Systemy Wbudowane - Projekt Mini Lodówki

Mateusz Bardel
Wydział Fizyki i Informatyki Stosowanej
Akademia Górniczo Hutnicza im. Stanisława Staszica

Styczeń 2023

Spis treści

1 Wymagania wstępne

Cel projektu

Projekt dotyczy prototypu małej lodówki opartej na ogniwie Peltiera. Urządzenie jest w stanie schłodzić małą ilość napoju do odczuwalnie niskiej temperatury.

Wymiary lodówki pozwalają na trzymanie jej na widoku i w zasięgu ręki. Prosty interfejs użytkownika pozwala na szybkie i intuicyjne ustawienie parametrów pracy.

Zakres projektu

System składa się z centralnego kontrolera, czujnika temperatury, interfejsu użytkownika, oraz tranzystorów sterujących chłodzeniem.

Całość jest sterowana przez płytkę WeAct Studio Blackpill 3.0, opartej na mikro-kontrolerze STM32F401CDU6. Zadaniem płytki jest odczytywanie temperatury komory, przechowywanie danych, sterowanie PID, kontrola mocy, oraz obsługa interfejsu użytkownika.

Temperatura wewnętrznego radiatora jest mierzona poprzez czujnik DS18B20. Komunikacja z mikro-kontrolerem odbywa się za pomocą protokołu OneWire i UART.

Interfejs użytkownika składa się z enkodera EC11, przycisku mechanicznego, oraz małego wyświetlacza oled. Przyciski i enkoder są obsługiwane przerwaniami. Wyświetlacz ssd1306 jest sterowany protokołem I2C.

Ogniwo Peltiera i wentylatory są odpowiedzialne za obniżenie temperatury. Elementy chłodzące są sterowane wyjściami GPIO kontrolera za pomocą mosfetów IRF540N. Ogniwo jest sterowane sygnałem PWM.

W skład projektu wchodzi jeszcze modelowanie i wydrukowanie obudowy zewnętrznej, oraz zestawienie gotowego rozwiązania na płytce proto-board.

2 Hardware

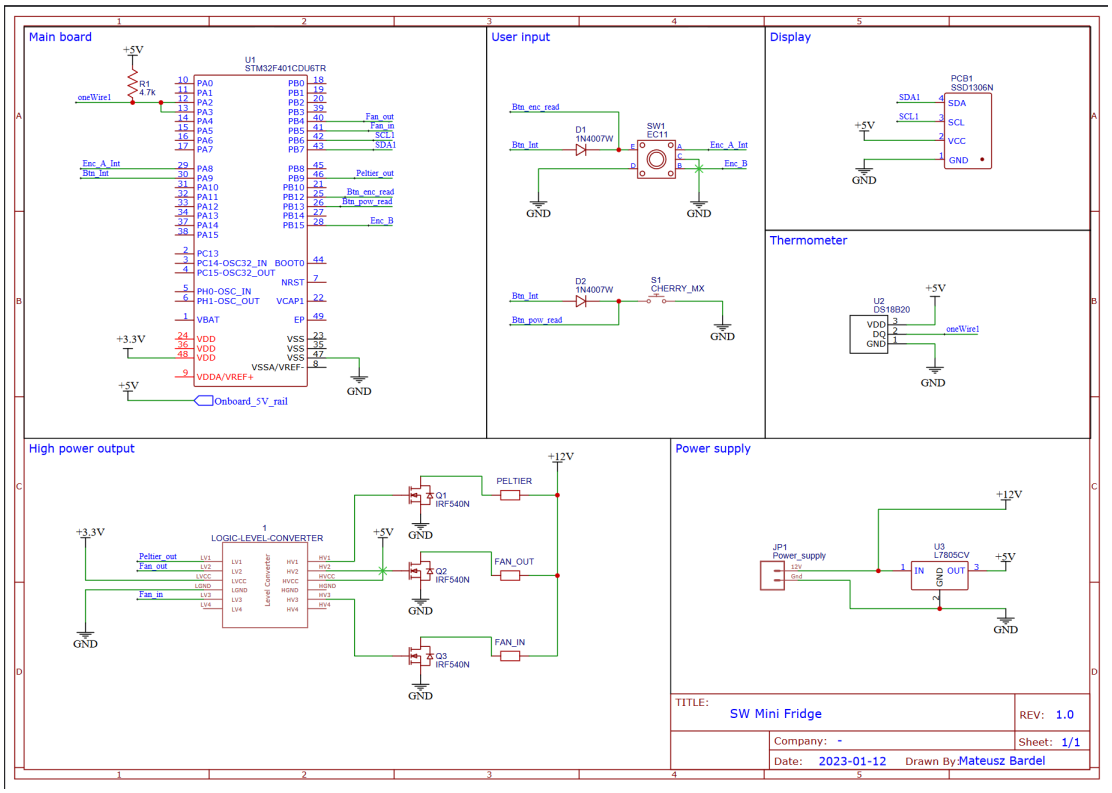
Obudowa

Części zaprojektowano parametrycznie w programie Fusion360. Elementy zostały wydrukowane na drukarce 3D w technologii FDM z polimerów PLA i PETG.

Rysunek 1: Obudowa lodówki

Schemat

Komponenty elektroniczne zostały zalutowane na płytce proto-board. Peryferia są podpięte za pomocą złącz JST-XH, w celu wygodnego montażu i debugowania. Dla czytelności złącza nie są zawarte w schemacie.



Rysunek 2: Schemat elektroniki

Rysunek techniczny i schemat są dostępne do wglądu w oddzielnych plikach.

Tabela 1: Lista wybranych komponentów

Nazwa komponentu	Ilość
WeAct Studio Black Pill V3.0	1
Ogniwo Peltiera TEC1-12704	1
Termometr DS18B20	1
Wyświetlacz ssd1306	1
Enkoder EC11	1
Przycisk Cherry MX Blue	1
Radiator AMD Wraith Stealth	1
Wentylator 5V 3030	1
Mosfet IRF540N	3
Regulator L7805V	1
Zasilacz sieciowy 12V 4A	1

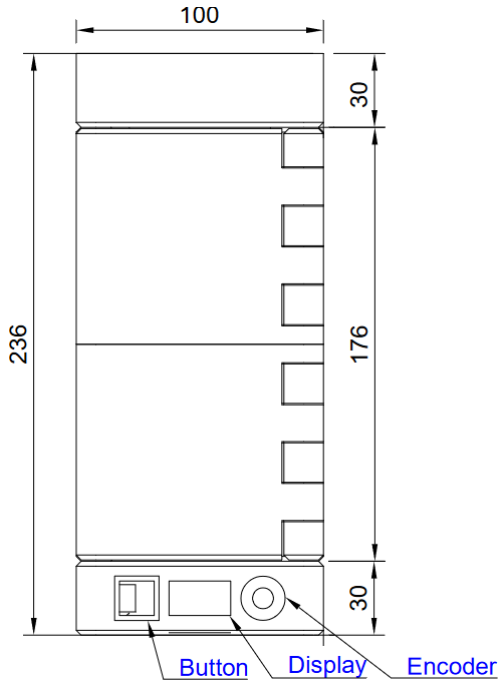
3 Instrukcja użytkownika

Sterowanie

Projekt włącza się od razu po podpięciu do zasilania. Prototyp może pracować w trzech trybach:

- Docelowej temperatury - Sterownik PID mikro-kontrolera stara się utrzymać temperaturę wybraną przez użytkownika.
- Docelowej mocy - Ogniwo działa z mocą ustawioną przez użytkownika.
- Uśpienia - Wentylatory, ogniwo, oraz wyświetlacz są wyłączone.

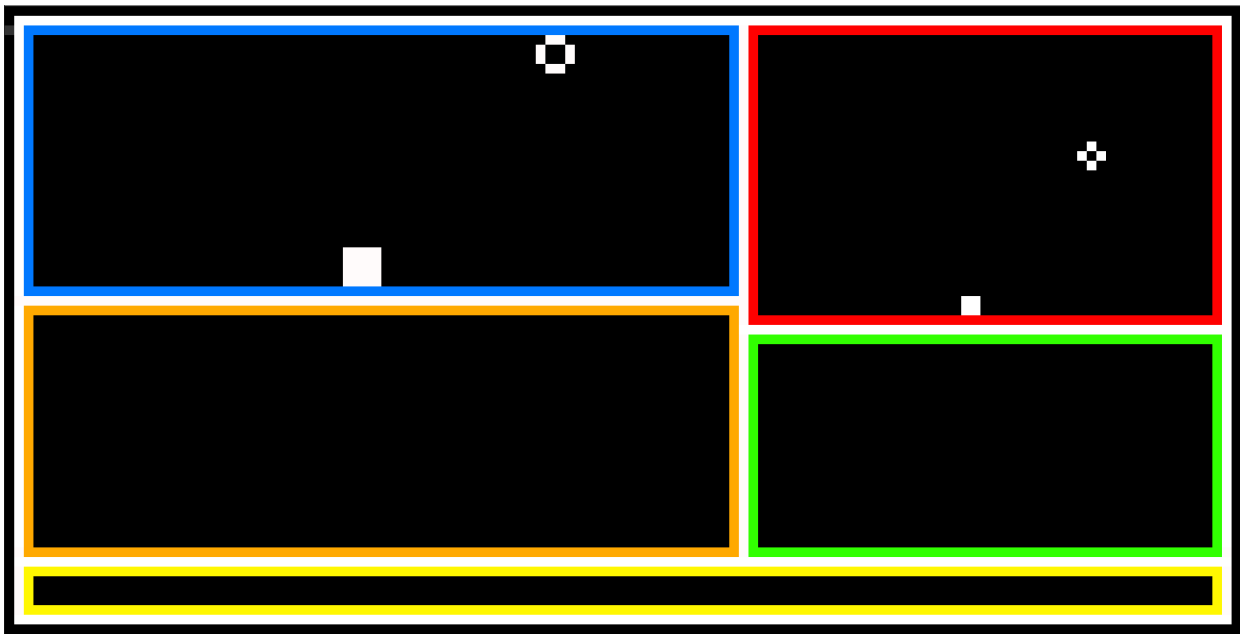
Kwadratowy przycisk mechaniczny przełącza tryb uśpienia.
Przycisk na enkoderze zmienia pozostałe tryby pracy.
Obrót gałki enkodera zwiększa/zmniejsza docelową temperaturę, lub moc.



Rysunek 3: Panel sterowania

Ekran

Wyświetlacz OLED przedstawia w 5 obszarach następujące informacje:



Rysunek 4: Obszary GUI

- Niebieski - Aktualna temperatura wewnątrz lodówki.
- Czerwony - Docelowa temperatura lodówki.
- Zielony - Docelowa/aktualna moc ogniwa Peltiera.
- Pomarańczowy - Inicjały autora, skrót uczelni, przedmiotu i rok.
- Żółty - Animowany pasek wyświetlający aktualną moc z jaką pracuje ogniwo Peltiera.

4 Analiza kodu

Narzędzia i biblioteki

Do programowania wykorzystano środowisko STM32CubeIde, oraz program STM32CubeMx.

Wykorzystano biblioteki:

- STM 32 HAL - Hardware Abstraction Layer - Inicjalizacja i zarządzanie timerami, przerwaniami, GPIO.
- lwow - Lightweight one-wire - Odczyt czujnika temperatury za pomocą protokołu one-wire na wyjściu UART.
- stm32-ssd1306 - Sterowanie wyswietlaczem ssd1306.

Bitmapę GUI narysowano narzędziem online Piskel. Napisano jeszcze skrypt Pythonowy konwertujący małe obrazy GIF na odpowiednie macierze w C.

Inicjalizacja

Firmware zawiera strukturę danych przechowującą zmienne związane z działaniem lodówki.

```
/*
 * Data model
 */

typedef struct Data{

    float Kp;
    float Ki;
    float Kd;
    float integral;

    float measuredTemp;
    int targetTemp;
    int currentPow;

    enum operationMode mode;
    char* statusMessage;

} Data;
```

Rysunek 5: Struktura z danymi

Inicjalizacja następuje w funkcji main(). Wyświetlacz startuje na początku, żeby poinformować użytkownika o potencjalnym problemie z czujnikiem. Następnie wykonywany jest skan urządzeń na wyjściu OW. Dla bezpieczeństwa program nie może kontynuować bez działającego czujnika.

Ponieważ mamy tylko jedno urządzenie OW, wybieramy pierwszy rom z listy rom_ids. Ustawiamy rozdzielczość pomiarów.

Startuje timer 4 sterujący wyjściem PWM kontrolera.

Startuje timer 3 odczytujący temperaturę i zarządzający logiką.

Jeżeli wszystko działa bez problemu, usuwamy powiadomienie, włączamy tryb chłodzenia i wentylatory.

```
Data data = {30,0.15,30,0,0,100,0,POWER_OFF,""};    //Data model used by pretty much everything


int main(void)
{

  HAL_Init();
  SystemClock_Config();

  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_USART2_UART_Init();

  //Init display
  menuDisplay_Init();
  data.statusMessage = "LWOW scan";
  menuDisplay_Update(0);

  //Init LWOW
  lwow_init(&ow, &lwow_ll_drv_stm32_hal, &huart2);

  //Thermometer must be found to ensure safe operation.
  while(!scan_onewire_devices(&ow, rom_ids, LWOW_ARRAYSIZE(rom_ids), &rom_found) == lwowOK)
      wait(100);

  //Configure found DS18B20 thermometer
  lwow_ds18x20_start(&ow, rom_ids);
  lwow_ds18x20_set_resolution(&ow, rom_ids, 12);

  //Start Peltier PWM timer.
  MX_TIM4_Init();
  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);

  //Start display and logic timer.
  MX_TIM3_Init();
  HAL_TIM_Base_Start_IT(&htim3);

  //Start cooling!
    data.statusMessage = "";
    data.mode = TEMPERATURE_TARGET;
    toggleFans(1);

  while (1);

}
```

Rysunek 6: Inicjalizacja projektu

Główna pętla

Timer 3 jest odpowiedzialny za sterowanie lodówką. Wykonywany jest odczyt temperatury, aktualizowane są wartości na wyświetlaczu, a na koniec obliczany jest cykl pracy PWM.

```
/**
  * @brief This function handles TIM3 global interrupt.
  */
void TIM3_IRQHandler(void)
{
    //Read temperature
    lwow_ds18x20_read(&ow, rom_ids, &data.measuredTemp);
    lwow_ds18x20_start(&ow, rom_ids);

    //Update display with incremented animation step
    menuDisplay_Update(1);

    //Set output PWM
    handleOutput(&data);

    HAL_TIM_IRQHandler(&htim3);

}
```

Rysunek 7: Główna pętla

Przerwania

Zarejestrowane są dwa przerwania. Pierwszy od zmiany poziomu wyjścia A enkodera, a drugi od obsługi obu przycisków.

Przeskok enkodera podaje na wyjściach AB kod Grey'a, po którym można stwierdzić kierunek obrotu. Debounce uzyskano poprzez odczytywanie stabilnego wyjścia B tylko podczas zmiany A.

Przyciski są połączone do jednego pinu przerwania za pomocą diod. Podczas wywołania przerwania program wchodzi w pętlę blokującą, która czyta piny obu przycisków. Jeżeli żadnego nie odczytano w oknie czasu, przerwanie jest ignorowane.

```
void handleButtons(){
    static uint32_t pressCooldown;
    if(pressCooldown < uwTick){
        pressCooldown = uwTick + 500;
        while(1){
            if(HAL_GPIO_ReadPin(BUTTON_POWER_PORT, BUTTON_POWER_PIN) == GPIO_PIN_RESET){
                toggleMode();
                break;
            }
            if(HAL_GPIO_ReadPin(BUTTON_ENCODER_PORT, BUTTON_ENCODER_PIN) == GPIO_PIN_RESET){
                togglePower();
            }
            if(pressCooldown >= uwTick){
                break;
            }
        }
    }
}

void handleEncoderTurn(){
    static bool lastTurn;
    static bool sameDirTurnFlag;
    static bool a0; // previous A state
    static bool b0; // previous B state
      bool a = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_8);
      bool b = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_15);
      if (a != a0) {              // A changed
        a0 = a;
        if (b != b0) {
          b0 = b;
          if((lastTurn != (a==b) )|| (sameDirTurnFlag == 1)){
              sameDirTurnFlag = 0;
              lastTurn = a==b;
              changeTargetValues(a==b);
          }else{
              sameDirTurnFlag = 1;
          }


        }
      }
}
//Encoder and button callbacks
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{

if(GPIO_Pin == GPIO_PIN_8) // Encoder
    handleEncoderTurn();
if(GPIO_Pin == GPIO_PIN_9)
    handleButtons();
}
```

Rysunek 8: Obsługa przerwań

5 Podsumowanie i uwagi

Projekt zakończył się sukcesem. Mini-lodówka jest w stanie schłodzić 200ml puszkę napoju do temperatury $8°C$.

Powstały jednak problemy, które można by poprawić w następnej wersji:

- Główny wątek nic nie robi, a nie jest zatrzymywany. Po wystartowaniu mikro-kontroler może działać w trybie obsługi przerwań.
- Brakuje watchdoga zapewniającego bezpieczne wyłączenie sprzętu podczas nieoczekiwanego błędu.
- Czujnik temperatury czasem przestaje działać. Możliwe, że trzeba blokować przerwania GPIO podczas operacji wrażliwych na czas.
- Regulator 5V nie jest stworzony do zasilania wentylatora. Pomimo możliwości dostarczenia mocy, komponent jest niewydajny i szybko przegrzewa się do temperatury ok. $90°C$.
- Brak ochrony przed zwarciem. Nie powinno się zostawiać projektu bez nadzoru.
- Nieszczelność komory mocno ogranicza możliwości lodówki. Dodanie uszczelek z TPU i zmiana zawiasów drzwiczek naprawiłaby problem.

Dokumenty, oraz kod są umieszczone na platformie GitHub: