: The function terminates the data transfer and releases the I^2C bus.

Code Explanation:

- **TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);**
  This is done to transmit the stop condition.
- **while(TWCR & (1<<TWSTO));**
  Then the function waits till the STOP condition is executed and the bus is released.

E) i2c_rep_start()

Functionality: The function is used to repeat the start function mentioned earlier and sends the address and transfer direction for communication

Code Explanation:

- **return i2c_start(address);**
  It calls the i2c_start() function defined earlier with the provided address.

F) i2c_readAck()

Functionality: The function reads one byte of data from the I^2C device and requests more data from the device

Code Explanation:

- **TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);**
  This is done so that the i2c device transmits data to the main device. Here, the TWEA bit controls the generation of the ACK pulse. The ACK pulse is created since a data byte is obtained by the master device. The generation of ACK pulse ensures that more data can be requested from the device.

- **while(!(TWCR & (1<<TWINT)));**

Last edited on 2020/08/02 18:18 EDT

The function waits till TWINT is cleared which signifies that the transmission process has been completed.

- **return TWDR;**
  The one byte of data stored in TWDR is returned to the main code.

G) i2c_readNak()

Functionality: The function reads one byte of data from the I^2C device and temporarily disconnects the I^2C device to stop data transfer.

Code Explanation:

- **TWCR = (1<<TWINT) | (1<<TWEN);**
  This is done so that the I^2C device transmits data to the main device. Here, since the TWEA bit is not set, the I^2C device is virtually disconnected temporarily so that data transfer stops.

- **while(!(TWCR & (1<<TWINT)));**
  The function waits till TWINT is cleared which signifies that the transmission process has been completed.

- **return TWDR;**
  The one byte of data stored in TWDR is returned to the main code.

II. DS1307 RTC Library: ds1307.h and ds1307.h

A) ds1307_dec2bcd()

Functionality: It transforms the given decimal value to binary coded decimal and returns the value.

Code Explanation:

- **return val + 6 * (val / 10);**
  The transforms the obtained decimal “val” into binary coded decimal and returns the BCD value.

Last edited on 2020/08/02 18:18 EDT

B) ds1307_bcd2dec()

Functionality: It transforms the given binary coded decimal value to decimal and returns the value.

Code Explanation:

- **val - 6 * (val >> 4);**
  The transforms the obtained binary coded decimal “val” into decimal and returns the value.

C) ds1307_date2days()

Functionality: Given the year, month and day, it calculates the number of days it has been since 1st January 2000 (2000/01/01).

Code Explanation:

- **uint16_t days = d;**
  Initialize a variable “days”.
- **for (uint8_t i = 1; i < m; ++i)**
  days += pgm_read_byte(ds1307_daysinmonth + i - 1);
  An integer array by the name exists in flash memory which holds the number of days in each month. For example, ds1307_daysinmonth[0] would be the number of days in January which is 31 and so on. This part of the code adds the number of days it has been till the given month since the start of the year.
- **if (m > 2 && y % 4 == 0)**
  ++days;
  The part of the code accounts for whether the given year is a leap year. If it is a leap year, “days” is incremented.
- **return days + 365 * y + (y + 3) / 4 - 1;**
  This returns the final number of days by adding the days variable to the number of days it has been since the year 2000 including the extra days due to leap year.

D) ds1307_getdayofweek()

Functionality: Returns the specific day in the week given the year, month and day.

Code Explanation:

- **uint16_t day = ds1307_date2days(y, m, d);**

Puts the number of days it has been since 2000/01/01 into the “day” variable.

- **return (day + 6) % 7;**
  Returns the specific day in the week in the form of integers from 0 to 6.

E) ds1307_isleapyear()

Functionality: Check if the given year is a leap year.

Code Explanation:

- **if (year % 400 == 0) {**
  return true;
  } else if (year % 100 == 0) {
  return false;
  } else if (year % 4 == 0) {
  return true;
  } else {
  return false;
  }
  Mathematical constraints to check whether the given is a leap year. If it is a leap year return True. Otherwise, return False.

F) ds1307_setdate()

Functionality: Sets a specific date and time into the ds1307 RTC module from where it starts counting. This is retained even when the main device is powered down because of the additional power supply of the RTC.

Code Explanation:

- **if (second < 0 || second > 59 ||**
  minute < 0 || minute > 59 ||
  hour < 0 || hour > 23 ||
  day < 1 || day > 31 ||
  month < 1 || month > 12 ||
  year < 0 || year > 99)
  return 2;
  Checks if the given values are in the proper format individually.
- **if(day > pgm_read_byte(ds1307_daysinmonth + month - 1) + (month == 2 && ds1307_isleapyear(year)))**

Last edited on 2020/08/02 18:18 EDT

return 1;
Checks if the given values are reasonable after considering the leap year constraints the number of days in a specific month constraint. If the conditions are violated, 1 is returned and no data is communicated with the I^2C device.

- **uint8_t dayofweek = ds1307_getdayofweek(year, month, day);**
  Stores day of the week in “dayofweek” variable.

- **i2c_start_wait(DS1307_ADDR | I2C_WRITE);**
  Initiates transmission from the Arduino UNO to the DS1307

- **i2c_write(0x00);**
  Signals DS1307 to start writing to memory address 0x00 in the DS1307

- **i2c_write(ds1307_dec2bcd(second));**
  Writes the BCD value for “second” into DS1307

- **i2c_write(ds1307_dec2bcd(minute));**
  Writes the BCD value for “minute” into DS1307

- **i2c_write(ds1307_dec2bcd(hour));**
  Writes the BCD value for “hour” into DS1307

- **i2c_write(ds1307_dec2bcd(dayofweek));**
  Writes the BCD value for “dayofweek” into DS1307

- **i2c_write(ds1307_dec2bcd(day));**
  Writes the BCD value for “day” into DS1307

- **i2c_write(ds1307_dec2bcd(month));**
  Writes the BCD value for “month” into DS1307

- **i2c_write(ds1307_dec2bcd(year));**
  Writes the BCD value for “year” into DS1307

- **i2c_write(0x00);**
  Signals the DS1307 to remain disconnected to the external oscillator.

- **i2c_stop();**
  Virtually stops transmission between the Arduino UNO and the DS1307 temporarily.

- **return 0;**
  Returns 0 to the main code to signify the date has been successfully set into the DS1307.

G) ds1307_getdate()

Functionality: Returns the year, month, day, hour, minute and second to the main code by reading them from the DS1307

Code Explanation:

- **i2c_start_wait(DS1307_ADDR | I2C_WRITE);**
  Initiates transmission from the Arduino UNO to the DS1307

- **i2c_write(0x00);**
  Writes 0x00 into DS1307.

- **i2c_stop();**
  Virtually stops transmission between the Arduino UNO and the DS1307 temporarily.

- **i2c_rep_start(DS1307_ADDR | I2C_READ);**
  Initiates transmission from the DS1307 to the Arduino UNO.

- ***second = ds1307_bcd2dec(i2c_readAck() & 0x7F);**
  Masks the unwanted bits and reads the seconds from DS1307 in BCD format, converts it into decimal format and stores it into the memory address held by the “second” pointer.

- ***minute = ds1307_bcd2dec(i2c_readAck());**
  Reads the minutes from DS1307 in BCD format, converts it into decimal format and stores it into the memory address held by the “minute” pointer.

- ***hour = ds1307_bcd2dec(i2c_readAck());**

Last edited on 2020/08/02 18:18 EDT

Reads the hour from DS1307 in BCD format, converts it into decimal format and stores it into the memory address held by the “hour” pointer.

- **i2c_readAck();**
  Reads a byte of unwanted data from the DS1307 which is discarded. This step can’t be skipped because data is sent sequentially by the DS1307.

- ***day = ds1307_bcd2dec(i2c_readAck());**
  Reads the day from DS1307 in BCD format, converts it into decimal format and stores it into the memory address held by the “day” pointer.

- ***month = ds1307_bcd2dec(i2c_readAck());**
  Reads the month from DS1307 in BCD format, converts it into decimal format and stores it into the memory address held by the “month” pointer.

- ***year = ds1307_bcd2dec(i2c_readNak());**
  Reads the year from DS1307 in BCD format, converts it into decimal format and stores it into the memory address held by the “year” pointer.

- **i2c_stop();**
  Virtually stops transmission between the Arduino UNO and the DS1307 temporarily.
