Картина, която съдържа текст, Шрифт, Графика, лого

Описанието е генерирано автоматично

DAC - Digital Alarm Clock



Изготвено от: Марин Тодоров, Мартин Узунов и Георги Камбарев

Съдържание

1. Описание на проекта
2. Блокова схема
3. Електрическа схема
4. Списък на съставни части
5. Сорс код- описание на функционалността
6. Заключение

Описание на проекта

DAC е иновативна технология за изобразяване на текущото време и подпомагане на своите потребители чрез алармена функция. Проектът е изготвен с помощта на Tinkercad средата за обработка и симулация на Arduino технологии.

Блокова схема

Това е блок схемата на нашият проект. Usb кабелът директно се свързва към главната Arduino платка и така захранва целия проект.

От платката се подава ток на двата Breadboard-a, единият за контролите(Бутони, лампички и пиезо), а другият- за LCD екрана

Картина, която съдържа текст, екранна снимка, диаграма, План

Описанието е генерирано автоматично

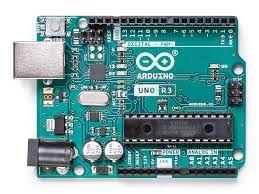
Електрическа схема

Картина, която съдържа текст, диаграма, Паралелен, линия

Описанието е генерирано автоматично

Схемата е много компактна и ефикасна. Използваме материали които са евтини и лесно достъпни. Материалите за проекта са:

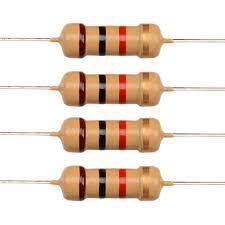
* Arduino Uno R3



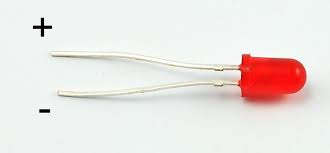
* LCD 16x2 екран



* 6 броя резистори по 1 кΩ



* 1 брой LED лампа



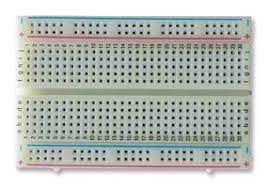
* 1 брой Piezo buzzer



* 4 броя бутони



* 2 броя Breadboard small



* Кабели

Сорс код

Сорс кодът за нашият проект е много компактен и кратък в сравнение с други подобни проекти. Всяка основна функционалност е изнесена в отделни методи и namespace-ове, за да максимизираме четимостта и ефикасността му.

#include <LiquidCrystal.h>

//Enum, съдържащ всички режими на часовника.

enum Mode : int {

SETTINGTIME = 0,

SETTINGALARM,

CLOCK

};

//Enum, съдържащ всички опции за редактиране на време.

enum SelectedSet : int {

HOURS = 0,

MINUTES,

SECONDS

};

//Enum, съдържащ всички режими на аларма.

enum AlarmMode : int {

ON = 0,

OFF,

ACTIVE

};

//Namespace, съдържащ константи с пиновете на ардуйното.

namespace Pins {

constexpr short ResetPin = 13;

constexpr short EnablePin = 12;

constexpr short DB4 = 11;

constexpr short DB5 = 10;

constexpr short DB6 = 9;

constexpr short DB7 = 8;

constexpr short SetTimePin = 7;

constexpr short IncrementPin = 6;

constexpr short SwitchPin = 5;

constexpr short DecrementPin = 4;

constexpr short PiezoPin = 3;

constexpr short IndicatorPin = 2;

}

//Namespace, съдържащ главни и често ползвани променливи за DAC.

namespace Globals {

//Избрана единица за редактиране.

SelectedSet CurrentSet = HOURS;

//Избран режим на часовника.

Mode CurrentMode = CLOCK;

//Режим на алармата.

AlarmMode AlarmMode = OFF;

//Брояч за текущо време.

long int ElapsedSeconds = 0;

//Променлива за време на алармата.

long int AlarmSeconds = 0;

//Променливи за инциализацията на диспей.

constexpr short LCDCols = 16;

constexpr short LCDRows = 2;

//Дисплей променлива.

LiquidCrystal LiquidCrystalDisplay(Pins::ResetPin, Pins::EnablePin, Pins::DB4, Pins::DB5, Pins::DB6, Pins::DB7);

}

//Namespace, съдържащ функции нужни за инициализация на DAC.

namespace Setup {

//Прави Interrupt, който ние използваме като главен таймер за основна функционалност на DAC.

void SetupInterrupt() {

//Магия.

TCCR1A = 0;

TCCR1B = 0;

TCNT1 = 0;

TCCR1B |= (1 << CS12) | (1 << CS10);

OCR1A = 15625;

TCCR1B |= (1 << WGM12);

TIMSK1 |= (1 << OCIE1A);

}

//Инициализира диспрей и сериален монитор.

void SetupOutput() {

Serial.begin(9600);

Globals::LiquidCrystalDisplay.begin(Globals::LCDCols, Globals::LCDRows);

}

//Инициализира входни и изгодни пинове.

void SetupInput() {

pinMode(Pins::SetTimePin, INPUT);

pinMode(Pins::IncrementPin, INPUT);

pinMode(Pins::SwitchPin, INPUT);

pinMode(Pins::DecrementPin, INPUT);

pinMode(Pins::PiezoPin, OUTPUT);

pinMode(Pins::IndicatorPin, OUTPUT);

}

}

//Namespace, съдържащ най-важните и ключови функции за функционалността на DAC.

namespace Core {

//Функция, която пуска и изключва светлинката, когато е подобаващо.

void HandleAlarmLight() {

if(Globals::AlarmMode == ON || Globals::AlarmMode == ACTIVE){

digitalWrite(Pins::IndicatorPin, HIGH);

}

else {

digitalWrite(Pins::IndicatorPin, LOW);

}

}

//Функция, която пуска и изключва звуковия сигнал на алармата.

void HandleAlarm(){

if(Globals::ElapsedSeconds >= Globals::AlarmSeconds && (Globals::AlarmMode == ON || Globals::AlarmMode == ACTIVE)){

Globals::AlarmMode = ACTIVE;

digitalWrite(Pins::PiezoPin, HIGH);

}

else {

digitalWrite(Pins::PiezoPin, LOW);

}

}

//Функция, която проверява и предотвратява невалидни времена за аларма и текъщо време.

void CheckForInvalidTimes(){

if(Globals::ElapsedSeconds >= 86400) {

Globals::ElapsedSeconds = 0;

Globals::LiquidCrystalDisplay.clear();

}

else if(Globals::ElapsedSeconds < 0) {

Globals::ElapsedSeconds = 86399;

Globals::LiquidCrystalDisplay.clear();

}

if(Globals::AlarmSeconds >= 86400) {

Globals::AlarmSeconds = 0;

Globals::LiquidCrystalDisplay.clear();

}

else if(Globals::AlarmSeconds < 0) {

Globals::AlarmSeconds = 86399;

Globals::LiquidCrystalDisplay.clear();

}

}

//Главна функция, която ползваме за контрол на дисплея, получава два аргумента - текст на първи и втори ред.

void PrintToLCD(String FirstRow, String SecondRow) {

Globals::LiquidCrystalDisplay.setCursor(0, 0);

Globals::LiquidCrystalDisplay.print(FirstRow);

Globals::LiquidCrystalDisplay.setCursor(0, 2);

Globals::LiquidCrystalDisplay.print(SecondRow);

Globals::LiquidCrystalDisplay.setCursor(15, 2);

switch(Globals::AlarmMode){

case OFF:

Globals::LiquidCrystalDisplay.print("D");

break;

case ON:

Globals::LiquidCrystalDisplay.print("E");

break;

case ACTIVE:

Globals::LiquidCrystalDisplay.print("A");

break;

}

if(Globals::CurrentMode == SETTINGTIME) {

Globals::LiquidCrystalDisplay.setCursor(12, 0);

Globals::LiquidCrystalDisplay.print("TIME");

}

else if(Globals::CurrentMode == SETTINGALARM){

Globals::LiquidCrystalDisplay.setCursor(11, 0);

Globals::LiquidCrystalDisplay.print("ALARM");

}

}

//Главна функция, която превръща изминали секунди в низ с време.

String SecondsToTimeString(long int Elapsed) {

long int Hours = Elapsed / 3600;

long int Minutes = (Elapsed - Hours \* 3600) / 60 ;

long int Seconds = (Elapsed - Minutes \* 60 - Hours \* 3600);

String Result {""};

if(Hours < 10) {

Result += "0";

}

Result += String(Hours);

Result += ":";

if(Minutes < 10) {

Result += "0";

}

Result += String(Minutes);

Result += ":";

if(Seconds < 10) {

Result += "0";

}

Result += String(Seconds);

return Result;

}

//Най-важна функция в проекта, справя се с входни данни от бутоните.

void HandleInput() {

if(digitalRead(Pins::SetTimePin) == HIGH) {

if(Globals::CurrentMode != CLOCK) {

Globals::CurrentMode = CLOCK;

Globals::CurrentSet = HOURS;

}

else {

Globals::CurrentMode = SETTINGTIME;

}

Globals::LiquidCrystalDisplay.clear();

delay(100);

}

if(digitalRead(Pins::IncrementPin) == HIGH) {

if(Globals::CurrentMode == SETTINGTIME) {

switch(Globals::CurrentSet) {

case HOURS:

Globals::ElapsedSeconds += 3600;

break;

case MINUTES:

Globals::ElapsedSeconds += 60;

break;

case SECONDS:

Globals::ElapsedSeconds += 1;

break;

}

delay(50);

}

else if(Globals::CurrentMode == SETTINGALARM) {

switch(Globals::CurrentSet) {

case HOURS:

Globals::AlarmSeconds += 3600;

break;

case MINUTES:

Globals::AlarmSeconds += 60;

break;

case SECONDS:

Globals::AlarmSeconds += 1;

break;

}

delay(50);

}

else if(Globals::CurrentMode == CLOCK){

Globals::LiquidCrystalDisplay.clear();

Globals::CurrentMode = SETTINGALARM;

}

}

if(digitalRead(Pins::SwitchPin) == HIGH) {

if(Globals::CurrentMode == SETTINGTIME || Globals::CurrentMode == SETTINGALARM) {

switch(Globals::CurrentSet) {

case HOURS:

Globals::CurrentSet = MINUTES;

break;

case MINUTES:

Globals::CurrentSet = SECONDS;

break;

case SECONDS:

Globals::CurrentSet = HOURS;

break;

}

Globals::LiquidCrystalDisplay.clear();

delay(100);

}

else if(Globals::CurrentMode == CLOCK && Globals::AlarmMode == ACTIVE) {

Globals::AlarmMode = ON;

Globals::AlarmSeconds = Globals::ElapsedSeconds + 540;

}

}

if(digitalRead(Pins::DecrementPin) == HIGH) {

if(Globals::CurrentMode == CLOCK && Globals::AlarmMode == ACTIVE) {

Globals::AlarmMode = OFF;

delay(100);

}

else if(Globals::CurrentMode == CLOCK && Globals::AlarmMode == ON) {

Globals::AlarmMode = OFF;

delay(100);

}

else if(Globals::CurrentMode == CLOCK && Globals::AlarmMode == OFF) {

Globals::AlarmMode = ON;

delay(100);

}

if(Globals::CurrentMode == SETTINGTIME) {

switch(Globals::CurrentSet) {

case HOURS:

Globals::ElapsedSeconds -= 3600;

break;

case MINUTES:

Globals::ElapsedSeconds -= 60;

break;

case SECONDS:

Globals::ElapsedSeconds -= 1;

break;

}

delay(50);

}

else if(Globals::CurrentMode == SETTINGALARM) {

switch(Globals::CurrentSet) {

case HOURS:

Globals::AlarmSeconds -= 3600;

break;

case MINUTES:

Globals::AlarmSeconds -= 60;

break;

case SECONDS:

Globals::AlarmSeconds -= 1;

break;

}

delay(50);

}

}

}

//Функция, която изписва подобаваща информация на дисплея, използва функцията по-горе за помощ.

void HandleLCD(){

switch(Globals::CurrentMode) {

case SETTINGTIME:

switch(Globals::CurrentSet) {

case HOURS:

Core::PrintToLCD(Core::SecondsToTimeString(Globals::ElapsedSeconds), "ADJUST HOURS");

break;

case MINUTES:

Core::PrintToLCD(Core::SecondsToTimeString(Globals::ElapsedSeconds), "ADJUST MINUTES");

break;

case SECONDS:

Core::PrintToLCD(Core::SecondsToTimeString(Globals::ElapsedSeconds), "ADJUST SECONDS");

break;

}

break;

case SETTINGALARM:

switch(Globals::CurrentSet) {

case HOURS:

Core::PrintToLCD(Core::SecondsToTimeString(Globals::AlarmSeconds), "ADJUST HOURS");

break;

case MINUTES:

Core::PrintToLCD(Core::SecondsToTimeString(Globals::AlarmSeconds), "ADJUST MINUTES");

break;

case SECONDS:

Core::PrintToLCD(Core::SecondsToTimeString(Globals::AlarmSeconds), "ADJUST SECONDS");

break;

}

break;

case CLOCK:

Core::PrintToLCD(Core::SecondsToTimeString(Globals::ElapsedSeconds) + " CLOCK" , Core::SecondsToTimeString(Globals::AlarmSeconds) + " ALARM");

break;

}

}

}

//Setup функция.

void setup(){

//Инизиалицира всичко нужно за работата на проекта.

Setup::SetupInterrupt();

Setup::SetupOutput();

Setup::SetupInput();

}

//Главен цикъл на DAC. Тук се случва почти цялата функционалност.

void loop(){

//Извиква всички функции, нужни за работата на проекта.

Core::HandleInput();

Core::HandleLCD();

Core::CheckForInvalidTimes();

Core::HandleAlarm();

Core::HandleAlarmLight();

}

//Interrupt-а, който създадохме. Служи като таймер, извиква се веднъж в секунда.

ISR(TIMER1\_COMPA\_vect){

//Увеличава преминалите секунди с една.

if(Globals::CurrentMode) {

Globals::ElapsedSeconds++;

}

}\\

Заключение

Ние успешно създадохме симулация на дигитален часовник с функционалност на аларма. Проектът бе създаден с удоволствие и предполага подобрения по себе си, например допълнителна функционалност- таймер и хронометър. Идеята, както и изпълнението й са вдъхновени от интернет.

Нашият проект има своите недостатъци, но общо взето мислим че се правихме блестящо.

Сорс кода към проекта е достъпен на: <https://github.com/MartinUzunov1707/Digital-Alarm-Clock>

ТинкерКад симулация на проекта е достъпна на:

<https://www.tinkercad.com/things/8HscRCGxLn0?sharecode=L6KejLkBksW4paYq5uTkBQLMJ2RhMuoNMXYugjRcK98>