ЗМІСТ

ВСТУП

Згідно технічному завданню курсової, за основу при розробці електронного кухонного таймеру, було взято платформу Arduino. Вона – є найпростішою, розповсюдженою та найдешевшою платформою, для розробки електронних приладів для студентів. Arduino в своєму складі має гнучкий інтерфейс перепрограмування та композиційні властивості, що дозволяють повністю зробити автономну систему, яка відповідає вимогам поставленої мети.

Ринок мікроконтролерів може розглядатися як складна, слабо структурована система з великим числом входів. Система схильна до змін, швидкість яких в даний час швидко зростає. Вибір мікроконтролера при проектуванні або модернізації апаратури можна формулювати як рішення оптимальної задачі при трикомпонентного критерії (працездатність, надійність, вартість). Вибір здійснюється на безлічі варіантів, допустимих при даному стані ринку. Зростання обсягу необхідних операцій призводить до думки про необхідність автоматизації процедури вибору.

Мікроконтролер - комп'ютер на одній мікросхемі, призначений для управління різними електронними пристроями та здійснення взаємодії між ними відповідно до закладеної в мікроконтролер програмою.

З розвитком мікроелектронної індустрії, а зокрема з розвитком мікроконтролерів, з'явилася можливість робити малогабаритні і порівняно дешеві електронні вироби.

Arduino - апаратна обчислювальна платформа, що складається з двох основних компонентів: плата введення-виведення і середовище розробки на мові Processing / Wiring. Arduino зручна для розробки електронних пристроїв як для новачків, так і для професіоналів. Ця платформа користується величезною популярністю в усьому світі через простої мови програмування, відкритої архітектури і програмного коду. Особливість даної платформи є те

що вона програмується без використання программаторів через USB. За допомогою Arduino комп'ютер може вийти за рамки віртуального світу в фізичний, завдяки безлічі датчиків які можна підключити до плати. Датчики можуть отримувати інформацію про навколишнє середовище, а також управляти різними виконавчими пристроями.

Інтегроване середовище розробки Arduino - це поліплотформений додаток на Java, що включає в себе редактор коду, компілятор і модуль передачі прошивки в плату. Мова програмування використовується для Arduino дуже схожий на CI ++, доповнений деякими бібліотеками. Обробка програм здійснюється за допомогою препроцесора, а компілюється за допомогою AVR-GCC.

Мікроконтролери дозволяють менше використовувати типові елементи в розробках, тому що практично все можна зробити програмними засобами, тим самим електронні вироби зроблені на базі мікроконтролерів є малогабаритними і коштують порівняно не дорого.

В даний час, велика частина приладів побудована на цифровий логіці, головним елементом якої є мікроконтролер, і індикація відбувається на цифрових індикаторах, більшу частину з яких складають LCD.

Згідно технічному завданню курсової, за основу при розробці електронного кухонного таймеру, було взято платформу Arduino. Вона – є найпростішою, розповсюдженою та найдешевшою платформою, для розробки електронних приладів для студентів. Arduino в своєму складі має гнучкий інтерфейс перепрограмування та композиційні властивості, що дозволяють повністю зробити автономну систему, яка відповідає вимогам поставленої мети.

1. АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ ЗГІДНО ТЕХНІЧНОМУ ЗАВДАННЮ

1.1 Початок роботи з Arduino

Для того що б почати працювати з Arduino знадобиться наступне:

1) Плата Arduino.

2) USB-кабель.

3) Середовище розробки для Arduino.

Після того як придбано все необхідне підключаємо плату до комп'ютера. Arduino Uno, Mega, Duemilanove і Arduino Nano отримують живлення автоматично від будь-якого USB-підключення до комп'ютера або іншого джерела живлення. При використанні Arduino Diecimila необхідно переконатися, що плата налаштована для отримання харчування через USB-підключення. Джерело живлення вибирається за допомогою маленького пластикового джампера, надягнутого на два з трьох штирьків між роз'ємами USB і харчування. Необхідно перевірити, щоб він був встановлений на два штирі, найближчих до гнізда USB. Підключіть плату Arduino до комп'ютера, використовуючи USB-кабель. Повинний засвітитися зелений світлодіод харчування, позначений PWR. Наступним кроком буде установка драйверів, для наявної моделі. Після того як драйвера були встановлені, запускаємо середу розробки Arduino. У багатьох середовищах розробки Arduino вже є готовий приклад скетчу, який можна запустити і перевірити правильність підключення плати до комп'ютера. В налаштуваннях середовища розробки необхідно вказати модель плати Arduino, інакше навіть при правильно написаному скетчі, плата буде працювати неправильно. Так само необхідно вибрати послідовний порт. Виберіть пристрій послідовної передачі плати Arduino з меню Tools | Serial Port. Ймовірно, це буде COM3 або вище (COM1 і COM2 зазвичай резервуються для апаратних COM-портів). Щоб

знайти потрібний порт, можна від'єднати плату Arduino і повторно відкрити меню; пункт, який зник, і буде портом плати Arduino. Підключіть плату і виберіть послідовний порт. Далі перевіряємо працездатність плати.

Натискаємо кнопку «Upload» в програмі - середовищі розробки. Чекаємо кілька секунд - мигають світлодіоди RX і TX на платі. У разі успішного завантаження в рядку стану з'явиться повідомлення «Done uploading (Завантаження виконана). (Зауваження. Якщо мається на увазі плата Arduino Mini, NG або інша плата, то необхідно фізично кнопкою подати команду reset безпосередньо перед натисканням кнопки «Upload»). Кілька секунд після закінчення завантаження буде видно як світлодіод виведення 13 (L) на платі почне блимати помаранчевим кольором. Це свідчить про правильно підключенні плати до комп'ютера, її працездатності та правильній установці програмного забезпечення.

1.1.1 Цифрові виходи

Висновки платформи Arduino можуть працювати як входи або як виходи. В даному розділі описується функціонування висновків в цих режимах. Також необхідно звернути увагу на те, що більшість аналогових входів Arduino (Atmega) можуть конфігуроватся і працювати так само як і цифрові порти введення / виводу. Властивості порту введення / виводу (pin), сконфігурованого як порт введення. Висновки Arduino (Atmega) стандартно налаштовані як порти введення, таким чином, не потрібно явної декларації в функції pinMode (). Сконфігуровані порти введення знаходяться в високоімпедансних стані. Це означає те, що порт введення дає занадто малу навантаження на схему, в яку він включений. Еквівалентом внутрішньому опору буде резистор 100 МОм підключений до висновку мікросхеми. Таким чином, для перекладу порту введення з одного стану в інший потрібно маленьке значення струму. Це дозволяє застосовувати висновки мікросхеми для підключення ємнісного датчика торкання, фотодіода, аналогового датчика зі схемою, схожою на RC-ланцюг.

З іншого боку, якщо до цього висновку нічого не підключено, то значення на ньому братимуть випадкові величини, що наводяться електричними перешкодами або ємнісний взаємозв'язком з сусіднім висновком.

Підтягують (навантажувальні) резистори

Якщо на порт введення не надходить сигнал, то в даному випадку рекомендується поставити порту відоме стан. Це робиться додаванням підтягують резисторів 10 кОм, що включають вхід або до +5 В (підтягують до харчування резистори), або до землі (підтягують до землі резистори). Мікроконтролер Atmega має програмовані вбудовані підтягують до харчування резистори 20 ком. Програмування даних резисторів здійснюється наступним чином.

```
pinMode (pin, INPUT);  // призначити висновку порт введення
digitalWrite (pin, HIGH);  // включити «підтягаючий» резистор
```

«Підтягаючий» резистор пропускає струм достатній для того, щоб злегка світився світлодіод підключений до висновку, що працює як порт введення. Також легке світіння світлодіодів означає те, що при програмуванні висновок не був налаштований як порт виводу в функції pinMode (). Підтягує резистори управляються тими ж регістрами (внутрішні адреси пам'яті мікроконтролера), що управляють станами виведення: HIGH або LOW. Отже, якщо висновок працює як порт введення зі значенням HIGH, це означає включення підтягує до харчування резистора, то конфігурація функцією pinMode () порту виведення на даному виводі мікросхеми передасть значення HIGH. Дана процедура працює і у зворотному напрямку, тобто якщо висновок має значення HIGH, то конфігурація виведення мікросхеми як порту введення функцією pinMode () включить підтягуючий до харчування резистор. Примітка: Трудно використовувати висновок мікросхеми 13 в якості порту введення через підключених до нього світлодіоду і резистора. При підключенні підтягує до

харчування резистора 20 кОм на вводі буде 1.7 В замість 5 В, тому що відбувається падіння напруги на світлодіоді і включеному послідовно резистори. При необхідності використовувати висновок мікросхеми 13 як цифровий порт вводу потрібно підключити між висновком і землею зовнішній «підтягаючий» резистор. Властивості порту введення / виведення, сконфігурованого як порт виводу. Висновки, сконфігуровані як порти виводу, знаходяться в нізкоімпедансном стані. Дані висновки можуть пропускати через себе досить великий струм. Висновки мікросхеми Atmega можуть бути джерелом (позитивний) або приймачем (негативний) струму до 40 мА для інших пристроїв. Такого значення струму досить щоб підключити світлодіод (обов'язковий послідовно включений резистор), датчики, але недостатньо для більшості реле, соленоїдів і двигунів.

Короткі замикання висновків Arduino або спроби підключити енергоємні пристрої можуть пошкодити вихідні транзистори виведення або весь мікроконтролер Atmega. У більшості випадків дані дії приведуть до відключення виведення на мікроконтролері, але інша частина схеми буде працювати згідно з програмою. Рекомендується до виходів платформи підключати пристрої через резистори 470 Ом або 1 кОм, якщо пристрою не потрібно більший струм для роботи.

1.2 Аналогові входи

Опис портів, що працюють як аналогові входи, платформи Arduino (Atmega8, Atmega168, Atmega328, або Atmega1280)

1.2.1 Аналого-цифровий перетворювач

Мікроконтролер Atmega, використовувані в Arduino, містять шестиканальний аналого-цифровий перетворювач (АЦП). Дозвіл перетворювача складає 10 біт, що дозволяє на виході отримувати значення від 0 до 1023. Основним застосуванням аналогових входів більшості платформ Arduino є читання аналогових датчиком, але в той же час вони

мають функціональність вводів / висновків широкого застосування (GPIO) (то ж, що і цифрові порти введення / виводу 0 - 13).

Таким чином, при необхідності застосування додаткових портів введення / виводу є можливість конфігурувати невикористовуванні аналогові входи.

1.2.2 Основні входи та виходи мікроконтролеру

Висновки Arduino, відповідні аналоговим входам, мають номери від 14 до 19. Це відноситься тільки до висновків Arduino, а не до фізичних номерами висновків мікроконтролера Atmega. Аналогові входи можуть використовуватися як цифрові висновки портів введення / виводу. Наприклад, код програми для установки виведення 0 аналогового входу на порт виводу із значенням HIGH:

1) pinMode (14, OUTPUT);

2) digitalWrite (14, HIGH);

1.2.3 Резистори

Висновки аналогові входів мають підтягує резистори працюють як на цифрових висновках. Включення резисторів проводиться командою digitalWrite (14, HIGH); // включити резистор на виведення аналогового входу 0 поки висновок працює як порт введення. Підключення резистора вплине на величину яка надається функцією analogRead () при використанні деяких датчиків. Більшість користувачів використовує «підтягаючий» резистор при застосуванні виведення аналогового входу в його цифровому режимі.

Застереження

Для виведення, який працював раніше як цифровий порт виводу, команда analogRead буде працювати некоректно. В цьому випадку рекомендується налаштувати його як аналоговий вхід. Аналогічно, якщо висновок працював як цифровий порт виведення зі значенням HIGH, то зворотна установка на введення підключить підтягаючий резистор.

Керівництво на мікроконтролер Atmega не рекомендує проводити швидке переключення між аналоговими входами для їх читання. Це може викликати накладення сигналів і внести спотворення в аналогову систему. Однак після роботи аналогового входу в цифровому режимі може знадобитися налаштувати паузу між читанням функцією analogRead () інших входів.

1.3 Широтно-імпульсна модуляція

Широтно-Імпульсна модуляція, скорочено ШІМ (англ. PWM). Приклад використання аналогового виходу (ШІМ) для управління світлодіодом доступний з меню File-> Sketchbook-> Examples-> Analog програми Arduino. Широтно-Імпульсна модуляція, або ШИМ, це операція отримання змінюється аналогового значення за допомогою цифрових пристроїв. Пристрої використовуються для отримання прямокутних імпульсів - сигналу, який постійно перемикається між максимальним і мінімальним значеннями. Даний сигнал моделює напруга між максимальним значенням (5 В) і мінімальним (0 В), змінюючи при цьому тривалість часу включення 5 В щодо включення 0 В. Тривалість включення максимального значення називається шириною імпульсу. Для отримання різних аналогових величин змінюється ширина імпульсу. При досить швидкій зміні періодів включення-виключення можна подавати постійний сигнал між 0 і 5 В на світлодіод, тим самим керуючи яскравістю його свічення. На графіку зелені лінії відзначають постійні тимчасові періоди. Тривалість періоду обернено пропорційна частоті ШІМ. Тобто якщо частота ШІМ становить 500 Гц, то зелені лінії відзначатимуть інтервали тривалістю в 2 мілісекунди кожен. Виклик функції analogWrite () з масштабом 0 - 255 означає, що значення analogWrite (255) буде відповідати 100% робочого циклу (постійне включення 5 В), а значення analogWrite (127) - 50% робочого циклу. Для прикладу можна взяти платформу і почати трясти її взад і вперед. Для наших очі людини дане рух перетворює в лінії, що світяться мигання світлодіода.

Нарощування або зменшення ширини імпульсу на світлодіоді буде збільшувати або зменшувати світяться лінії світлодіода.

1.4 Пам'ять в Arduino

У мікроконтролері ATmega168, використовуваному на платформах Arduino, існує три види пам'яті:

1) Флеш-пам'ять: використовується для зберігання скетчів.

2) ОЗУ (Статична оперативна пам'ять з довільним доступом): використовується для зберігання і роботи змінних.

3) EEPROM (незалежна пам'ять): використовується для зберігання постійної інформації.

Флеш-пам'ять та EEPROM є незалежними видами пам'яті (дані зберігаються при відключенні харчування). ОЗУ є енергозалежною пам'яттю.

Мікроконтролер ATmega168 має:

1) 16 Кб флеш-пам'яті (2 КБ використовується для зберігання завантажувача)

2) 1024 байти ОЗУ

3) 512 байт EEPROM

Необхідно звернути увагу на малий обсяг ОЗУ, тому що велике число рядків в скетчі може повністю її витратити. Наприклад, наступна оголошення: char message [] = "I support the Cape Wind project."; займає 32 байта із загального обсягу ОЗУ (кожен знак займає один байт). При наявності великого обсягу тексту або таблиць для виведення на дисплей можливо повністю використовувати допустимі 1024 байти ОЗУ. При відсутності вільного місця в ОЗУ можуть статися збої програми, наприклад, вона може записатися, але не працювати. Для визначення даного стану потрібно перетворити в коментарі або вкоротити рядки скетчу (без зміни коду). Якщо після цього програма працює коректно, то на її виконання був витрачений весь обсяг ОЗУ. Існує кілька шляхів вирішення даної проблеми:

При роботі скетчу з програмою на комп'ютері можна перекинути частину даних або розрахунків на комп'ютер для зниження навантаження на Arduino.

При наявності таблиць пошуку або інших великих масивів можна використовувати мінімальний тип даних для зберігання значень. Наприклад, тип даних займає два байти, а byte - тільки один (але може зберігати невеликий діапазон значень).

Незмінні рядки і дані під час роботи скетчу можна зберігати у флеш-пам'яті. Для цього необхідно використовувати ключ PROGMEM. Для використання EEPROM зверніться до бібліотеки EEPROM.

1.5 Апаратні переривання в Arduino

Як підказує назва, переривання - це сигнали, що переривають нормальний перебіг програми. Переривання зазвичай використовуються для апаратних пристроїв, що вимагають негайної реакції на появу подій. Наприклад, система послідовного порту або UART (універсальний асинхронний приймач) мікроконтролера повинен бути обслужений при отриманні нового символу. Якщо цього не зробити швидко, новий символ може бути втрачений. При надходженні нового символу UART генерує переривання. Мікроконтролер зупиняє виконання основної програми (додатки) і перескакує на програму обробки переривань (ISR), призначену для даного переривання. В даному випадку це переривання за отриманим символу. Ця ISR захоплює новий символ з UART, поміщає в буфер, потім очищає переривання і виконує повернення. Коли ISR виконує повернення, мікроконтролер повертається в основну програму і продовжує її з точки виклику. Все це відбувається в фоновому режимі і не впливає безпосередньо на основний код вашої програми. Якщо запускається багато переривань або переривання генерує швидкодіючий таймер, основна програма буде виконуватися повільніше, так як мікроконтролер розподіляє своє машинний час між основною програмою і всіма функціями обробки переривань.

Наприклад, щоб побачити, наскільки важливі процеси переривання. Скажімо, у є послідовний порт зі швидкістю передачі даних 9600 бод. Це означає, що кожен біт символу посилається з частотою 9600 Гц або близько 10 кГц. На кожен біт йде 100 мкс. Близько 10 біт потрібно, щоб послати один символ, так що ми отримуємо один повний символ кожну мілісекунду або близько того. Якщо наш UART буферізований, ми повинні витягти останній символ до завершення прийому наступного, це дає на всю роботу 1 мс. Якщо UART НЕ буферізованого, необхідно позбутися від символу за 1 біт або 1 мкс. Розглянемо для початку буферізовані приклад. Потрібно перевіряти отримання байта швидше, ніж кожну мілісекунду, щоб запобігти втраті даних. Стосовно до Arduino це означає, що наша функція циклу повинна звертатися для читання статусу UART і можливо, байта даних, 1000 разів в секунду. Це легко здійснимо, але сильно ускладнить код, який вам потрібно написати. До тих пір, поки функція циклу не вимагає більше 1 мс до завершення, це може зійти з рук. Але можливо, що потрібно обслуговувати кілька пристроїв введення-виведення, або що необхідно працювати на набагато більшій швидкості передачі. Такі неприємності це незабаром може принести. З переривань не потрібно відстежувати надходження символу. Апаратура подає сигнал за допомогою переривання, і процесор швидко викличе ISR, щоб вчасно захопити символ. Замість виділення величезної частки процесорного часу на перевірку статусу UART, ви ніколи не повинні перевіряти статус, просто встановлюється апаратне переривання і виконуєте необхідні дії в ISR. Головна програма безпосередньо не зачіпається, і від апаратного пристрою не потрібно особливих можливостей.

1.5.1 Переривання по таймеру

В даному розділі буде описано на використання програмного таймера 2 для періодичних переривань. Вихідна ідея полягала у використанні цього таймера для генерації частоти биття в звукових проектах Arduino. Щоб

виводити тон або частоту нам потрібно перемикати порт введення-виведення на узгодженої частоті. Це можна робити з використанням циклів затримки. Це просто, але означає, що наш процесор буде зайнятий, нічого не виконуючи, але чекаючи точного часу перемикати. З використанням переривання по таймеру ми можемо зайнятися іншими справами, а висновок нехай перемикає ISR, коли таймер подасть сигнал, що час настав.

Необхідно тільки встановити таймер, щоб подавав сигнал з перериванням в потрібний час. Замість прокрутки марного циклу для затримки по часу, головна програма може робити щось інше, наприклад, контролювати датчик руху або керувати електроприводом. Що б не було потрібно проекту, більше нам не потрібно процесорний час для отримання затримок. Далі буде описано ISR в загальному тільки те, що стосується переривань таймера 2. Більш докладно про використання переривань в процесорах AVR можна прочитати в керівництві користувача avr-libc (англ). На даному етапі не потрібно повного розуміння, але, в кінцевому рахунку, ви можете захотіти отримати можливість прискорити використання переривань, раз це важливий інструмент для додатків на мікроконтролерах.

1.5.2 Таймери на Arduino

Arduino користується всіма трьома таймерами ATMega168:

1) Таймер 0 (Системний час, ШІМ 5 and 6)

Використовується для зберігання лічильника часу роботи програми. Функція millis () повертає число мілісекунд з моменту запуск програми, використовуючи ISR глобального збільшення таймера 0. Таймер 0 також використовується для реалізації ШІМ на висновках 5 і 6.

2) Таймер 1 (ШІМ 9 і 10)

Використовується для реалізації ШІМ для цифрових висновках 9 і 10.

3) Таймер 2 (ШІМ 3 і 11)

Використовується для управління виходами ШІМ для цифрових висновків 3 і 11.

Хоча все таймери використовуються, тільки Таймер 0 має призначену таймером ISR. Це означає, що можна захопити Таймер 1 і / або Таймер 2 під свої потреби. Однак в результаті можна буде використовувати ШІМ на деяких портах введення-виведення. Якщо планується використовувати ШІМ, необхідно пам'ятати про це.

1.5.3 Завантаження у мікроконтролер переривань

Щоб дати уявлення про ефект, припустимо, що таймер ISR запускався б кожні 20 мкс. Процесор, що працює на 16 МГц, може виконати близько 1 машинної команди кожні 63 мс або близько 320 машинних команд для кожного циклу переривання (20 мкс). Припустимо також, що виконання кожного рядка програми на С може зайняти багато машинних команд. Кожна інструкція, яка використовується в ISR, забирає час, доступне для виконання будь-якої іншої програми. Якби ISR використовувала близько 150 машинних циклів, було б витрачено половина доступного процесорного часу. При активних перериванння головна програма відкладалася б близько ½ часу, займаного їй в інших випадках. 150 машинних команд - не дуже велика програма на С, тому необхідно бути уважними.

Якщо буде занадто довга ISR, то головна програма буде виконуватися вкрай повільно, якщо ж ISR буде довшим, ніж тривалість циклу таймера, то практично ніколи не виконається ваша головна програма, і, крім того, в кінці кінців відбудеться збій системного стека.

1.5.4 Вимірювання завантаження переривань

Оскільки необхідно мати дуже швидкий таймер ISR, то потрібно виміряти, наскільки завантажені доступні ресурси. Для цього необхідний певний метод.

Таймер ні встановлено в режим, коли він перезавантажується автоматично. Це означає, що ISR повинна перезавантажити таймер для наступного інтервалу рахунки. Було б точніше мати автоматично перезавантажувати таймер, але, використовуючи цей режим, можна виміряти

час, що проводиться в ISR, і відповідно виправити час, завантажувати в таймер. Ключ у тому, що за допомогою цієї корекції при розумній точності, також отримуємо і число, що показує, скільки часу проводимо в ISR.

Метод полягає в тому, що таймер зберігає час, навіть якщо він переповнений і перерваний. В кінці ISR можна захопити поточне значення лічильника таймера. Це значення той час, який він відібрав у розробника до наступної точки програми. Це сумарний час, витрачений на перехід в процедуру переривання і виконання програми в ISR. Невелика помилка буде того, що не підраховується час, витрачений на команду перезавантаження таймера, але цю помилку можна виправити емпірично. Фактично саме тому використовується у формулі підрахунку завантажується значення 257 замість 256. Було виявлено дослідним шляхом, що це дає кращий результат. Зайвий такт компенсує команду перезавантаження таймера.

2. РОЗРОБКА ПРИНЦИПОВОЇ СХЕМИ ЕЛЕКТРОННОГО КУХОННОГО ТАЙМЕРУ

2.1 Розробка принципової схеми електронного кухонного таймеру

Людина проводить на кухні багато часу. Велика частина його йде на приготування їжі. А в цьому процесі дуже важливо стежити за часом варіння, смаження і т.д. Також часто підводить людський фактор. Господиня може відволіктися, забути, що у неї включена духовка або плита. У вирішення цієї проблеми може допомогти кухонний таймер. Один з його видів наведено тут.

В результаті аналізу літератури та патентної інформації було розроблено функціональну схему (Рис 2.1).

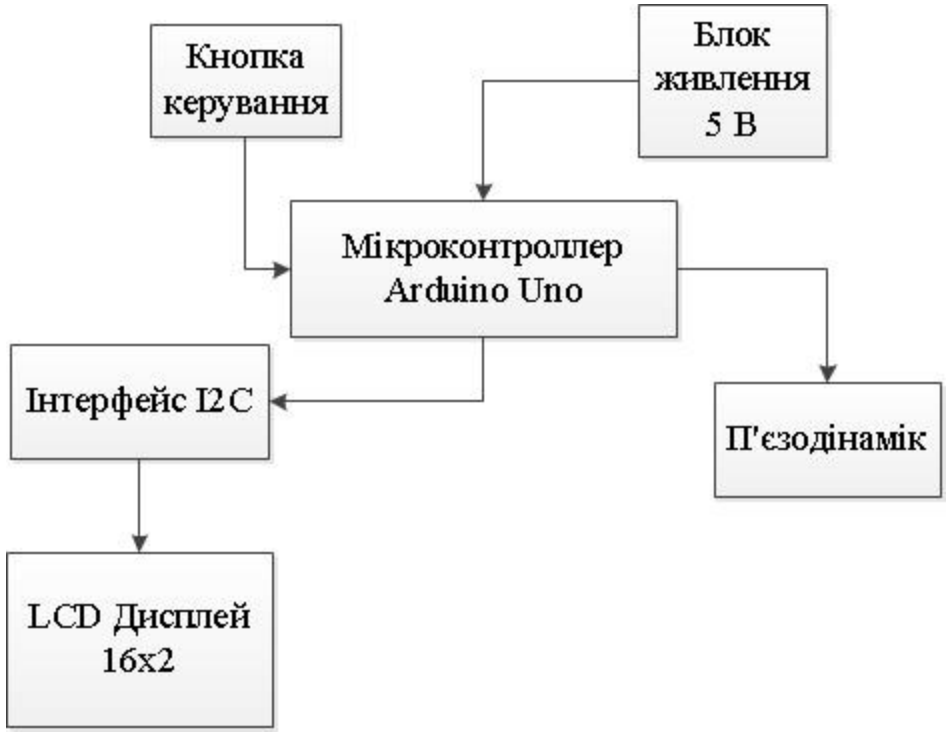

Рис 2.1. Функціональна схема електронного кухонного таймеру

Функціональна схема складається з:

Мікроконтролер Arduino - елемент, керуючий електронним стетоскопом, тобто запитувач датчик з певною періодичністю і перетворює його аналоговий електричний сигнал в цифровий.

Блок живлення – елемент, який відповідає за живлення усієї системи, відповідає опорному значенню 5 В.

П'єзодінамік – елемент, електроакустичний пристрій, здатний відтворювати звук, або випромінювати ультразвук, завдяки зворотному п'єзоелектричного ефект.

Інтерфейс I2C (***Inter-Integrated Circuit***)– елемент, послідовна асиметрична шина для зв'язку між інтегральними схемами всередині електронних приладів. Використовує дві двонаправлені лінії зв'язку (SDA і SCL), застосовується для з'єднання низькошвидкісних периферійних компонентів з процесорами і мікроконтролерами.

LCD дисплей – це елемент, різновид рідкокристалічного дисплея, в якому використовується активна матриця, керована тонко плівковими транзисторами.

2.2 Обґрунтування та визначення вимог до характеристик функціональних блоків електронного таймеру

2.2.1 Вибір мікроконтролера

Мікроконтролер для електронного стетоскопу повинен мати вбудований восьмирозрядний АЦП (аналогово-цифровий перетворювач), напруга живлення 5 В.

Існують різноманітні види керуючих мікроконтролерів. Контролери класифікують по розрядності:

Чотирьохрозрядний - найпростіші й дешеві пристрої, призначені для заміни нескладних схем на "жорсткою" логікою в системах з невисоким швидкодією. Типові випадки застосування-годинники, калькулятори, іграшки, прості пристрої управління.

Вісьмирозрядних - найбільш численна група (оптимальне поєднання ціни і можливостей). До цієї групи відносяться мікроконтролери серії MCS-51 (Intel) і сумісні з ними: PIC (MicroChip), HC68 (Motorola), Z8 (Zilog) та ін

Шестнадцатіразрядное - MCS-96 (Intel) та ін - більш високопродуктивні, але більш дорогі і менш поширені.

Тридцятидвухрозрядні - зазвичай є модифікаціями універсальних мікропроцесорів, наприклад i80186 або i386EX.

Для курсової роботи обираємо мікроконтролер Atmega328, фірми Atmel, який входить до складу Arduino Uno.

Мікроконтролер **ATMEGA328P**, має такі характеристики:

ATmega328P - мікроконтролер сімейства AVR, як і всі інші має 8-бітний процесор і дозволяє виконувати більшість команд за один такт. пам'ять:

- 32 kB Flash (пам'ять програм, що має можливість самопрограмування)
- 2 kB ОЗУ
- 1 kB EEPROM (постійна пам'ять даних)

Периферійні пристрої:

- Два 8-бітних таймера / лічильника з модулів порівняння і делителями частоти
- 16-бітний таймер / лічильник з модулем порівняння і дільником частоти, а також з режимом запису
- Лічильник реального часу з окремим генератором
- Шість каналів PWM (аналог ЦАП)
- 6-канальний ЦАП з вбудованим датчиком температури
- Програмований послідовний порт USART
- Послідовний інтерфейс SPI
- інтерфейс I2C

Програмований сторожовий таймер з окремим внутрішнім генератором

Внутрішня схема порівняння напруг

Блок обробки переривань і пробудження при зміні напруги на висновках мікроконтролера

Спеціальні функції мікроконтролера:

Скидання при включенні харчування і програмне розпізнавання зниження напруги живлення

Внутрішній тактовий генератор

Обробка внутрішніх і зовнішніх переривань

6 режимів сну (знижене енергоспоживання і зниження шумів для більш точного перетворення АЦП)

Напруги харчування і швидкість процесора:

- 1.8 - 5.5 В при частоті до 4 МГц
- 2.7 - 5.5 В при частоті до 10 МГц
- 4.5 - 5.5 В при частоті до 20 МГц

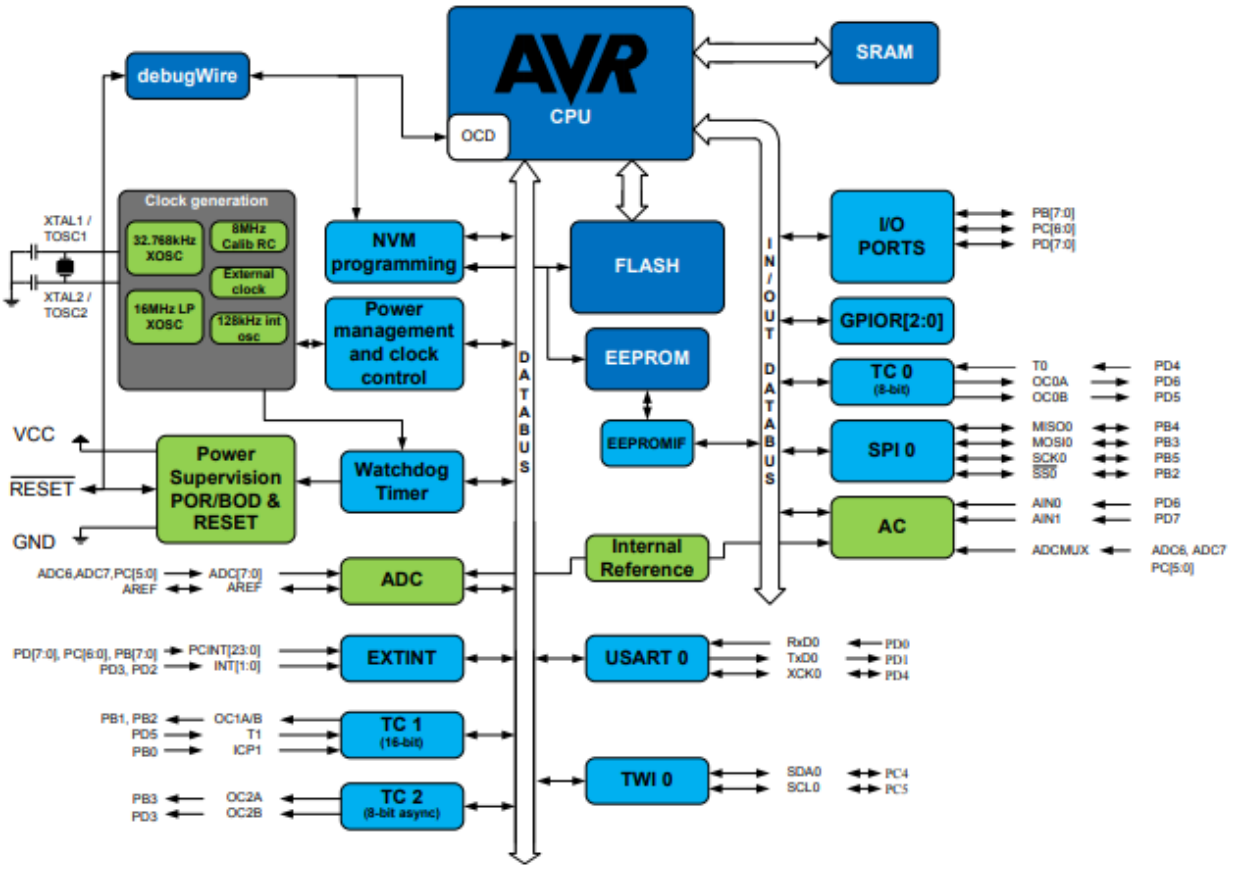

Рис 3.2 - Блок схема мікроконтролеру Atmega328P

2.2.2 Вибір пристрою виводу даних

Згідно з технічним завданням, індикатор повинен відображати значення часу від 5 секунд до 60 хвилин., З точністю 5 секунд, а значить індикатор повинен бути четирьохразрядним.. Вхідна напруга - 5 В. Існують наступні види індикаторів:

Семисегментний індикатори - індикатори, у яких управляється кожен сегмент:

- двійковій-десяткові індикатори - індикатори, керовані шістнадцятковим кодом, тобто можуть виводитися числа від 0 до 9 і букви англійського алфавіту від A до F.
- індикатори з динамічною індикацією - індикатори, у яких все розряди виводяться по черзі (мінімальна частота оновлення 30 Гц)
- програмовані індикатори - індикатори, керовані мікропроцесорами.

Для електронних таймера потрібно вибрати семисегментний індикатор, який має чотири розряду.

Розглянемо індикатор LFD2110-XX, який має такі характеристики:

- число розрядів - 4;
- вхідна напруга -1, 5..3 В;
- висота цифр - 7 мм;
- даний індикатор не підходить, вхідна напруга не 5 В;

Розглянемо індикатор LFD3162-XX, який має такі характеристики:
число розрядів - 4;

- вхідна напруга -1, 5..3 В;
- висота цифр - 9,2 мм;
- даний індикатор не підходить, вхідна напруга не 5 В;

Розглянемо індикатор LFD3164-XX, який має такі характеристики:

- число розрядів - 4;
- вхідна напруга -1, 5..3 В;

- висота цифр - 9,2 мм;
- даний індикатор не підходить ,. вхідна напруга не 5 В;
- індикатор HD44780 не підходить, тому що він програмований, в даній роботі не потрібен весь спектр його можливостей;

Розглянемо індикатор LCD1602, що має такі характеристики:

- Розміри 80 x 36 мм
- Робоча температура 0 ~ 50 ° C
- Підсвічування блакитне
- Колір символів білий
- Розмір символу 4.35 x 2.95мм
- Формат 16 x 2
- Розміри точки 0.5 x 0.5мм
- Інтерфейс HD44780
- Видима область 64.5 x 13.8мм
- Живлення 5В

Даний індикатор підходить за всіма критеріями. Він є рідкокристалічним, ці індикатори характеризуються низьким енергоспоживанням в порівнянні зі світлодіодними, але у них є і недоліки: погана видимість при поганому освітленні, погана працездатність при низькій температурі. Індикатор LCD 1602 зображено на рис 3.3.

Рис 3.3 – LCD дисплей 1602

2.2.3 Інтерфейс передачі даних I2C(IIC)

Шина I2 C підтримує будь-яку технологію виготовлення мікросхем (НМОП, КМОП, біполярну). Дві лінії, даних (SDA) і синхронізації (SCL), служать для перенесення інформації. Кожен пристрій має унікальний адрес і може працювати як передавач або приймач, в залежності від призначення пристрою. Крім того, пристрої можуть бути класифіковані як провідні і ведені при передачі даних. Ведучий - це пристрій, який ініціює передачу даних і виробляє сигнали синхронізації. При цьому будь-який пристрій, що адресується вважається відомим по відношенню до ведучого.

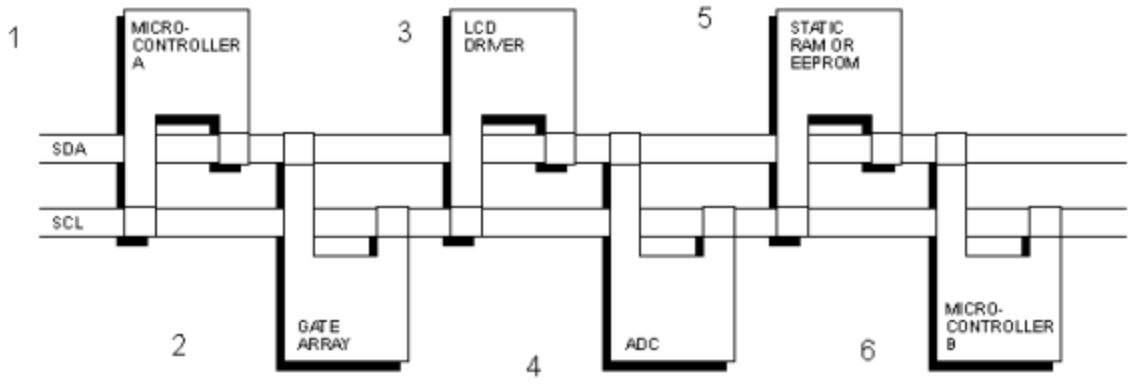

Рис. 3.4 - Приклад конфігурації шини I2C з двома мікроконтролерами

В інтерфейсі використовуються два дроти, це лінія тактування SCL, і лінія передачі даних SDA, які разом утворюють шину даних. Пристрої, підключені до шини підрозділяються на ведучого і веденого. Ведучий ініціюється процес передачі даних і видає тактові імпульси на лінію SCL, ведений приймає команди / дані, а також видає дані за запитом ведучого. Лінії SDA і SCL двонаправлені, пристрої підключаються до шини повинні мати висновки перенастроювати на вхід і вихід. Причому тип виходу повинен бути з відкритим колектором або відкритим стоком, в зв'язку з чим, обидві лінії SDA і SCL через резистори підтягуються до позитивного полюса джерела живлення. На наступній картинці приведена схема підключення інтерфейсу I2C

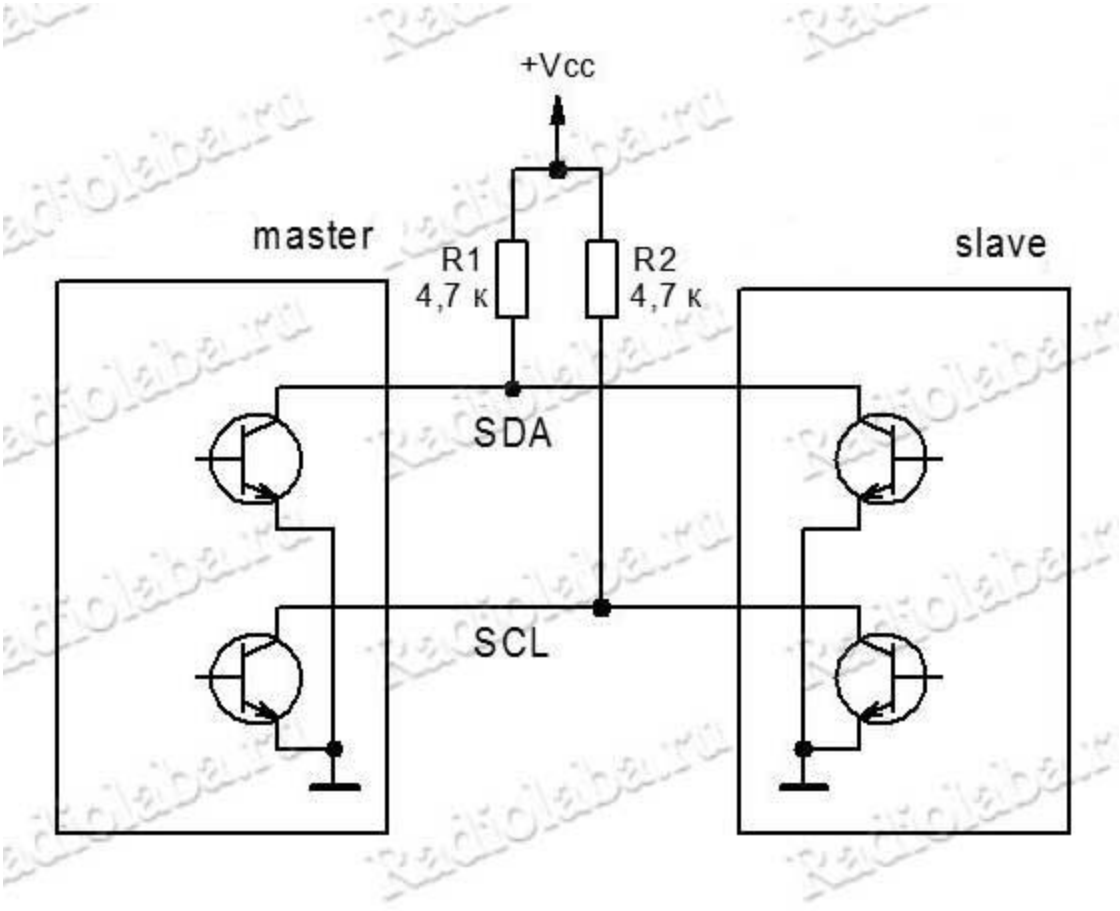

Рис. 3.5 - Схема підключення інтерфейсу I2C

У разі використання мікроконтролера, для установки лог. 1 на лінії, досить переналаштувати порт мікроконтролера на вхід, при цьому резистор "підтягне" лінію до високого логічного рівня, подача високого логічного рівня з порту мікроконтролеру на лінію не допускається. Для установки лог. 0 на лінії, порт перепрограмується на вихід, в вихідний порт заздалегідь записується значення 0, при цьому лінія "притискається" до низького логічного рівня.

В інтерфейсі передбачена програмна адресація пристроїв підключених до шини, найбільш поширена довжина адреси в 7 біт, теоретично це дозволяє підключати на шину до 127 пристроїв, але частина адрес по специфікації зарезервовані і не можуть використовуватися розробниками. Кожен пристрій має свою унікальну адресу, який закладений виробником і вказано в технічній документації. Адреса пристрою може бути фіксованим, або з можливістю апаратної настройки, в цьому випадку пристрій має додаткові

входи, в залежності від рівня напруги на входах (високий або низький), можна отримати різні адреси. Зазвичай кількість входів варіюється від 1-го до 3-х, які задають значення певних бітів 7-бітного адреси. Апаратна настройка адреси передбачена для можливості підключення декількох однотипних пристроїв на одну шину.

Також інтерфейс передбачає більш рідкісну 10-бітну адресацію, під яку зарезервований 7-бітову адресу 11110XX (XX-залежать від значення адреси), в цьому випадку спочатку віддається зарезервований адресу, в якому два останніх біта є старші біти 10-бітного адреси, потім передаються молодші 8 біт адреси. При використанні даної адресації на шину можна підключати більше 1000 пристроїв.

2.2.4 Вибір випромінювача звуку

Випромінювачі звуку (звукові випромінювачі, гучномовці, телефони, драйвери) - технічні пристрої, призначені для збудження звукових хвиль в різних середовищах шляхом перетворення електричного сигналу в енергію звукового поля.

На сьогоднішній день створено багато різних не схожих один на одного випромінювачів звуку, в зв'язку з чим, їх прийнято ділити на типи і види. Іноді, роблячи покупку будь-якої аудіосістеми, цікаво зрозуміти до якого типу і виду відносяться що містяться в ній випромінювачі. Але як це зробити? Адже про це нічого не сказано в специфікаціях.

Типи і види випромінювачів:

електромагнітні;

електродинамічні (котушкові, стрічкові, ізодінамічні, ортодинамичні, Хейла);

електростатичні (конденсаторні, електретні);

п'єзоелектричні (п`езокерамічні, біморфний).

Згідно технічному завданню розглянемо п'єзоелектричний випромінювач звуку.

П'єзоелектричні випромінювачі звуку - тип випромінювачів, в яких звук створюється п'єзоелементом, принцип дії зображено на рис. 3.6.

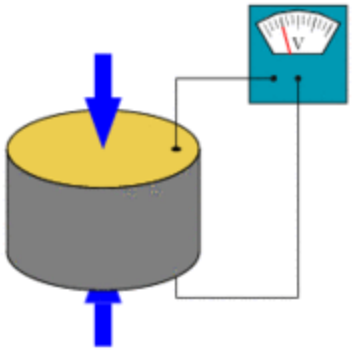

Рис.3.6 - Принципова схема п'єзоелектричного випромінювачу

Принцип дії п'єзоелектричних випромінювачів заснований на зворотному п'єзоелектричного ефекту (виникненні механічних деформацій під дією електричної напруги).

У п'єзоелектричних випромінювачів використовуються п'єзокерамічні або сегнетову пластини, край яких з'єднується з дифузором.

Електромеханічне перетворення відбувається так само, як і в конденсаторних випромінювачі і має ідентичні параметри. Однак за рахунок використання дифузора, з'являється притаманна електродинамічним випромінювачам значна нерівномірність АЧХ і нелінійні спотворення.

3. АПАРАТНА ЧАСТИНА

3.1 Розробка електричної принципової схеми

Електрична принципова схема електронного таймеру

В процесі розгляду багатьох середовищ моделювання було обрано те середовище, яке як найкраще підходить для курсової роботи. Було обрано середовище Proteus.

На принциповій електричній схемі мають бути відображені всі електричні зв'язки, тобто: блок живлення з мікроконтролером, кнопка програмування з входом мікроконтролеру, інтерфейс I2C для побудови схеми зв’язку між дисплеєм LCD 1602 та мікроконтролером, а також зв'язок між п'єзоелектричним випромінювачем та мікроконтролером(Рис 3.1).

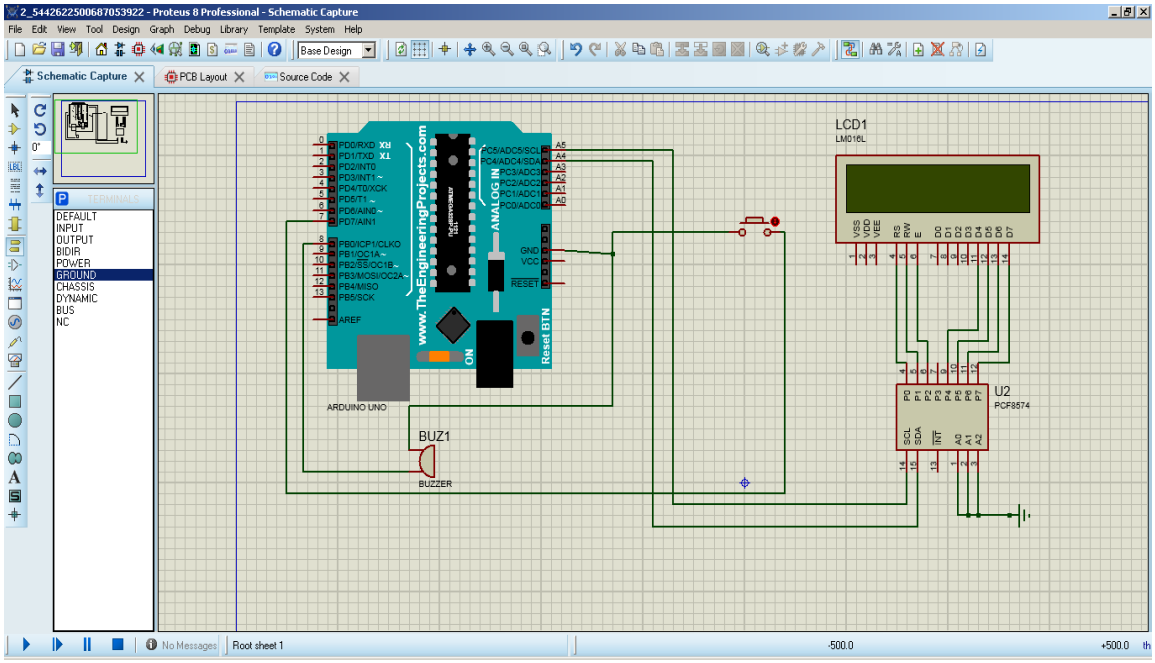

Рис. 3.1 - Електрична схема електронного таймеру

3.2 Розробка алгоритму роботи програми

Алгоритм роботи електронного таймеру повинен бути наступним:

Масштабування програми на ПК

Ініціалізація мікроконтролера Atmega8. Підготовка АЦП – настройка АЦП (номер каналу PC0-PC5, режим стандартний або точний, переривання), старт перетворення.

Індикація – сигнал про те що мікроконтролер готов почати роботу

Програмування початкового значення таймеру

Зчитування даних з АЦП. Перетворене число зберігається в регістрах.

Порівняння значень, пошук мінімального значення

Виведення на індикатор змінних.

Звукова індикація

Блок схема алгоритму програми має наступний вигляд:

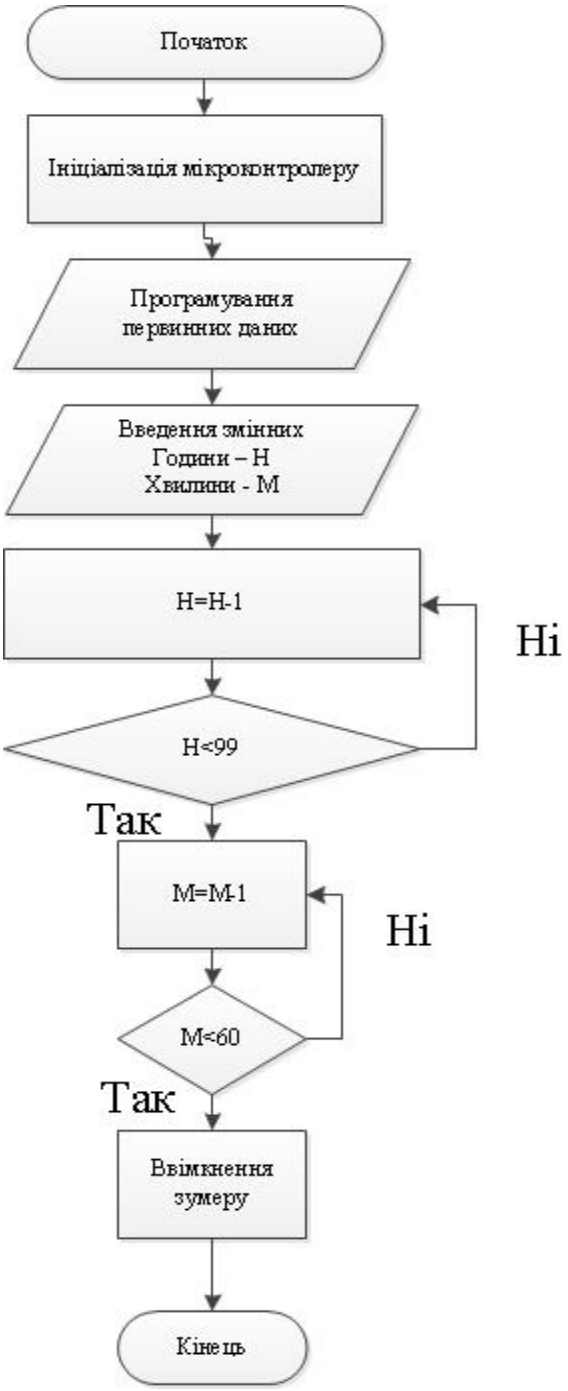

Рис 3.2 – Алгоритм роботи кухонного таймеру

3.3 Моделювання роботи електронного таймеру у середовищі Proteus

Для моделювання роботи електронного кухонного таймеру, будуємо електричну принципову схему Рис 3.4

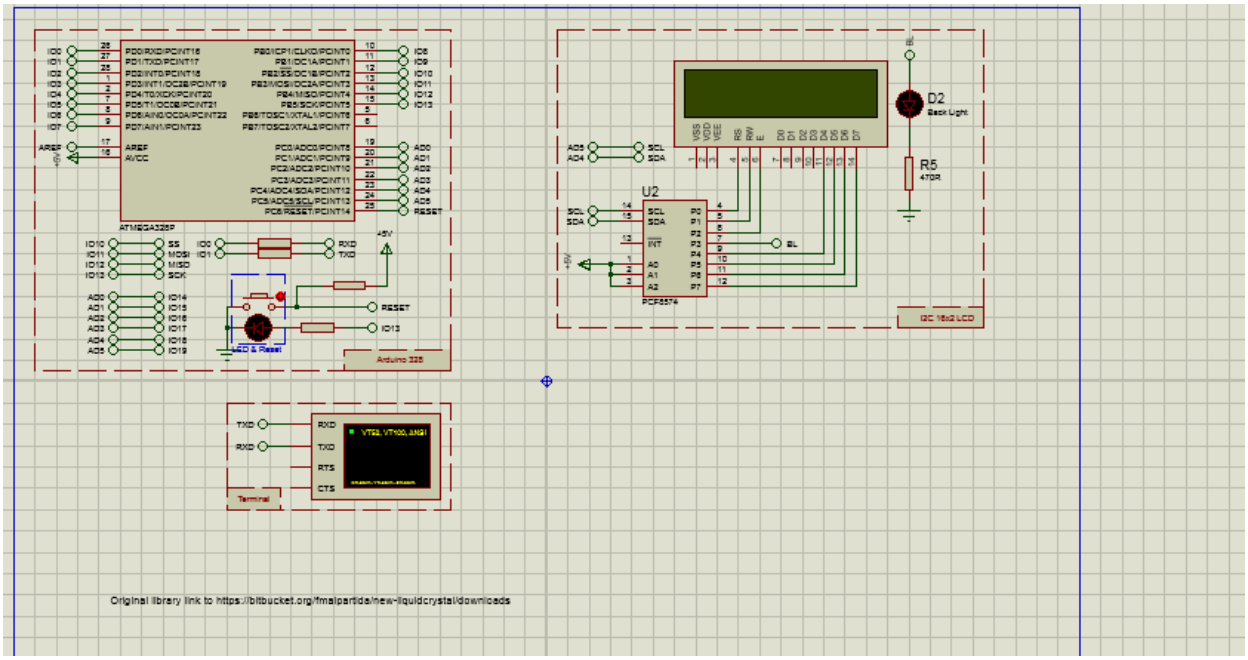

Рис. 3.4. - Електрична схема кухонного таймеру в Proteus.

Програмування мікроконтролеру

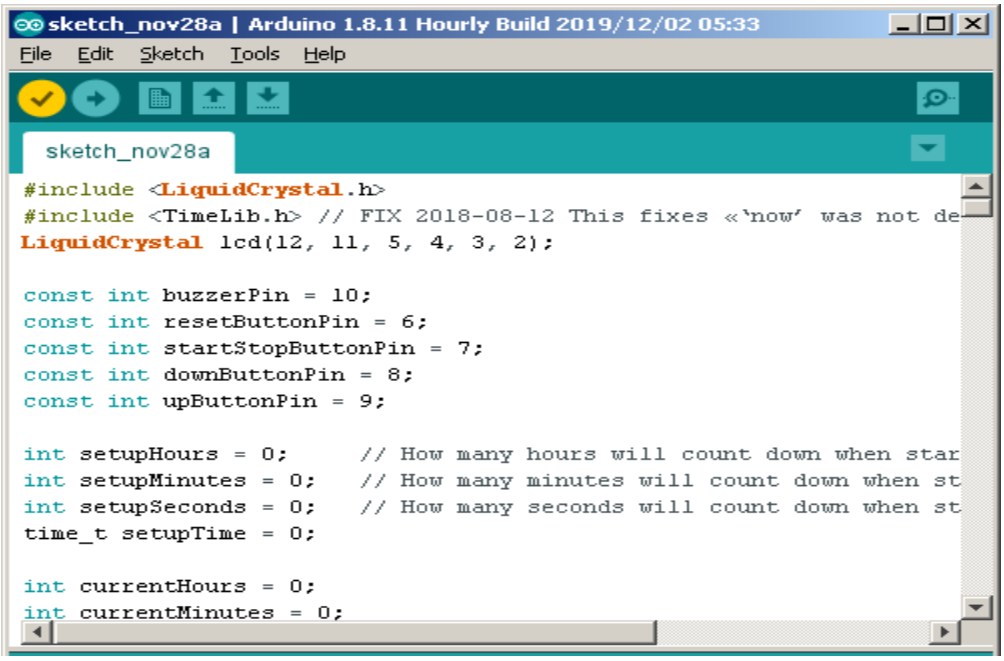

Рис 3.5. - Перевірка та запуск коду програми

Програма пройшла перевірку, помилок не має, код працює задовільно.

Запускаємо моделювання у середовищі Proteus

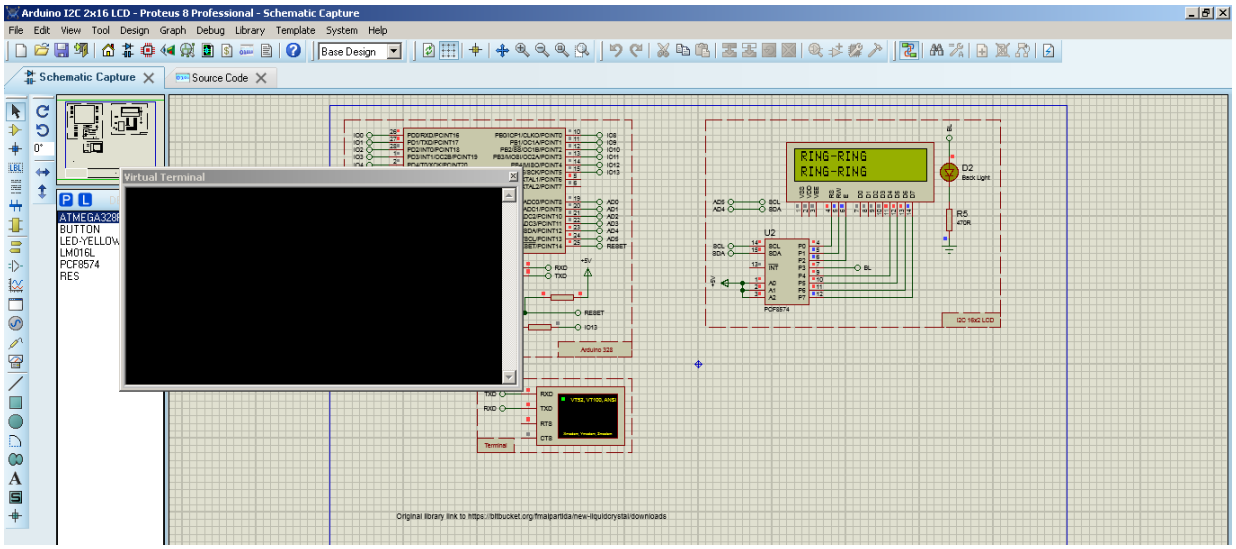

Рис. 3.6. - Запуск моделювання роботи електронного кухонного таймеру

ВИСНОВКИ

Результатом виконання курсової роботи є розробка електричної принципової схеми електронного кухонного таймеру на базі Arduino Uno та п’єзоелектричного випромінювача звуку, а саме:

- Було проведено аналіз літератури: платформа Arduino, випромінювачі звуку, різновиди LCD дисплеїв та інтерфейс I2C;
- Було створено електричну принципову схему електронного кухонного таймеру;
- Досліджено оптимальні параметри для побудови приладу: середу моделювання, доступність ринку електрокомпонентів;
- Вирішено ряд проблем, щодо узгодження зв’язку між LCD дисплеєм, інтерфейсом шини I2C та платформи Arduino Uno.
- Визначено напрями апаратного і програмного спрощення даного проекту.

Новизна даної розробки полягає в тому, що проведені дослідження: дозволяють спростити використання різної електроніки для побутових користувачів, а також для великої промисловості, шляхом доробки приладу: встановлення датчику забруднення повітря, відносної вологості, а також датчиком температури та пожежної безпеки.

Цей проект стане кроком для активного використання Arduino Uno в розробках автоматизованих пристроїв для безпеки життєдіяльності працівників побутових та промислових закладів, за рахунок гнучкості платформи.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Голубцов, М.С. Мікроконтролери AVR від простого до складного / М.С. Голубцов - М .: Салон-Пресс, 2003. - 288 с.

2. Макробертс, М. Почала Arduino / М. Макробертс - London: CUP, 2010. - 459 с.

3. Массімо, Б. Arduino для початківців чарівників / Б. Массімо - М .: VSD, 2012. - 128 с.

4. Соммер, У. Програмування мікроконтролерних плат Arduino / У. Соммер - Philadelphia: SIAM, 2012. - 241 с.

5. Еванс, Б. Arduino блокнот програміста / Б. Еванс - London: CUP, 2007. - 40 с.

6. Бєлов, А.В. Конструювання пристроїв на мікроконтролерах. / А.В. Бєлов - СПБ .: Наука і Техніка, 2005. - 256 с.

7. Гололобов, В.Н. З чого починаються роботи? / В.Н. Гололобов - 2011. - 189 с.

8. Предко, М. 123 експерименту з робототехніки / М. Предко - М .: НТ Пресс, 2007. - 271 с.

9. Суемацу, Е. Мікрокомп'ютерні системи управління. Перше знайомство / Е. Суемацу - М .: Видавничий дім «Додека - XXI», 2002. -256 с.

10. Бєлов, А.В. Мікроконтролери AVR в радіоаматорського практиці / А.В. Бєлов - СПБ .: Наука і Техніка, 2007. - 339 с.

11. Іванов, Ю. І. Мікропроцесорні пристрої систем управління: Навчальний посібник / Ю.І. Іванов - Таганрог: Видавництво ТРТУ, 2005. - 135 с.

12. Кнут, Д. Е. Т.2. Получисленние алгоритми. Глава 3. Випадкові числа / Дональд Е. Кнут // Мистецтво програмування. - 3-е изд. - М .: Вільямс, 2000. - 832 с.

13. Корабельников, Е.А. Самовчитель по програмуванню PIC контролерів для початківців / Е.А. Корабельников - М .: Салон-Пресс, 2008. - 287 с.

14. Трамперт, В. Вимірювання, управління і регулювання за допомогою AVR-мікроконтролерів / В.Трамперт - К .: «МК-Пресс», - 2006. - 208 с.

15. Парр, Е. Програмовані контролери: керівництво для інженера / Е. Парр - М .: БИНОМ. Лабораторія знань, - 2007. - 516 с.

ДОДАТОК

Текст програми

```
#include <LiquidCrystal.h>
#include <TimeLib.h> // FIX 2018-08-12 This fixes «‘now’ was not declared in this scope» error
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

const int buzzerPin = 10;
const int resetButtonPin = 6;
const int startStopButtonPin = 7;
const int downButtonPin = 8;
const int upButtonPin = 9;

int setupHours = 0;     // How many hours will count down when started
int setupMinutes = 0;   // How many minutes will count down when started
int setupSeconds = 0;   // How many seconds will count down when started
time_t setupTime = 0;

int currentHours = 0;
int currentMinutes = 0;
int currentSeconds = 0;
time_t currentTime = 0;

time_t startTime = 0;
time_t elapsedTime = 0;

int resetButtonState = LOW;
long resetButtonLongPressCounter = 0;
```

```
int startStopButtonState = LOW;
int upButtonState = LOW;
int downButtonState = LOW;
int resetButtonPrevState = LOW;
int startStopButtonPrevState = LOW;
int upButtonPrevState = LOW;
int downButtonPrevState = LOW;
bool resetButtonPressed = false;
bool resetButtonLongPressed = false;
bool startStopButtonPressed = false;
bool upButtonPressed = false;
bool downButtonPressed = false;

const int MODE_IDLE = 0;
const int MODE_SETUP = 1;
const int MODE_RUNNING = 2;
const int MODE_RINGING = 3;

int currentMode = MODE_IDLE;    // 0=idle 1=setup 2=running 3=ringing
                                // Power up --> idle
                                // Reset --> idle
                                //  Start/Stop --> start or stop counter
                                //  Up / Down --> NOP
                                // Reset (long press) --> enter setup
                                //   Start/Stop --> data select
                                //   Up --> increase current data value
                                //   Down --> decrease current data value
                                //   Reset --> exit setup (idle)
```

```
int dataSelection = 0;  // Currently selected data for edit (setup mode, changes with Start/Stop)
                        // 0=hours (00-99) 1=minutes (00-59) 2=seconds (00-59)

void setup() {
  // put your setup code here, to run once:
  lcd.begin(16, 2);
  pinMode(resetButtonPin, INPUT);
  pinMode(startStopButtonPin, INPUT);
  pinMode(upButtonPin, INPUT);
  pinMode(downButtonPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  startStopButtonPressed = false;
  upButtonPressed = false;
  downButtonPressed = false;

  /*
   * Reset button management
   */
  resetButtonPressed = false;
  resetButtonLongPressed = false;
  resetButtonState = digitalRead(resetButtonPin);
  if(resetButtonState != resetButtonPrevState)
  {
    resetButtonPressed = resetButtonState == HIGH;
```

```
  resetButtonPrevState = resetButtonState;
}
else  // Long press management...
{
  if(resetButtonState == HIGH)
  {
    resetButtonLongPressCounter++;
    if(resetButtonLongPressCounter == 100)
    {
      resetButtonPressed = false;
      resetButtonLongPressed = true;
      resetButtonLongPressCounter = 0;
    }
  }
  else
  {
    resetButtonLongPressCounter = 0;
    resetButtonPressed = false;
    resetButtonLongPressed = false;
  }
}

/*
 * Start/Stop button management
 */
startStopButtonPressed = false;
startStopButtonState = digitalRead(startStopButtonPin);
if(startStopButtonState != startStopButtonPrevState)
{
  startStopButtonPressed = startStopButtonState == HIGH;
```

```
  startStopButtonPrevState = startStopButtonState;
}

/*
 * Down button management
 */
downButtonPressed = false;
downButtonState = digitalRead(downButtonPin);
if(downButtonState != downButtonPrevState)
{
  downButtonPressed = downButtonState == HIGH;
  downButtonPrevState = downButtonState;
}

/*
 * Up button management
 */
upButtonPressed = false;
upButtonState = digitalRead(upButtonPin);
if(upButtonState != upButtonPrevState)
{
  upButtonPressed = upButtonState == HIGH;
  upButtonPrevState = upButtonState;
}

/*
 * Mode management
 */
switch(currentMode)
{
```

```
        case MODE_IDLE:
          if(resetButtonPressed)
          {
            Reset();
          }
          if(resetButtonLongPressed)
          {
            currentMode = MODE_SETUP;
          }
          if(startStopButtonPressed)
          {
            currentMode = currentMode == MODE_IDLE ? MODE_RUNNING :
MODE_IDLE;
            if(currentMode == MODE_RUNNING)
            {
              // STARTING TIMER!
              startTime = now();
            }
          }
          break;

        case MODE_SETUP:
          if(resetButtonPressed)
          {
            // Exit setup mode
            setupTime = setupSeconds + (60 * setupMinutes) + (3600 * setupHours);
            currentHours = setupHours;
            currentMinutes = setupMinutes;
            currentSeconds = setupSeconds;
            dataSelection = 0;
```

```
  currentMode = MODE_IDLE;
}
if(startStopButtonPressed)
{
  // Select next data to adjust
  dataSelection++;
  if(dataSelection == 3)
  {
    dataSelection = 0;
  }
}
if(downButtonPressed)
{
  switch(dataSelection)
  {
    case 0: // hours
      setupHours--;
      if(setupHours == -1)
      {
        setupHours = 99;
      }
      break;
    case 1: // minutes
      setupMinutes--;
      if(setupMinutes == -1)
      {
        setupMinutes = 59;
      }
      break;
    case 2: // seconds
```

```
      setupSeconds--;
      if(setupSeconds == -1)
      {
        setupSeconds = 59;
      }
      break;
  }
}
if(upButtonPressed)
{
  switch(dataSelection)
  {
    case 0: // hours
      setupHours++;
      if(setupHours == 100)
      {
        setupHours = 0;
      }
      break;
    case 1: // minutes
      setupMinutes++;
      if(setupMinutes == 60)
      {
        setupMinutes = 0;
      }
      break;
    case 2: // seconds
      setupSeconds++;
      if(setupSeconds == 60)
      {
```

```
            setupSeconds = 0;
          }
          break;
        }
      }
      break;

    case MODE_RUNNING:
      if(startStopButtonPressed)
      {
        currentMode = MODE_IDLE;
      }
      if(resetButtonPressed)
      {
        Reset();
        currentMode = MODE_IDLE;
      }
      break;

    case MODE_RINGING:
      if(resetButtonPressed || startStopButtonPressed || downButtonPressed ||
upButtonPressed)
      {
        currentMode = MODE_IDLE;
      }
      break;
  }

  /*
   * Time management
```

```
 */
switch(currentMode)
{
  case MODE_IDLE:
  case MODE_SETUP:
    // NOP
    break;
  case MODE_RUNNING:
    currentTime = setupTime - (now() - startTime);
    if(currentTime <= 0)
    {
      currentMode = MODE_RINGING;
    }
    break;
  case MODE_RINGING:
    analogWrite(buzzerPin, 20);
    delay(20);
    analogWrite(buzzerPin, 0);
    delay(40);
    break;
}

/*
 * LCD management
 */
//lcd.clear();
lcd.setCursor(0, 0);
switch(currentMode)
{
  case MODE_IDLE:
```

```
  lcd.print("Timer ready     ");
  lcd.setCursor(0, 1);
  lcd.print(currentHours);
  lcd.print(" ");
  lcd.print(currentMinutes);
  lcd.print(" ");
  lcd.print(currentSeconds);
  lcd.print("    ");
  break;
 case MODE_SETUP:
  lcd.print("Setup mode: ");
  switch(dataSelection)
  {
   case 0:
    lcd.print("HRS ");
    break;
   case 1:
    lcd.print("MINS");
    break;
   case 2:
    lcd.print("SECS");
    break;
  }
  lcd.setCursor(0, 1);
  lcd.print(setupHours);
  lcd.print(" ");
  lcd.print(setupMinutes);
  lcd.print(" ");
  lcd.print(setupSeconds);
  lcd.print("    ");
```

```
      break;
    case MODE_RUNNING:
      lcd.print("Counting down...");
      lcd.setCursor(0, 1);
      if(hour(currentTime) < 10) lcd.print("0");
      lcd.print(hour(currentTime));
      lcd.print(":");
      if(minute(currentTime) < 10)

lcd.print("0");
      lcd.print(minute(currentTime));
      lcd.print(":");
      if(second(currentTime) < 10) lcd.print("0");
      lcd.print(second(currentTime));
      break;
    case MODE_RINGING:
      lcd.print("   RING-RING!   ");
      lcd.setCursor(0, 1);
      lcd.print("                ");
      break;
  }
  delay(10);
}

void Reset()
{
  currentMode = MODE_IDLE;
  currentHours = setupHours;
  currentMinutes = setupMinutes;
  currentSeconds = setupSeconds;
```

```
}
```