№2 зертханалық жұмыс
Электромобильді басқару жүйелеріндегі таймерлер, үзілістер және PWM

1. Жұмыстың мақсаты

Электромобиль батареясын салқындатудың оқу үлгісіндегі жүйесі негізінде STM32F103C8T6 микроконтроллерінің аппараттық таймерлерінің, үзілістерінің және импульс енін модуляциялаудың (PWM) жұмыс істеу принциптерін зерттеу.

Зертханалық жұмыс барысында білім алушы келесі дағдыларды меңгеруі тиіс:

- аппараттық таймердің қызметін түсіну;
- аппараттық таймер мен delay() функциясының айырмашылығын анықтау;
- периодты оқиғаларды қалыптастыру;
- таймер үзілістерін пайдалану;
- ISR (үзілісті өңдеу функциясы) қызметін түсіну;
- volatile кілт сөзімен жарияланған айнымалыларды қолдану;
- бірнеше периодты тапсырманың орындалуын ұйымдастыру;
- PWM жұмыс істеу принципін түсіну;
- Duty Cycle (импульстің толтыру коэффициенті) мәнін өзгерту;
- PWM арқылы атқарушы құрылғыны басқару;
- блоктамайтын бағдарламаны (Non-blocking Program) құру;
- батарея температурасы мен салқындату қарқындылығы арасындағы байланысты жүзеге асыру.

Жұмыстың негізгі логикасы

Температура датчигі → ADC → STM32 микроконтроллері → Температураны талдау → Салқындатуды басқару алгоритмі → PWM сигналын қалыптастыру → Желдеткіш драйвері → Салқындату желдеткіші.

Бұл ретте аппараттық таймер мен үзілістер температураны периодты түрде өлшеуді және басқару тапсырмаларының уақытылы орындалуын ұйымдастыру үшін қолданылады.



2. Нақты электромобильмен байланысы

Электромобильде келесі компоненттердің температуралық режимін бақылау қажет:
- жоғары кернеулі батареяның;
- инвертордың;
- электр қозғалтқышының;
- борттық зарядтау құрылғысының (OBC);
- DC/DC түрлендіргішінің.
Жылуды басқару жүйесінде (Thermal Management System) келесі құрылғылар қолданылуы мүмкін:



Желдеткіштерді, сорғыларды және клапандарды басқару жүйелері электромобильдердің жылуды басқару жүйесінің (Thermal Management System) нақты құрамдас бөліктері болып табылады.

Мысалы, Texas Instruments (TI) компаниясы автомобильдердің жылуды басқару жүйесінде келесі ішкі жүйелерді жеке қарастырады:
- Желдеткіштерді басқару (Automotive Blower/Fan);
- Салқындатқыш сұйықтық сорғысын басқару (Coolant Pump);
- Салқындатқыш сұйықтық клапандарын басқару (Coolant/Water Valve).

Біздің зертханалық жұмыста қолданылатын жеңілдетілген модель
Потенциометр → Температура датчигі (Temperature Sensor)
STM32 → Электрондық басқару блогы (ECU/Controller)
LED (жарықдиод)→ Салқындату желдеткіші (Cooling Fan)

Маңызды: Нақты электромобильде қуатты салқындату желдеткіші микроконтроллердің GPIO шығысына тікелей қосылмайды.

Нақты жүйенің құрылымы
MCU → PWM → Драйвер / MOSFET → Электрлік желдеткіш

Wokwi симуляторында LED тек PWM сигналының өзгерісін көрнекі түрде көрсету үшін қолданылады.

3. Қолданылатын бағдарламалық орта

Зертханалық жұмыста тек келесі құралдар қолданылады:
- Wokwi — электрондық схемаларды модельдеу ортасы;
- STM32F103C8T6 Blue Pill — микроконтроллер;
- Wokwi код редакторы — бағдарламаны жазу және өзгерту үшін;
- Serial Monitor — бағдарламаның жұмыс нәтижелерін бақылау үшін;
- Logic Analyzer — цифрлық сигналдар мен PWM импульстерін талдау үшін.
VS Code және PlatformIO бағдарламаларын орнату талап етілмейді.

№1 зертханалық жұмыста қолданылған STM32F103C8 негізіндегі Wokwi жобасын пайдаланыңыз. Бұл жоба әдістемелік нұсқаулықта оқытушы ұсынған бастапқы жоба ретінде көрсетілген.

Бастапқы жоба:
https://wokwi.com/projects/476622661513474049

Жобаның көшірмесін жасап, оны келесідей атаңыз:
Lab_02_Timer_PWM_СіздіңАтыңыз

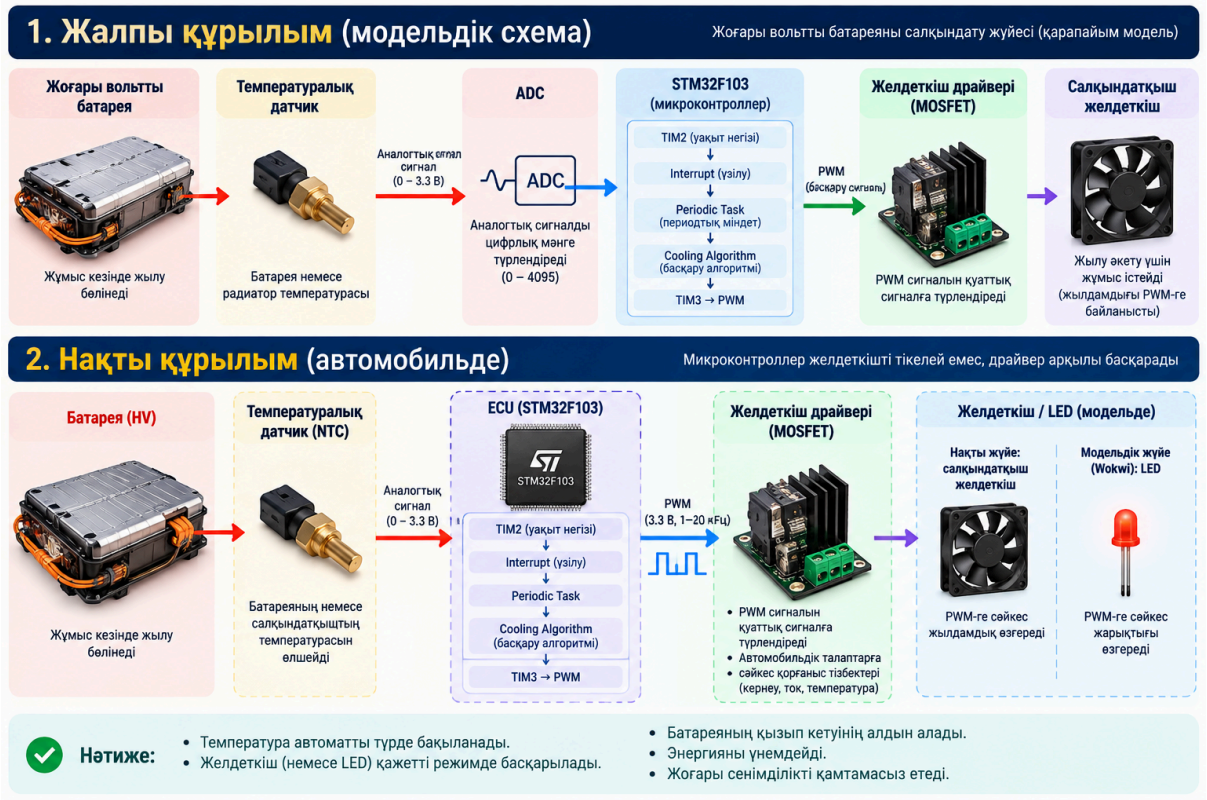
4. Қажетті компоненттер
Жобаға келесі компоненттерді қосыңыз:

Компонент	**Саны**	**Мақсаты**
STM32F103C8T6 Blue Pill	1	микроконтроллер
Potentiometer	1	Температура датчигі
LED	1	Шартты салқындату желдеткіші (Cooling Fan)
LED	1	STATUS-индикатор
Резистор 220–330 Ω	2	LED тогын шектеу
Logic Analyzer	1	PWM сигналын зерттеу

4. Зерттеу нысаны

Электромобильдің жоғары кернеулі батареясын салқындату жүйесінің жеңілдетілген моделі қарастырылады.

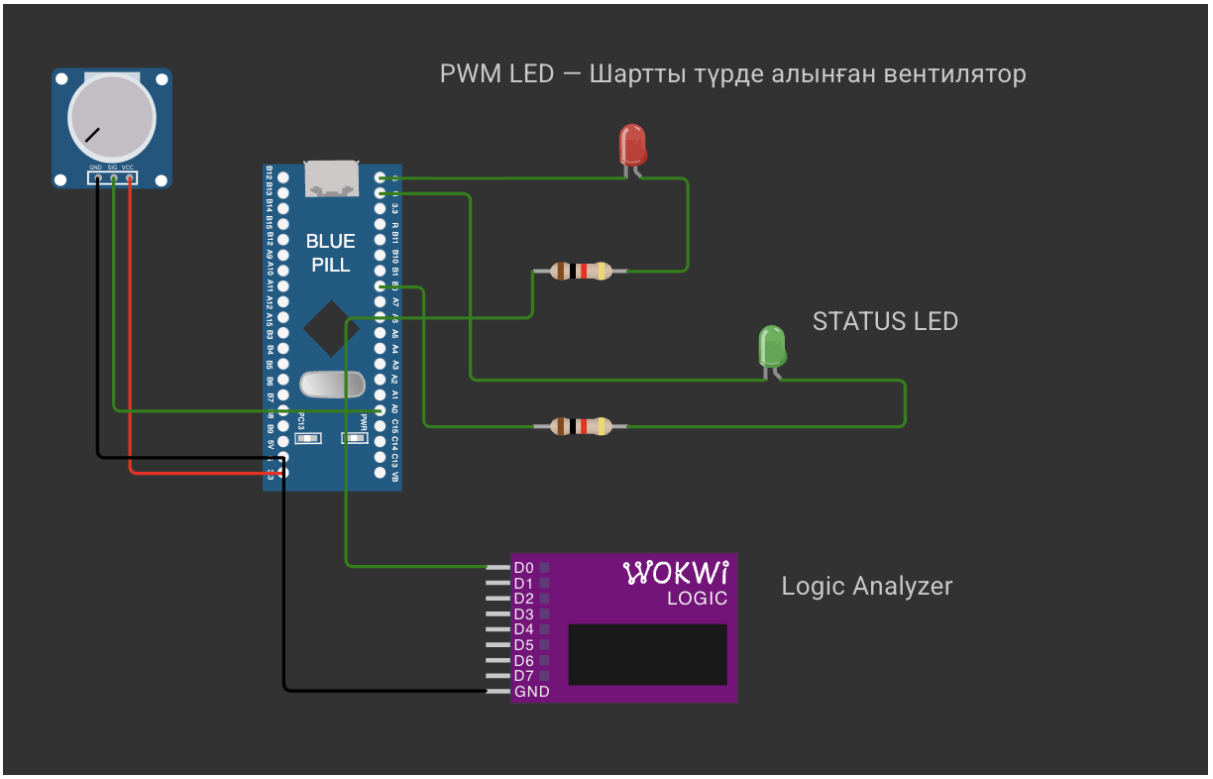
3-сурет. Жүйенің жалпы құрылымы



5. Wokwi жүйесіндегі қосылу сұлбасы

Келесі STM32 микроконтроллерінің шығыстары қолданылады:

Элемент	**STM32 шығысы**
Температура датчигі	PA0
Желдеткіштің PWM сигналы	PA6
STATUS жарықдиоды (LED)	PB0
Logic Analyzer D0 арнасы	PA6



Неліктен delay() функциясын қолдану жеткіліксіз?

Бірінші зертханалық жұмыста delay(500) команда қолданылды Қарапайым бағдарлама үшін бұл ыңғайлы.

Мысалы:
Serial.println("Temperature");
delay(3000);
Алайда delay(3000) командасы loop() функциясының әрі қарай орындалуын шамамен үш секундқа кідіртеді.
Температураны оқу
↓
3 секунд күту
↓
Жүйені тексеру
↓
3 секунд күту
...
...
...

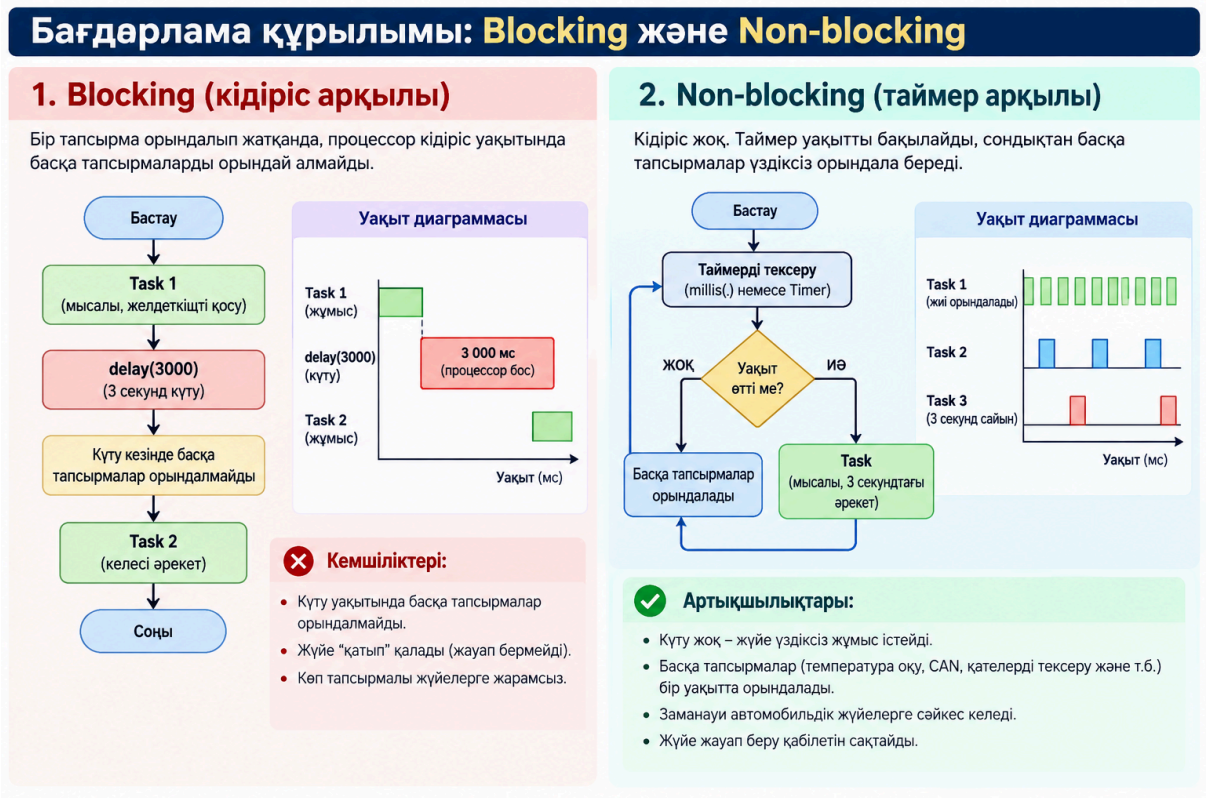
<u>Автомобиль контроллерінің жұмысын қарастырайық.</u>

Ол бір уақытта келесі тапсырмаларды орындауы тиіс:

Температураны оқу
+
Акселератор педалінің күйін оқу
+
CAN деректерін өңдеу
+
Қателерді тексеру
+
Салқындату жүйесін басқару
+
Атқарушы құрылғыларды басқару

Сондықтан ұзақ уақыт күту тиімсіз.

Сондықтан келесі кезеңде блоктаушы бағдарламадан (blocking program) блоктамайтын бағдарламаға (non-blocking program) көшеміз.

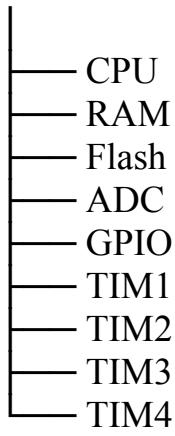


Аппараттық таймер

Аппараттық таймер (Hardware Timer) — микроконтроллердің ішінде орналасқан, негізгі бағдарламадан тәуелсіз уақытты автоматты түрде есептейтін арнайы электрондық модуль.

Мысалы, STM32F103 микроконтроллерінің ішінде бірнеше таймер (TIM) бар:

STM32F103C8T6



Олардың әрқайсысы жеке аппараттық блок болып табылады.

TIM1 және TIM3 — STM32 микроконтроллерінің ішіндегі аппараттық таймерлер.

Яғни, олар жеке сыртқы компоненттер емес. Олар STM32F103C8T6 микроконтроллерінің ішінде орналасқан.

TIM — Timer (таймер) дегенді білдіреді.

Мысалы, TIM1, TIM2, TIM3, TIM4 — STM32 микроконтроллерінің ішіндегі әртүрлі аппараттық таймер блоктары.

Олардың негізгі міндеті — уақытты есептеу және оқиғаларды қалыптастыру.

Мысалы, таймер келесі әрекеттерді орындай алады:

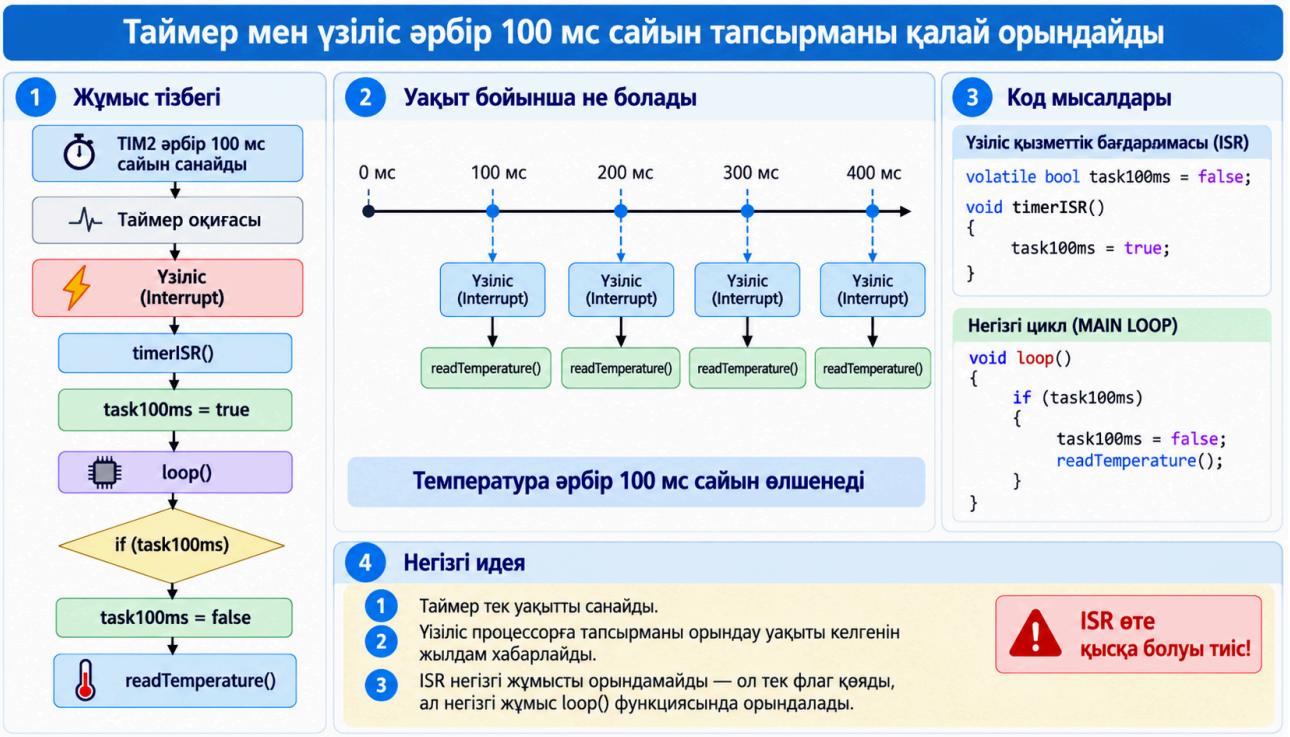
- уақытты есептеу;
- әрбір 100 мс сайын үзіліс шақыру;
- PWM сигналын қалыптастыру;
- импульстің ұзақтығын өлшеу;
- Capture/Compare (ұстап алу/салыстыру) режимдерінде жұмыс істеу.

Біздің зертханалық жұмыста келесі таймерлер қолданылады:

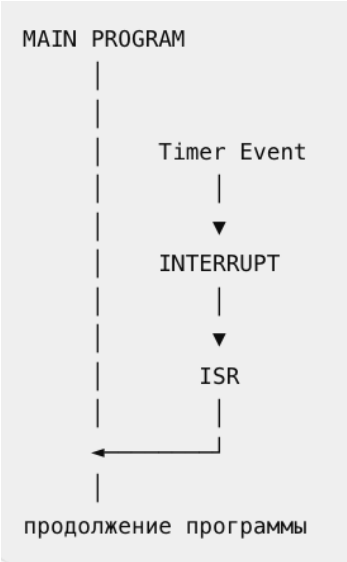
TIM2
↓
Әрбір 100 мс сайын
периодты үзіліс шақыру
Және:

TIM3
↓
PWM сигналын қалыптастыру
↓
PA6
↓
Шартты желдеткіш

Олар бір уақытта қатар жұмыс істей алады.
Мысалы, TIM2 әрбір 100 мс сайын үзіліс шақырып тұрған кезде, TIM3 бір уақытта үздіксіз PWM сигналын қалыптастырады.
Бұл — микроконтроллердің негізгі артықшылықтарының бірі.



Үзіліс дегеніміз не?
Interrupt — үзіліс — микроконтроллерге белгілі бір оқиғаға жауап беруге мүмкіндік беретін механизм.
Мысалы:



Interrupt Service Routine
ISR — Interrupt Service Routine — үзілісті өңдеу функциясы.

Мысалы:

```
volatile bool task100ms = false;

void timerISR()
{
    task100ms = true;
}
```

```
 void loop()
 {
     if (task100ms)
     {
         task100ms = false;

         // выполнение задачи
     }
 }
```

volatile нені білдіреді?
Айнымалы: volatile bool task100ms;
volatile деп жарияланады, себебі оның мәні бағдарламаның әдеттегі орындалу ретінен тыс, мысалы, ISR ішінде өзгеруі мүмкін.

Яғни компиляторға бұл айнымалының мәні негізгі бағдарлама үшін күтпеген уақытта өзгеруі мүмкін екені хабарланады.

1-тапсырма. delay() функциясын тексеру

Қарапайым бағдарлама құрыңыз:

```
void setup()
{
   Serial.begin(115200);
}

void loop()
{
   Serial.println("Task executed");

   delay(3000);
}
```

Симуляцияны іске қосыңыз.
delay(3000); мәнін келесі мәндерге өзгертіңіз: 1000, 5000.
Жауап беріңіз:

delay() мәнінің өзгеруі бағдарламаның орындалу периодына қалай әсер етеді?

2-тапсырма. TIM2 аппараттық таймерін құру

Алдыңғы бағдарламаны жойыңыз.

Келесі кодты енгізіңіз:

```
// ========================================
// 1. TIM2 таймерін құру
// ========================================

HardwareTimer *ControlTimer =
   new HardwareTimer(TIM2);


// ========================================
```

```
// 2. 100 ms тапсырмасының жалаушасы
// ========================================

volatile bool task100ms = false;


// ========================================
// 3. Таймер үзілісінің функциясы
// ========================================

void timerISR()
{
   task100ms = true;
}


// ========================================
// 4. Бастапқы баптаулар
// ========================================

void setup()
{
   Serial.begin(115200);


   // ======================================
   // 5. TIM2 периодын 100 ms етіп орнату
   // ======================================

   ControlTimer->setOverflow(
      100000,
      MICROSEC_FORMAT
   );


   // ======================================
   // 6. Үзіліс функциясын TIM2-ге байланыстыру
   // ======================================

   ControlTimer->attachInterrupt(
      timerISR
   );
```

```
    // ======================================
    // 7. TIM2 таймерін іске қосу
    // ======================================

    ControlTimer->resume();
}


// ==========================================
// 8. Негізгі программа циклі
// ==========================================

void loop()
{
    // ======================================
    // 9. 100 ms өткенін тексеру
    // ======================================

    if (task100ms)
    {
        task100ms = false;


        // ==================================
        // 10. 100 ms сайын орындалатын тапсырма
        // ==================================

        Serial.println(
            "100 ms task"
        );
    }
}
```

100000 нені білдіреді?
Бағдарламада:
100000,
MICROSEC_FORMAT
мынаны білдіреді:
100000 мкс
Себебі:
1000 мкс = 1 мс

Сондықтан:
100000 мкс = 100 мс

Таймерді тексеру

Келесі мәндерді кезекпен орнатыңыз:

Период	**setOverflow мәні**
100 мс	100000 мкс
250 мс	250000 мкс
500 мс	500000 мкс
1000 мс	1000000 мкс

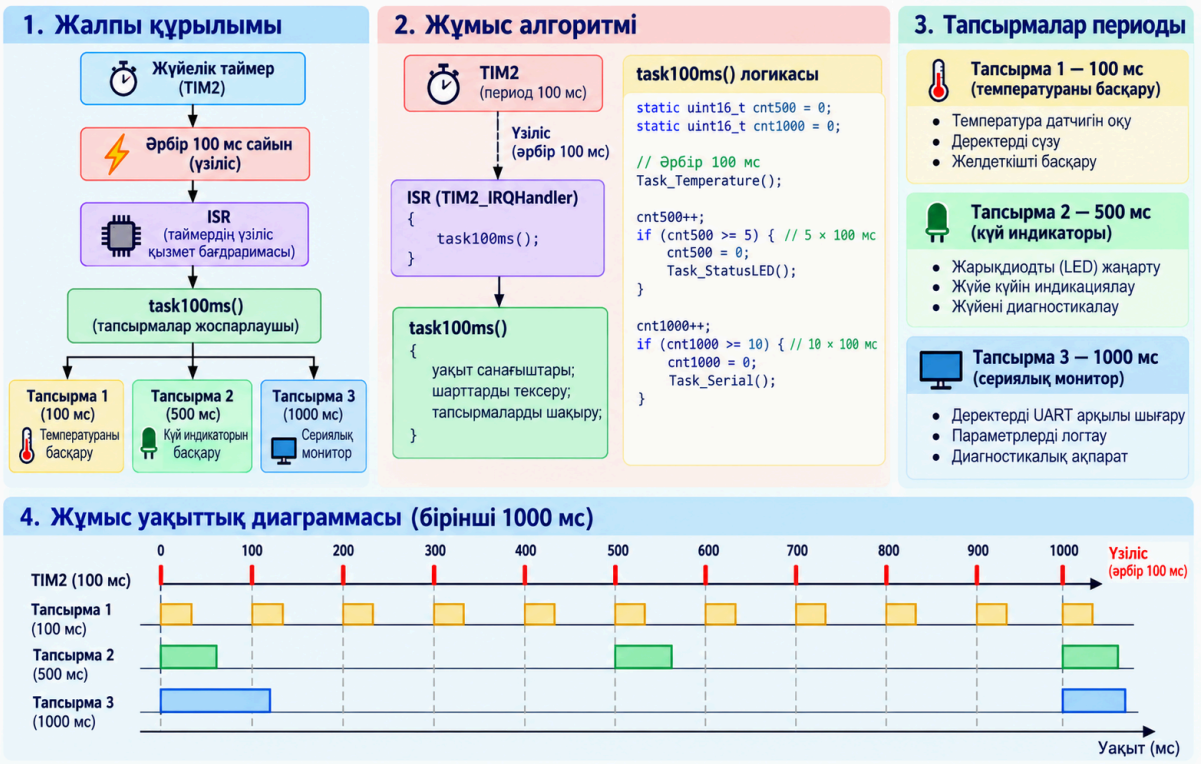
Serial Monitor терезесін бақылаңыз.

Жауап беріңіз:

Хабарламалардың пайда болу жиілігі қалай өзгереді?

3-тапсырма. Бірнеше периодтық тапсырмалар

Нақты ендірілген контроллерде әртүрлі тапсырмалардың орындалу периодтары әртүрлі болуы мүмкін.
Мысалы:
100 мс → өлшеу және басқару
500 мс → жүйенің белсенділігін бақылау
1000 мс → диагностикалық ақпаратты шығару
Біз үш аппараттық таймер құрмаймыз.
Бір негізгі таймерді қолданамыз: TIM2 = 100 мс



Бірнеше тапсырманың коды

Келесі кодты енгізіңіз:

```
// ============================================
// 1. STATUS LED шығысын анықтау
// ============================================

#define STATUS_LED PB0
```

```
// ==========================================
// 2. TIM2 аппараттық таймерін құру
// ==========================================

HardwareTimer *ControlTimer =
    new HardwareTimer(TIM2);


// ==========================================
// 3. 100 ms тапсырмасының жалаушасы
// ==========================================

volatile bool task100ms = false;


// ==========================================
// 4. 500 ms және 1000 ms есептегіштері
// ==========================================

int counter500 = 0;
int counter1000 = 0;


// ==========================================
// 5. STATUS LED күйі
// ==========================================

bool ledState = false;


// ==========================================
// 6. Таймер үзілісінің функциясы
// ==========================================

void timerISR()
{
    task100ms = true;
}


// ==========================================
```

```
// 7. Бастапқы баптаулар
// ========================================

void setup()
{
  Serial.begin(115200);

  pinMode(STATUS_LED, OUTPUT);


  // ========================================
  // 8. TIM2 периодын 100 ms етіп орнату
  // ========================================

  ControlTimer->setOverflow(
    100000,
    MICROSEC_FORMAT
  );


  // ========================================
  // 9. Үзіліс функциясын TIM2-ге байланыстыру
  // ========================================

  ControlTimer->attachInterrupt(
    timerISR
  );


  // ========================================
  // 10. TIM2 таймерін іске қосу
  // ========================================

  ControlTimer->resume();
}


// ========================================
// 11. Негізгі программа циклі
// ========================================

void loop()
{
```

```
// ====================================
// 12. 100 ms өткенін тексеру
// ====================================

if (task100ms)
{
   task100ms = false;


   // ====================================
   // 13. 100 ms сайын орындалатын тапсырма
   // ====================================

   Serial.println("100 ms task");


   // ====================================
   // 14. Уақыт есептегіштерін арттыру
   // ====================================

   counter500++;
   counter1000++;


   // ====================================
   // 15. 500 ms сайын STATUS LED күйін өзгерту
   // ====================================

   if (counter500 >= 5)
   {
      counter500 = 0;

      ledState = !ledState;

      digitalWrite(
         STATUS_LED,
         ledState
      );
   }


   // ====================================
   // 16. 1000 ms сайын диагностикалық хабарлама шығару
```

```
        // ====================================

        if (counter1000 >= 10)
        {
            counter1000 = 0;

            Serial.println(
                "1 second status task"
            );
        }
    }
}
```

Нені байқау қажет?

Бір уақытта үш процесс орындалуы тиіс:
Әрбір 100 мс сайын
→ Control Task
Әрбір 500 мс сайын
→ STATUS LED күйін өзгертеді
Әрбір 1000 мс сайын
→ диагностикалық хабарлама шығарылады

Негізгі идея:

Микроконтроллер ұзақ delay() функцияларын қолданбай, бірнеше периодтық тапсырманы орындайды.

4-тапсырма. Температура датчигін қосу

```
// ================================================
// ТАЙМЕР + ҮЗІЛІС + ТЕМПЕРАТУРА ДАТЧИГІ
// STM32F103C8T6
// ================================================


// ================================================
// ШЫҒЫСТАРДЫ АНЫҚТАУ
// ================================================

#define TEMP_PIN    PA0
#define STATUS_LED  PB0


// ================================================
// АППАРАТТЫҚ ТАЙМЕР
// ================================================

// TIM2 периодтық тапсырманы орындау үшін қолданылады
HardwareTimer *ControlTimer =
    new HardwareTimer(TIM2);


// ================================================
// АЙНЫМАЛЫЛАР
// ================================================

// 100 ms тапсырмасының жалаушасы
volatile bool task100ms = false;

// Бағдарламалық уақыт есептегіштері
int counter500 = 0;
int counter1000 = 0;

// STATUS LED күйі
bool ledState = false;

// Батарея температурасы
float temperature = 0.0;

// ADC мәні
```

```
int adcTemperature = 0;


// ================================================
// ТАЙМЕР ҮЗІЛІСІ
// ================================================

void timerISR()
{
   // ISR қысқа болуы керек
   task100ms = true;
}


// ================================================
// ТЕМПЕРАТУРАНЫ ОҚУ
// ================================================

void readTemperature()
{
   // 12-биттік ADC мәнін оқу
   adcTemperature =
      analogRead(TEMP_PIN);


   // Оқу мақсатындағы шкала:
   // ADC = 0    → 0°C
   // ADC = 4095 → 100°C

   temperature =
      adcTemperature
      * 100.0
      / 4095.0;
}


// ================================================
// БАСТАПҚЫ БАПТАУЛАР
// ================================================

void setup()
{
   // Serial Monitor іске қосу
```

```
    Serial.begin(115200);


    // STM32F103 ADC ажыратымдылығын
    // 12 бит етіп орнату
    analogReadResolution(12);


    // STATUS LED шығыс ретінде баптау
    pinMode(
        STATUS_LED,
        OUTPUT
    );


    // ============================================
    // TIM2
    // Период = 100 ms
    // ============================================

    ControlTimer->setOverflow(
        100000,
        MICROSEC_FORMAT
    );


    // Үзіліс функциясын TIM2-ге байланыстыру
    ControlTimer->attachInterrupt(
        timerISR
    );


    // TIM2 таймерін іске қосу
    ControlTimer->resume();
}


// ================================================
// НЕГІЗГІ ПРОГРАММА ЦИКЛІ
// ================================================

void loop()
{
```

```
// TIM2 оқиғасынан кейін орындалады
if (task100ms)
{
   // Жалаушаны қайта түсіру
   task100ms = false;


   // ======================================
   // 100 ms САЙЫН ОРЫНДАЛАТЫН ТАПСЫРМА
   // ======================================

   // Температураны оқу
   readTemperature();


   // Уақыт есептегіштерін арттыру
   counter500++;
   counter1000++;


   // ======================================
   // 500 ms САЙЫН ОРЫНДАЛАТЫН ТАПСЫРМА
   // STATUS LED
   // ======================================

   if (counter500 >= 5)
   {
      counter500 = 0;

      // LED күйін кері өзгерту
      ledState =
         !ledState;

      digitalWrite(
         STATUS_LED,
         ledState
      );
   }


   // ======================================
   // 1000 ms САЙЫН ОРЫНДАЛАТЫН ТАПСЫРМА
   // SERIAL MONITOR
```

```
// ========================================

if (counter1000 >= 10)
{
   counter1000 = 0;


   Serial.println(
      "================================="
   );

   Serial.println(
      "EV БАТАРЕЯ ТЕМПЕРАТУРАСЫН БАҚЫЛАУ"
   );

   Serial.println();


   Serial.print(
      "ADC мәні = "
   );

   Serial.println(
      adcTemperature
   );


   Serial.print(
      "Батарея температурасы = "
   );

   Serial.print(
      temperature,
      1
   );

   Serial.println(
      " C"
   );


   Serial.println(
      "Басқару тапсырмасы = 100 ms"
```

```
            );

            Serial.println(
                "STATUS тапсырмасы = 500 ms"
            );

            Serial.println(
                "Диагностикалық тапсырма = 1000 ms"
            );


            Serial.println(
                "================================="
            );

            Serial.println();
        }
    }
}
```

wokwi-stm32-blue-pill.ino ● diagram.json ●

Library Manager

```
{
    {
        {
            Serial.println(
                "============================="
            );

            Serial.println(
                "EV TEMPERATURE MONITOR"
            );

            Serial.println();


            Serial.print(
                "ADC = "
            );

            Serial.println(
                adcTemperature
            );


            Serial.print(
                "Battery Temperature = "
            );

            Serial.print(
                temperature,
                1
            );

            Serial.println(
                " C"
            );


            Serial.println(
                "Control Task = 100 ms"
            );

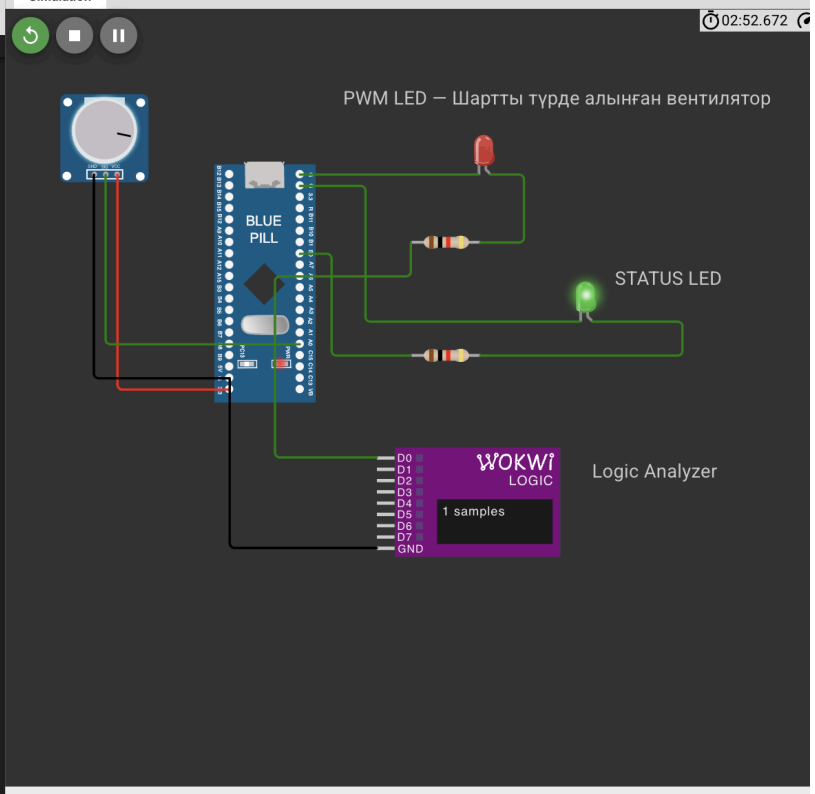
            Serial.println(
                "Status Task = 500 ms"
            );

            Serial.println(
                "Diagnostic Task = 1000 ms"
            );


            Serial.println(
                "============================="
            );

            Serial.println();
        }
    }
}
```

Simulation




```
ADC = 3595
Battery Temperature = 87.8 C
Control Task = 100 ms
Status Task = 500 ms
Diagnostic Task = 1000 ms
=============================

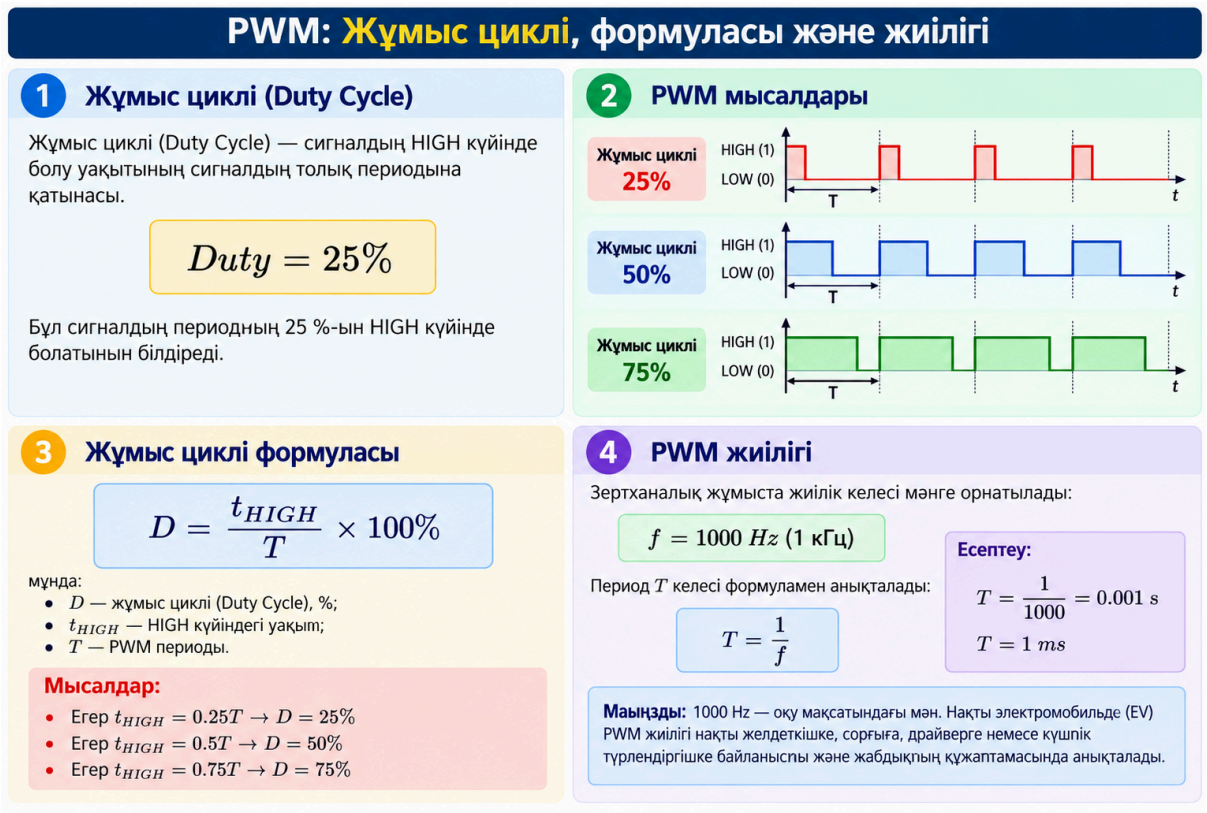
=============================
EV TEMPERATURE MONITOR

ADC = 3595
Battery Temperature = 87.8 C
Control Task = 100 ms
Status Task = 500 ms
Diagnostic Task = 1000 ms
=============================
```

5-тапсырма. TIM3 және Logic Analyzer арқылы PWM құру

PWM дегеніміз не?
PWM — Pulse Width Modulation — импульс енін модуляциялау.
PWM — белгілі бір жиілігі бар, HIGH және LOW күйлері арасында өте жылдам ауысатын цифрлық сигнал.
Негізгі өзгеретін параметр: Duty Cycle — толтыру коэффициенті.
Duty Cycle — сигналдың периодтың қанша пайызында HIGH күйінде болатынын көрсетеді.

Мысалы: Егер Duty = 25% болса, сигнал периодтың 25%-ында HIGH күйінде болады.



Зертханалық жұмыста қолданамыз:

TIM3 → PA6 → PWM LED

Яғни TIM3 аппараттық таймері PWM сигналын қалыптастырады, ал сигнал PA6 шығысы арқылы беріледі.

Код:

```
#define FAN_PWM_PIN PA6

HardwareTimer *PwmTimer =
   new HardwareTimer(TIM3);

void setup()
{
   PwmTimer->setPWM(
      1,
      FAN_PWM_PIN,
      1000,
      50
   );
}

void loop()
{

}
```

Мұнда:
1 — таймер арнасы TIM3 Channel 1.
FAN_PWM_PIN — PA6 шығысы.
1000 — PWM жиілігі: f = 1000 Гц.
50 — толтыру коэффициенті (Duty Cycle): D = 50%.

Яғни бұл команда мынаны білдіреді:
TIM3 таймері PA6 шығысында жиілігі 1000 Гц және толтыру коэффициенті 50% болатын PWM сигналын қалыптастыруы тиіс.

Duty Cycle мәнінің жарықдиодқа әсері

Duty Cycle	**LED-тің күтілетін жұмысы**
0%	Жарықдиод өшірулі
25%	Төмен жарықтық
50%	Орташа жарықтық

75%	Жоғары жарықтық
100%	Максималды жарықтық

Duty Cycle мәні артқан сайын бір период ішінде жарықдиодтың қосулы күйде болатын орташа уақыты артады.

PWM жиілігі жеткілікті жоғары болған кезде адам көзі жарықдиодтың жеке қосылуы мен өшірілуін әдетте ажырата алмайды, оның орнына жарықтықтың өзгеруін қабылдайды.

Маңызды: берілген кесте жарықдиод дұрыс қосылған және PWM арқылы басқарылған жағдайдағы күтілетін жұмысын көрсетеді. Қолданылатын Wokwi ортасында жарықтықтың визуалды өзгеруі шектеулі түрде көрсетілуі немесе мүлдем көрсетілмеуі мүмкін. Сондықтан PWM сигналын тексеру үшін Logic Analyzer құралын пайдаланып, импульстердің ұзақтығын талдау ұсынылады.
PWM технологиясының электромобильдерде қолданылуы
Тапсырманың негізгі мақсаты — Duty Cycle мәнін өзгерту арқылы басқару сигналының атқарушы құрылғыға орташа әсерін реттеуге болатынын түсіну.
Электромобильдерде PWM келесі мақсаттарда қолданылуы мүмкін:

1. Салқындату желдеткішін басқару. Батареяның немесе күштік электрониканың температурасы жоғарылаған кезде контроллер желдеткіш жылдамдығын арттыру командасын бере алады.
2. Салқындату жүйесінің сорғысын басқару. Үйлесімді жүйелерде басқару сигналын өзгерту сорғының өнімділігін реттеуге мүмкіндік береді.
3. Электромагниттік клапандарды басқару. PWM катушка тогын және тиісті конструкцияларда клапанның орналасуын реттеу үшін қолданылуы мүмкін.
4. Күштік электрониканы басқару. PWM электромобиль инверторларында электр қозғалтқышын басқаруды қамтамасыз ететін күштік кілттердің басқару сигналдарын қалыптастыру үшін қолданылады.

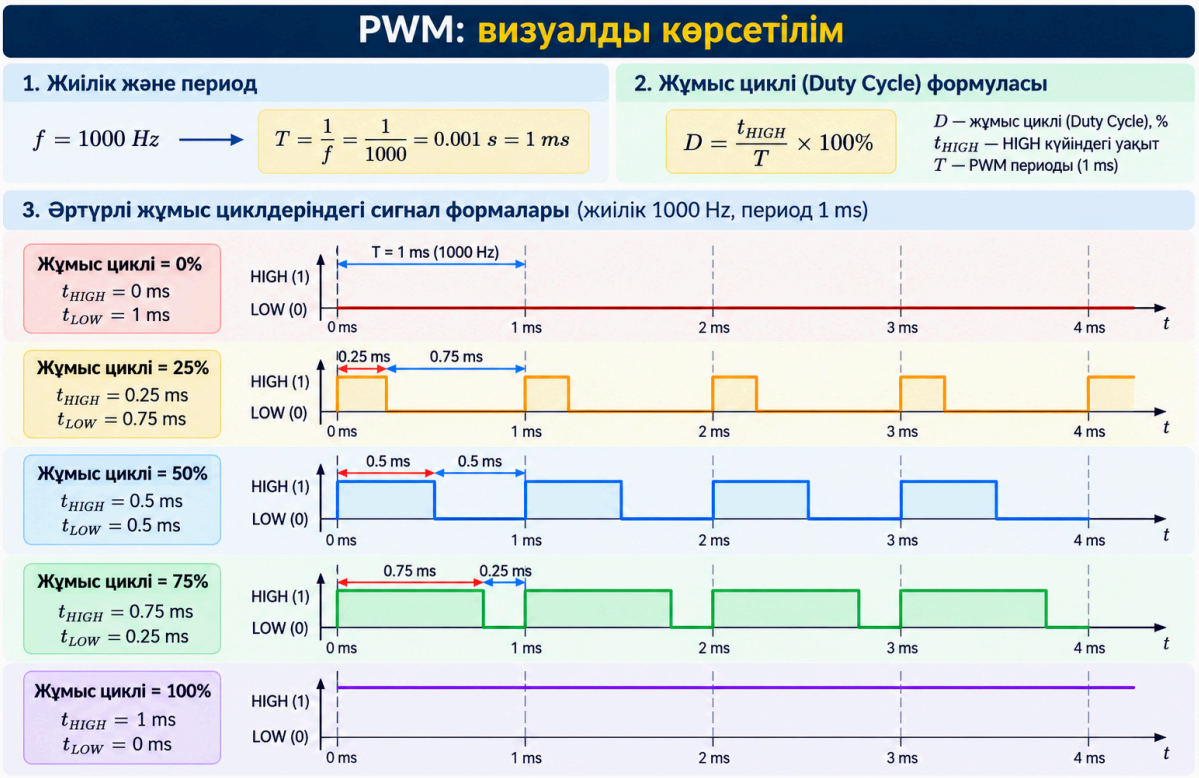
Алайда Duty Cycle мен нақты жылдамдық, өнімділік немесе қуат арасындағы тәуелділік әрқашан сызықтық бола бермейді. Ол атқарушы құрылғының конструкциясына, драйверге және басқару алгоритміне байланысты.
Duty Cycle мәнін өзгерте отырып, студент PWM сигналының жоғары логикалық деңгейінің ұзақтығы қалай өзгеретінін бақылайды.

Жарықдиод мысалында импульс енін модуляциялау арқылы басқарудың негізгі принципі зерттеледі. Бұл принцип электромобильдердің салқындату жүйелерінде, электр жетектерінде және күштік электроникасында да қолданылады.
Нақты электромобильде микроконтроллер қуатты атқарушы құрылғыларды әдетте GPIO шығысы арқылы тікелей емес, тиісті драйверлер арқылы басқарады.

Біздің модельде: PWM → LED
Ал нақты жүйеде жұмыс принципі келесідей:
PWM ↓ Драйвер ↓ Салқындату желдеткіші / Сорғы

Logic Analyzer арқылы PWM сигналын зерттеу

Logic Analyzer PWM сигналының «төмендетілген кернеу» емес, цифрлық импульстер тізбегі екенін көруге мүмкіндік береді.



Жалпы мақсатымыз:

PWM сигналының пішінін көру және Frequency мен Duty Cycle арасындағы айырмашылықты түсіну.

Студенттер үшін ең маңызды тұжырым:

Frequency PWM сигналының қаншалықты жиі қайталанатынын көрсетеді, ал Duty Cycle әр периодтың қандай бөлігінде сигнал HIGH күйінде болатынын көрсетеді.

Біздің жұмыста жиілік әзірге тұрақты болып қалады: 1000 Гц.
Ал тек Duty Cycle мәнін өзгертеміз:
0% → 25% → 50% → 75% → 100%

Осыдан кейін келесі тапсырмада PWM тұрақты болмайды, оның мәні температураға байланысты автоматты түрде өзгеретін болады:
Температура
↓
Басқару алгоритмі
↓
PWM толтыру коэффициенті (Duty Cycle)
↓
Салқындату желдеткіші

Яғни бұл тапсырма студенттің PWM технологиясын автоматты салқындатуды басқаруда қолданбас бұрын, алдымен оның жұмыс принципін жеке түсінуі үшін қажет.

Кодты іске қосыңыз.

```
#define FAN_PWM_PIN PA6

HardwareTimer *PwmTimer =
  new HardwareTimer(TIM3);

void setup()
{
  PwmTimer->setPWM(
    1,
    FAN_PWM_PIN,
    1000,
    50
  );
}
```

```
void loop()
{

}
```

Duty Cycle мәнін өзгертіңіз:
0%
25%
50%
75%
100%

Logic Analyzer арқылы алынған wokwi-logic( ).vcd форматындағы 5 файлды сақтаңыз!
Осциллограмманы көру үшін Wokwi бұл файлды PulseView немесе GTKWave бағдарламасында ашуды ұсынады.

Алайда бұл файлдарды ChatGPT-ке жүктеп, сигналдарды кесте түрінде талдауға және алынған сигналдардың графиктерінің скриншоттарын жасауға болады!

Кесте мен графиктің мысалы!

Бес PWM сигналын салыстыру

Сигнал №	**Жиілік**	**Бір периодтағы HIGH ұзақтығы**	**Толтыру коэффициенті (Duty Cycle)**	**Сигнал түрі**
1	—	0 мс	0%	Тұрақты LOW
2	1000 Гц	0,25 мс	25%	PWM
3				
4				

5	—	Үздіксіз	100%	Тұрақты HIGH

Сигналдарды көрнекі салыстыру



6-тапсырма. Батареяны салқындатуды басқару

Енді келесі компоненттерді біріктіреміз:
Temperature Sensor
+
ADC
+
Timer
+
Interrupt
+
PWM

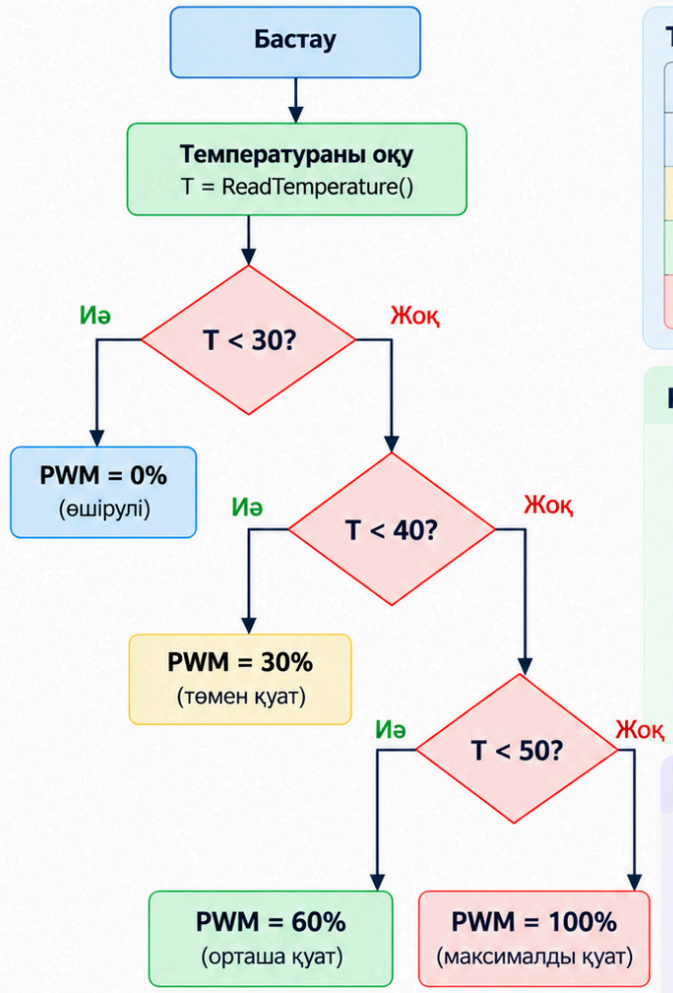
Алгоритм:

Температура	**Fan PWM**	**Режим**
T < 30°C	0%	OFF
30°C ≤ T < 40°C	30%	LOW
40°C ≤ T < 50°C	60%	MEDIUM
T ≥ 50°C	100%	MAX

Бұл шекті мәндер нақты сериялық батарея үшін белгіленген нормативтік мәндер емес, оқу мақсатында қолданылатын мәндер болып табылады.

Басқару алгоритмі

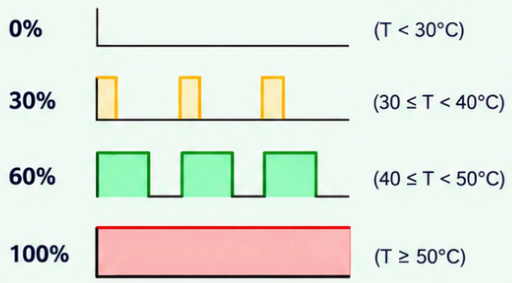
Температураға байланысты PWM басқару алгоритмі



Температура бойынша шекті мәндер

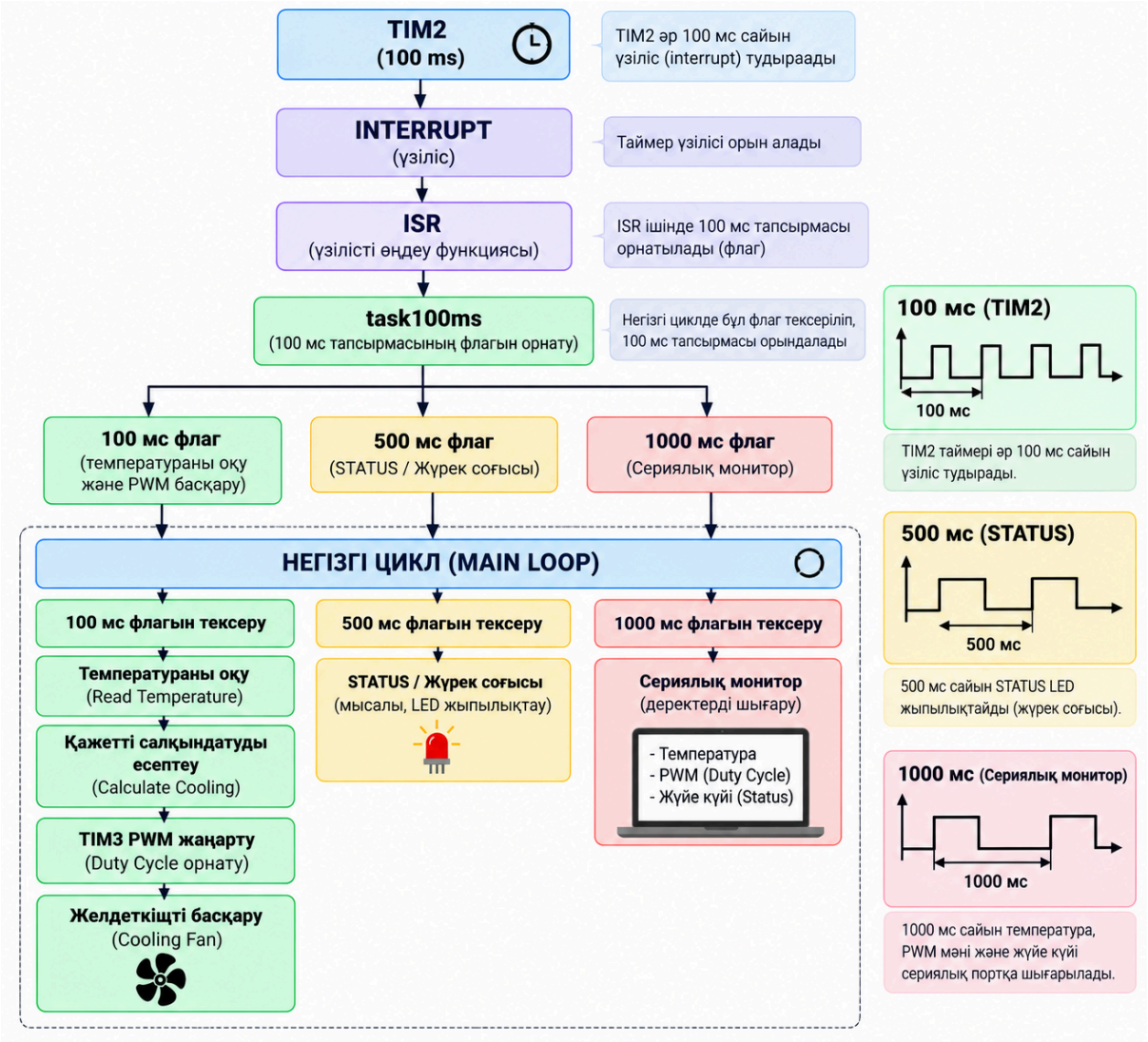
Температура диапазоны, °C	Duty Cycle (PWM)
T < 30	0%
30 ≤ T < 40	30%
40 ≤ T < 50	60%
T ≥ 50	100%

PWM сигналдарының мысалдары



Түсіндірме

- Ағымдағы температура (T) оқылады.
- Температура диапазонына байланысты тиісті Duty Cycle мәні орнатылады.
- PWM атқарушы құрылғыны басқарады (мысалы, салқындату желдеткіші).
- Температура артқан сайын атқарушы құрылғыға әсер (қуат) өседі.
- PWM жиілігі тұрақты болып қалады (мысалы, 1000 Гц).



Толық қорытынды бағдарлама

```
#define TEMP_PIN     PA0
#define FAN_PWM_PIN  PA6
#define STATUS_LED   PB0


// ========================================
// ТАЙМЕРЛЕР
// ========================================

HardwareTimer *ControlTimer =
    new HardwareTimer(TIM2);
```

```
HardwareTimer *PwmTimer =
   new HardwareTimer(TIM3);


// ========================================
// АЙНЫМАЛЫЛАР
// ========================================

volatile bool task100ms = false;

int counter500 = 0;
int counter1000 = 0;

float temperature = 0.0;

uint8_t fanDuty = 0;

bool statusLedState = false;

const char *coolingState = "OFF";


// ========================================
// ҮЗІЛІС ФУНКЦИЯСЫ — ISR
// ========================================

void timerISR()
{
   task100ms = true;
}


// ========================================
// ТЕМПЕРАТУРАНЫ ОҚУ
// ========================================

void readTemperature()
{
   int adcTemperature =
      analogRead(TEMP_PIN);

   // ADC мәнін 0–100°C диапазонына түрлендіру
   temperature =
```

```
        adcTemperature * 100.0 / 4095.0;
}


// ========================================
// САЛҚЫНДАТУДЫ БАСҚАРУ
// ========================================

void controlCooling()
{
   // Температура 30°C-тан төмен
   if (temperature < 30.0)
   {
      fanDuty = 0;
      coolingState = "ӨШІРУЛІ";
   }


   // Температура 30–40°C
   else if (temperature < 40.0)
   {
      fanDuty = 30;
      coolingState = "ТӨМЕН";
   }


   // Температура 40–50°C
   else if (temperature < 50.0)
   {
      fanDuty = 60;
      coolingState = "ОРТАША";
   }


   // Температура 50°C немесе одан жоғары
   else
   {
      fanDuty = 100;
      coolingState = "МАКСИМАЛДЫ";
   }


   // Есептелген PWM мәнін TIM3-ке беру
```

```
    PwmTimer->setCaptureCompare(
        1,
        fanDuty,
        PERCENT_COMPARE_FORMAT
    );
}


// ========================================
// ЖҮЙЕНІҢ ЖҰМЫС БЕЛСЕНДІЛІГІ
// STATUS LED
// ========================================

void heartbeatTask()
{
    // LED күйін кері өзгерту
    statusLedState =
        !statusLedState;

    digitalWrite(
        STATUS_LED,
        statusLedState
    );
}


// ========================================
// SERIAL MONITOR-ҒА АҚПАРАТ ШЫҒАРУ
// ========================================

void printStatus()
{
    Serial.println(
        "=================================="
    );

    Serial.println(
        "EV БАТАРЕЯСЫН САЛҚЫНДАТУ КОНТРОЛЛЕРІ"
    );

    Serial.println();
```

```
Serial.print(
   "Батарея температурасы: "
);

Serial.print(
   temperature,
   1
);

Serial.println(" C");


Serial.print(
   "Басқару тапсырмасының периоды: "
);

Serial.println(
   "100 ms"
);


Serial.print(
   "PWM жиілігі: "
);

Serial.println(
   "1000 Hz"
);


Serial.print(
   "Желдеткіштің PWM толтыру коэффициенті: "
);

Serial.print(
   fanDuty
);

Serial.println(" %");


Serial.print(
   "Салқындату деңгейі: "
```

```
    );

    Serial.println(
        coolingState
    );


    Serial.println(
        "====================================="
    );

    Serial.println();
}


// ========================================
// БАСТАПҚЫ БАПТАУЛАР
// ========================================

void setup()
{
    // Serial Monitor іске қосу
    Serial.begin(115200);


    // ADC ажыратымдылығын 12 бит етіп орнату
    analogReadResolution(12);


    // STATUS LED шығыс ретінде баптау
    pinMode(
        STATUS_LED,
        OUTPUT
    );


    // ------------------------------
    // TIM2 — 100 ms
    // Периодтық тапсырмалар үшін
    // ------------------------------

    ControlTimer->setOverflow(
        100000,
```

```
        MICROSEC_FORMAT
    );

    ControlTimer->attachInterrupt(
        timerISR
    );

    ControlTimer->resume();


    // ------------------------------
    // TIM3 — PWM 1000 Hz
    // Желдеткішті басқару үшін
    // ------------------------------

    PwmTimer->setPWM(
        1,
        FAN_PWM_PIN,
        1000,
        0
    );
}


// ========================================
// НЕГІЗГІ ПРОГРАММА ЦИКЛІ
// ========================================

void loop()
{
    // 100 ms өткенін тексеру
    if (task100ms)
    {
        task100ms = false;


        // ----------------------------
        // 100 ms САЙЫН ОРЫНДАЛАТЫН ТАПСЫРМА
        // ----------------------------

        // Температураны оқу
        readTemperature();
```

```
        // Салқындату деңгейін анықтау
        controlCooling();


        // Уақыт есептегіштерін арттыру
        counter500++;

        counter1000++;


        // ----------------------------
        // 500 ms САЙЫН ОРЫНДАЛАТЫН ТАПСЫРМА
        // STATUS LED
        // ----------------------------

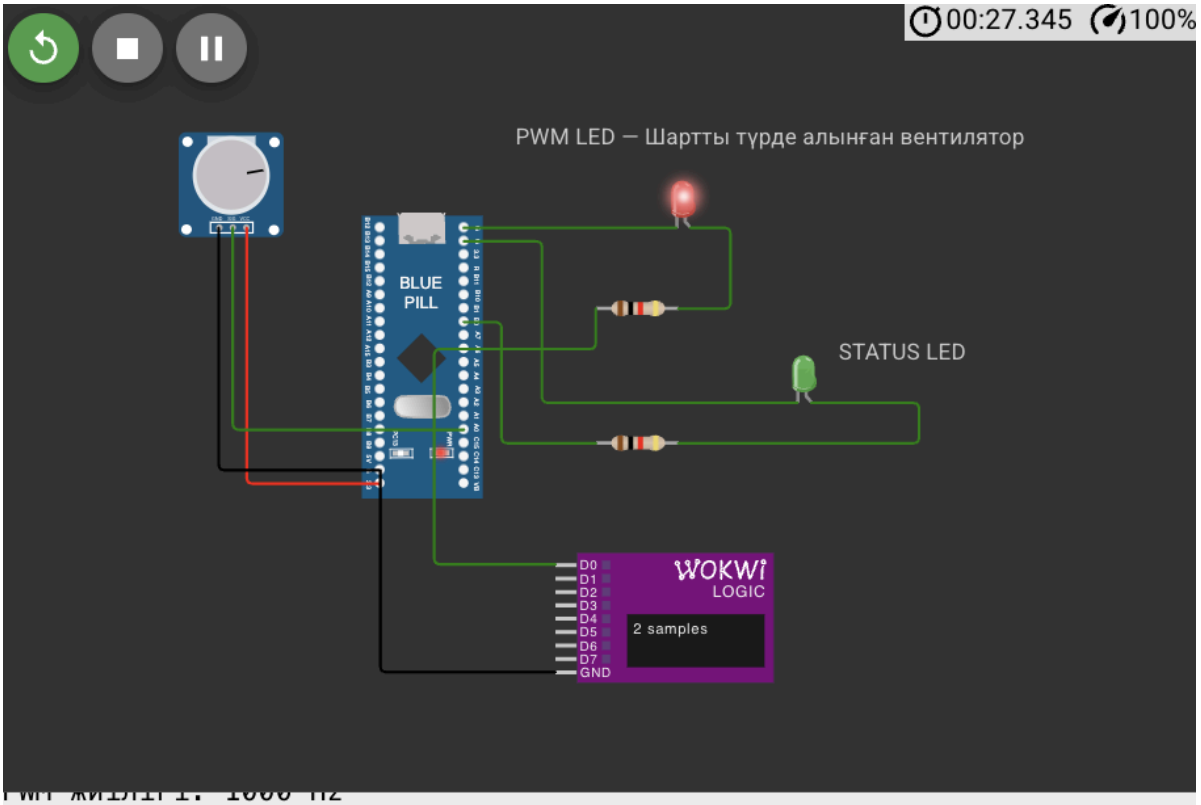
        if (counter500 >= 5)
        {
           counter500 = 0;

           heartbeatTask();
        }


        // ----------------------------
        // 1000 ms САЙЫН ОРЫНДАЛАТЫН ТАПСЫРМА
        // SERIAL MONITOR
        // ----------------------------

        if (counter1000 >= 10)
        {
           counter1000 = 0;

           printStatus();
        }
    }
}
```




```
PWM жиілігі: 1000 Hz
Желдеткіштің PWM толтыру коэффициенті: 100 %
Салқындату деңгейі: МАКСИМАЛДЫ
================================

================================
EV БАТАРЕЯСЫН САЛҚЫНДАТУ КОНТРОЛЛЕРІ

Батарея температурасы: 79.7 C
Басқару тапсырмасының периоды: 100 ms
PWM жиілігі: 1000 Hz
Желдеткіштің PWM толтыру коэффициенті: 100 %
Салқындату деңгейі: МАКСИМАЛДЫ
================================
```

Эксперименттік тексеру
Студент потенциометрдің орнын өзгертіп, кестені толтырады:

№	Temperature, °C	Алынған PWM	Cooling State
1	20		
2	29		

3	31		
4	39		
5	41		
6	49		
7	51		
8	70		

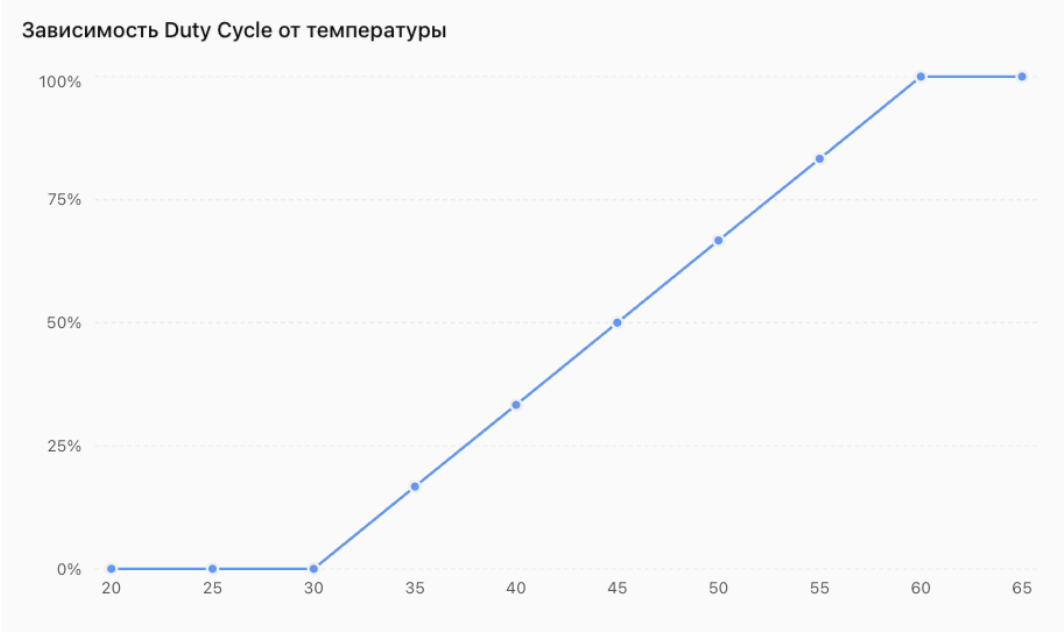
Келесі температура мәндеріне ерекше назар аударыңыз:
29°C 31°C 39°C 41°C 49°C 51°C

Неліктен?
Себебі бұл мәндер диапазондардың шекараларына тікелей дейін және кейін орналасқан.

Бұл келесі шартты операторлардың дұрыс жұмыс істеуін тексеруге мүмкіндік береді:
if
else if
else

7-тапсырма. PWM сигналын бірқалыпты басқару

Электромобиль батареясының салқындату жүйесінде PWM сигналын сатылы басқарудан бірқалыпты басқаруға көшу.



Алгоритм:

- T ≤ 30°C кезінде: PWM = 0%.
- 30°C < T < 60°C кезінде: PWM сызықтық түрде артады.
- T ≥ 60°C кезінде: PWM = 100%.

Аралық диапазонға арналған формула:

$$PWM = \frac{T - 30}{30} \times 100\%$$

Мысалы, температура 45°C болғанда:

$$PWM = \frac{45 - 30}{30} \times 100 = 50\%$$

Маңызды: Бұл — оқу мақсатында жасалған модель. Нақты электромобильдің салқындату жүйесінде басқару алгоритмі желдеткіштің техникалық сипаттамаларына, батареяның температуралық шектеулеріне, диагностикалық жүйеге және қауіпсіздік талаптарына байланысты анықталады.

Бірқалыпты басқару коды

Тек controlCooling() функциясын төмендегі кодпен ауыстырыңыз:

```
void controlCooling()
{
   if (temperature <= 30.0)
   {
      fanDuty = 0;
      coolingState = "OFF";
   }

   else if (temperature >= 60.0)
   {
      fanDuty = 100;
      coolingState = "MAX";
   }

   else
   {
      fanDuty =
         (temperature - 30.0)
         * 100.0
         / 30.0;

      coolingState = "VARIABLE";
   }


   PwmTimer->setCaptureCompare(
      1,
      fanDuty,
      PERCENT_COMPARE_FORMAT
   );
}
```

```
Батарея температурасы: 1.1 C
Басқару тапсырмасының периоды: 100 ms
PWM жиілігі: 1000 Hz
Желдеткіштің PWM толтыру коэффициенті: 0 %
Салқындату деңгейі: OFF
================================
```

Сатылы және бірқалыпты басқару
Қай басқару әдісі атқарушы құрылғының жылдамдығын бірқалыпты өзгертуге мүмкіндік береді?

Бірқалыпты басқарудың VCD файлын қарап, PWM сигналының өзгерісін талдаңыз!

Жеке тапсырмалар

Әрбір студент өз нұсқасына сәйкес бағдарламаны өзгертуі тиіс.
Барлық температура диапазондары оқу мақсатында берілген және Timer/PWM жұмыс принциптерін зерттеу үшін қолданылады. Олар нақты сериялық электромобильдердің (EV) шекті температуралық мәндері болып саналмайды.

№	Модельденетін жүйе	Температура диапазоны	PWM басқару логикасы
1	Батареяны салқындату	0–100 °C	<25 → 0%; 25–35 → 30%; 35–45 → 60%; ≥45 → 100%
2	Инверторды салқындату	0–120 °C	<40 → 0%; 40–60 → 35%; 60–80 → 70%; ≥80 → 100%
3	Электр қозғалтқышын салқындату	0–150 °C	<60 → 0%; 60–80 → 40%; 80–100 → 70%; ≥100 → 100%
4	Батареяның салқындатқыш сұйықтық сорғысын басқару	0–120 °C	<30 → 0%; 30–45 → 30%; 45–60 → 65%; ≥60 → 100%
5	Борттық зарядтау құрылғысын (OBC) салқындату	0–100 °C	<35 → 0%; 35–50 → 40%; 50–65 → 70%; ≥65 → 100%
6	DC/DC түрлендіргішін салқындату	0–120 °C	<40 → 0%; 40–55 → 30%; 55–70 → 60%; ≥70 → 100%
7	Күштік электроника модулін салқындату	0–120 °C	<45 → 0%; 45–60 → 35%; 60–75 → 70%; ≥75 → 100%
8	Батарея қыздырғышын басқару	−20…40 °C	<0 → 100%; 0–5 → 70%; 5–10 → 40%; ≥10 → 0%

9	Батареяны бірқалыпты салқындату	0–100 °C	≤25 → 0%; 25–55 → 0–100% аралығында бірқалыпты өзгеру; ≥55 → 100%
10	Батарея мен инверторды біріктірілген салқындату	2 датчик	PWM мәні екі температура бойынша есептелген басқару сұраныстарының ең үлкеніне сәйкес анықталады

Бақылау сұрақтары

1. Аппараттық таймер дегеніміз не?
2. Микроконтроллерде таймерлер қандай мақсатта қолданылады?
3. Период дегеніміз не?
4. Жиілік дегеніміз не?
5. Период пен жиілік арасында қандай байланыс бар?
6. Үзіліс (Interrupt) дегеніміз не?
7. Үзілісті өңдеу функциясы (ISR — Interrupt Service Routine) дегеніміз не?
8. Неліктен ISR функциясы мүмкіндігінше қысқа болуы тиіс?
9. volatile кілт сөзі қандай мақсатта қолданылады?
10. delay() функциясының аппараттық таймерден қандай айырмашылығы бар?
11. Блоктаушы бағдарлама (Blocking Program) дегеніміз не?
12. Блоктамайтын бағдарлама (Non-blocking Program) дегеніміз не?
13. Периодты тапсырма (Periodic Task) дегеніміз не?
14. Циклдік жоспарлаушы (Cyclic Scheduler) дегеніміз не?
15. PWM (импульс енін модуляциялау) дегеніміз не?
16. Duty Cycle (импульстің толтыру коэффициенті) дегеніміз не?
17. Duty Cycle мен PWM жиілігінің (PWM Frequency) айырмашылығы қандай?
18. PWM электромобильдерде қандай мақсаттарда қолданылады?
19. Неліктен салқындату желдеткішін микроконтроллерге (MCU) тікелей қосуға болмайды?
20. TIM2 таймері қандай мақсатта қолданылады?
21. TIM3 таймері қандай мақсатта қолданылады?
22. Неліктен ADC арқылы өлшеу деректері периодты түрде оқылады?
23. Duty Cycle мәні артқан кезде не болады?
24. Аппараттық PWM қолданудың қандай артықшылықтары бар?
25. Неліктен нақты PWM жиілігі (PWM Frequency) мен температураның шекті мәндері (Temperature Thresholds) жабдықтың техникалық құжаттамасына сәйкес анықталуы тиіс?