# Электромобильдерге арналған микроконтроллер жүйелерін бағдарламалау

Дәуренов ӘД

# Дәріс №3. Embedded C және микроконтроллерлерге арналған бағдарламаларды әзірлеу процесі

**Дәрістің мақсаты**

Студенттерді Embedded C бағдарламалау тілінің ерекшеліктерімен, микроконтроллер бағдарламасының құрылымымен және бағдарламаны әзірлеу, компиляциялау, микроконтроллер жадына жазу және жөндеу кезеңдерімен таныстыру.

Дәріс соңында студент:

- Embedded C мен кәдімгі C бағдарламалауының ерекшеліктерін түсіндіреді;
- микроконтроллер бағдарламасының құрылымын біледі;
- main(), while(1), айнымалылар және функциялардың қызметін түсінеді;
- uint8_t, uint16_t, uint32_t, bool сияқты деректер түрлерін қолдануды біледі;
- pointer, bit operations ұғымдарын түсінеді;
- Memory-Mapped I/O және регистрлердің жұмыс принципін біледі;
- compilation, linking, flashing және debugging кезеңдерін ажыратады;
- datasheet пен reference manual құжаттарының қызметін түсінеді.

# 1. Алдыңғы дәрісті еске түсіру

№2 дәрісте микроконтроллердің аппараттық архитектурасын қарастырдық.

Енді негізгі сұрақ:

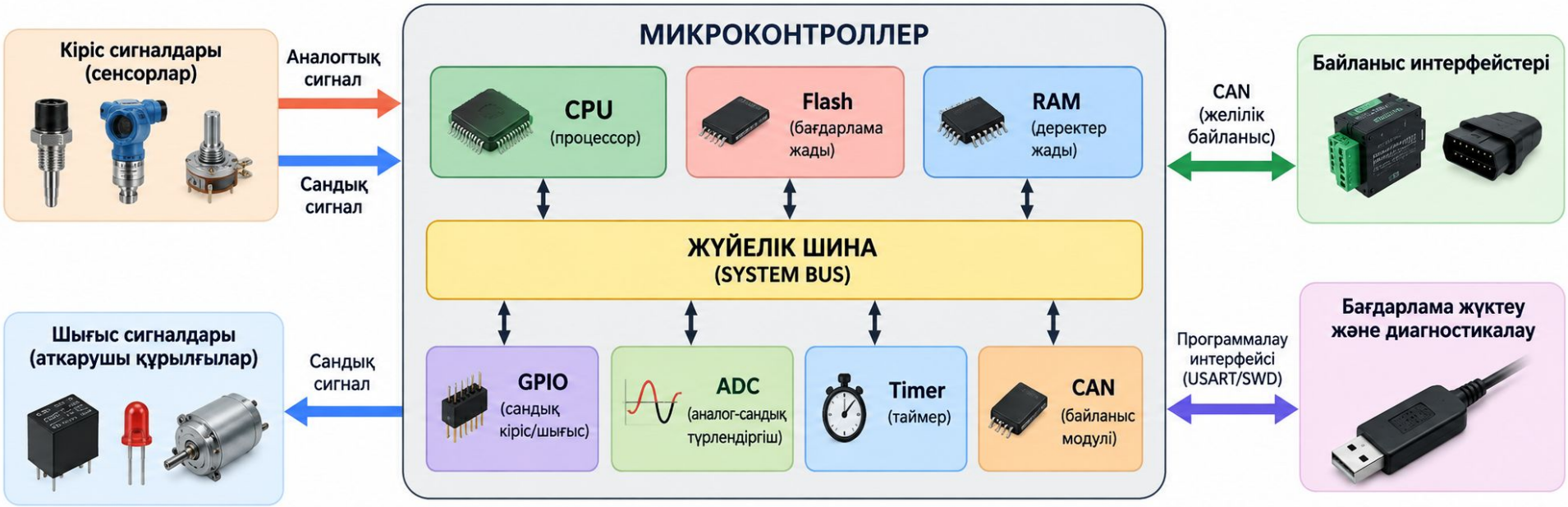
Микроконтроллердің осы аппараттық блоктарын қалай бағдарламалаймыз?

Мысалы, электр қозғалтқышының температурасын бақылау үшін микроконтроллер:

1. Температура датчигінен ақпарат алуы;
2. Алынған мәнді өңдеуі;
3. Белгіленген шекті мәнмен салыстыруы;
4. Қажет болған жағдайда салқындату жүйесін іске қосуы керек.

Осы әрекеттерді орындау үшін микроконтроллерге бағдарлама жазылады.

# Микроконтроллердің аппараттық архитектурасы



# 2. Embedded C дегеніміз не?

Embedded C — кірістірілген жүйелерді бағдарламалау үшін қолданылатын C тілінің мүмкіндіктері мен бағдарламалау тәсілдері.

Embedded C жеке, түбегейлі басқа бағдарламалау тілі емес. Ол стандартты C тілін микроконтроллердің аппараттық ресурстарын басқару үшін қолдануға негізделеді.

Кірістірілген дегеніміз — белгілі бір құрылғының немесе жүйенің ішіне орнатылған, сол құрылғының нақты қызметін орындауға арналған деген мағынаны білдіреді.

Ағылшынша Embedded сөзі қазақша «кірістірілген», «ендірілген» деп аударылады.

**Мысалы, электромобильде:**

Электромобильдің ішінде VCU (Vehicle Control Unit) электрондық басқару блогы орналасады. Оның құрамында микроконтроллер бар.

Микроконтроллерге арнайы бағдарлама жазылады. Ол акселератор педалінің сигналын оқиды, өңдейді және тиісті басқару командасын қалыптастырады.

Яғни бағдарлама компьютерде жеке жұмыс істеу үшін емес, электромобильдің ішіндегі белгілі бір құрылғыны басқару үшін жасалады.

**Сондықтан дәрісте мынадай анықтама берген дұрыс:**

Embedded System (кірістірілген жүйе) — белгілі бір құрылғының құрамына енгізілген және нақты басқару, бақылау немесе есептеу міндеттерін орындауға арналған электрондық-есептеу жүйесі.

Embedded C — осындай кірістірілген жүйелердің микроконтроллерлерін бағдарламалау үшін қолданылатын C бағдарламалау тілі мен бағдарламалау тәсілдері.

Мысалы, STM32 микроконтроллерін C тілінде бағдарламалап, температура датчигін оқу немесе желдеткішті басқару — Embedded C бағдарламалауының қарапайым мысалы.



## ЭЛЕКТРОМОБИЛЬ

Жалпы көліктің жүйесі (қозғалтқыш, батарея, инвертор, салқындату, тежегіш және т.б.)



## VCU басқару блогы

Автомобильдің негізгі функцияларын басқаратын электрондық блок (Vehicle Control Unit)



## Микроконтроллер

VCU ішінде орналасқан микроконтроллер (мысалы, STM32, NXP S32K, Infineon AURIX)




## Embedded Software (C бағдарламасы)

Микроконтроллерге жазылатын бағдарлама (C тілінде)

Мысалы, кәдімгі компьютердегі С бағдарламасы:

```c
#include <stdio.h>

int main(void)
{
    printf("Hello World!");

    return 0;
}
```

Ал микроконтроллер бағдарламасы көбінесе шексіз циклде жұмыс істейді:

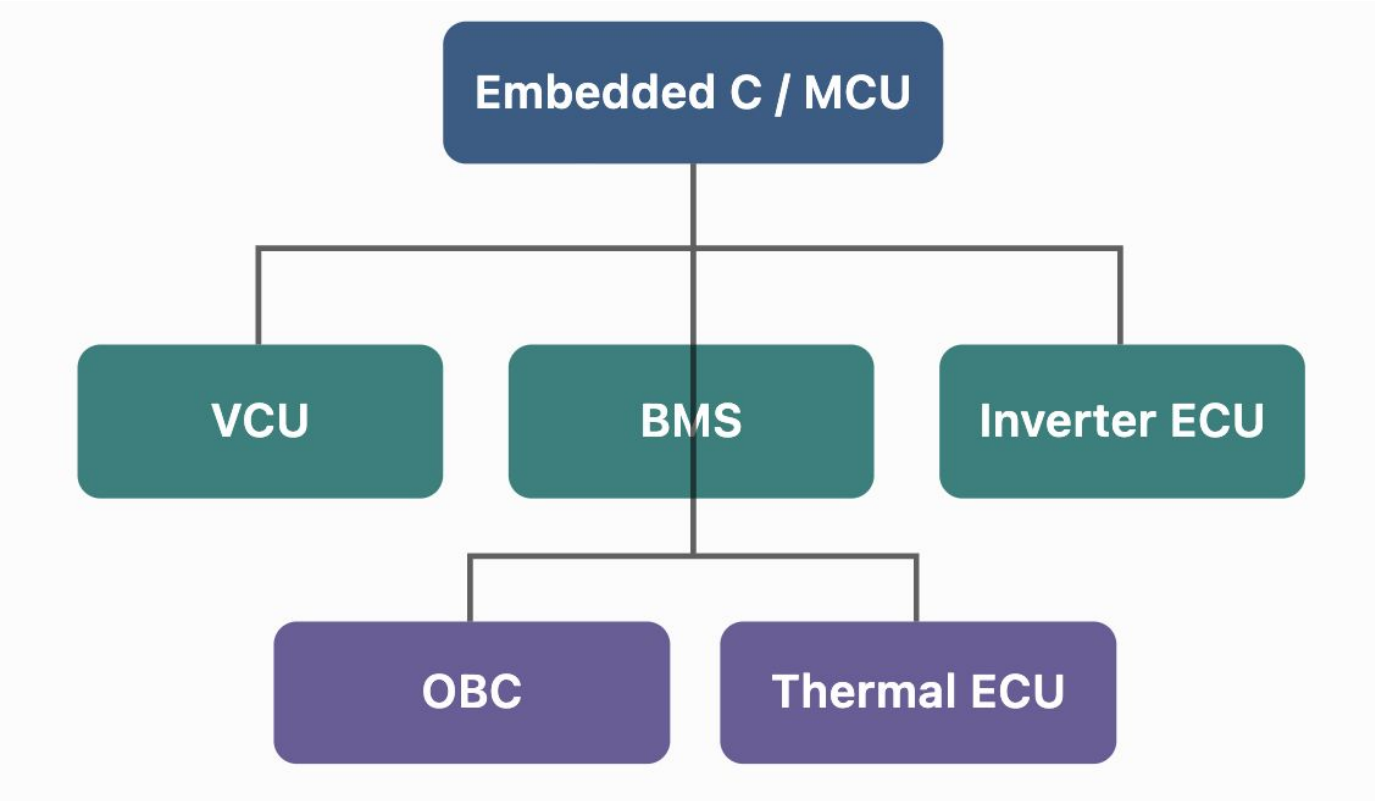
```c
int main(void)
{
    system_init();

    while (1)
    {
        read_sensors();
        process_data();
        control_outputs();
    }
}
```

# Негізгі айырмашылық:

| Кәдімгі C бағдарламасы | Embedded C бағдарламасы |
| --- | --- |
| Компьютерде орындалады | Микроконтроллерде орындалады |
| Операциялық жүйемен жиі жұмыс істейді | Операциялық жүйесіз де жұмыс істей алады |
| Файлдармен, пайдаланушы интерфейсімен жұмыс істей алады | Датчиктермен, регистрлермен, перифериямен жұмыс істейді |
| Бағдарлама аяқталуы мүмкін | Көбінесе үздіксіз жұмыс істейді |
| Ресурстар салыстырмалы түрде көп | RAM, Flash және CPU ресурстары шектеулі болуы мүмкін |

# 3. Электромобильде Embedded C қай жерде қолданылады?




Мысалы, Embedded C көмегімен келесі функцияларды жүзеге асыруға болады:

| Электрондық блок | Бағдарламалық міндет |
|---|---|
| VCU | Педаль сигналдарын өңдеу, автомобиль режимдерін басқару |
| BMS | Аккумулятор кернеуі мен температурасын бақылау |
| Inverter Controller | Электр қозғалтқышын басқару алгоритмдері |
| OBC | Зарядтау процесін басқару |
| Thermal Management ECU | Сорғы, желдеткіш және клапандарды басқару |

Нақты бағдарламалық жасақтама C тілінен басқа тілдерді, модельдік генерацияланған кодты және арнайы automotive платформаларын да қолдануы мүмкін.

# 4. Микроконтроллер бағдарламасының негізгі құрылымы

Embedded C бағдарламасының қарапайым құрылымы:

```
#include <stdint.h>
#include <stdbool.h>

int main(void)
{
    // 1. Инициализация
    system_init();

    // 2. Негізгі цикл
    while (1)
    {
        // Датчиктерді оқу
        read_sensors();

        // Деректерді өңдеу
        process_data();

        // Шығыстарды басқару
        control_outputs();
    }
}
```

Бұл бағдарламаны үш негізгі кезеңге бөлуге болады:



# Embedded C тіліндегі деректер түрлері

Микроконтроллерде жад көлемі шектеулі болуы мүмкін. Сондықтан деректер түрін дұрыс таңдау маңызды.

| Деректер түрі | Өлшемі | Мысал (мәндер диапазоны) | |
|---|---|---|---|
| **uint8_t** | 8 бит | 0 – 255 | Теріс емес 8-биттік бүтін сан |
| **int8_t** | 8 бит | -128 … 127 | Таңбалы 8-биттік бүтін сан |
| **uint16_t** | 16 бит | 0 – 65535 | Теріс емес 16-биттік бүтін сан |
| **int16_t** | 16 бит | -32768 … 32767 | Таңбалы 16-биттік бүтін сан |
| **uint32_t** | 32 бит | 0 – 4294967295 | Теріс емес 32-биттік бүтін сан |
| **float** | Көбінесе 32 бит | 36.5 | Үлкен ондық сандар (нақты сан) |
| **bool** | Көбінесе 8 бит | true / false | Логикалық түр (ақиқат / жалған) |

**uint8_t** (8 бит, 0–255)




**int16_t** (16 бит, –32768...32767)




**Мысал (STM32, стандартты кітапхана)**

```c
#include <stdint.h>
uint8_t a = 250;
int16_t temp = -120;
float voltage = 36.5f;
bool ready = true;
```

**uint8_t, uint16_t** сияқты түрлер **<stdint.h>** кітапханасында анықталады.

# Мысал: электромобиль параметрлерін сақтау (С тілінде)

## С коды

```c
#include <stdio.h>
#include <stdbool.h>

uint16_t battery_voltage = 380;
int16_t motor_temperature = 45;
uint8_t accelerator_percent = 35;
bool battery_ready = true;

```




## Айнымалылардың түсіндірмесі

**`uint16_t battery_voltage = 380;`**
battery_voltage — батарея кернеуі (Вольт, V)

**`int16_t motor_temperature = 45;`**
motor_temperature — қозғалтқыш температурасы (°C)

**`uint8_t accelerator_percent = 35;`**
accelerator_percent — педаль күйінің пайызы (%)

**`bool battery_ready = true;`**
battery_ready — батареяның дайындық күйі (true — дайын, false — дайын емес)

**Нақты автомобильде өлшем бірліктері мен масштабтау тәсілдері арнайы анықталады**
(мысалы, ADC арқылы өлшенген мәндерді Вольт, °C немесе % бірліктеріне түрлендіру).

# 6. Айнымалылар және шартты операторлар

Embedded C тілінде кәдімгі C тіліндегі операторлар қолданылады: if, else, switch, for, while.

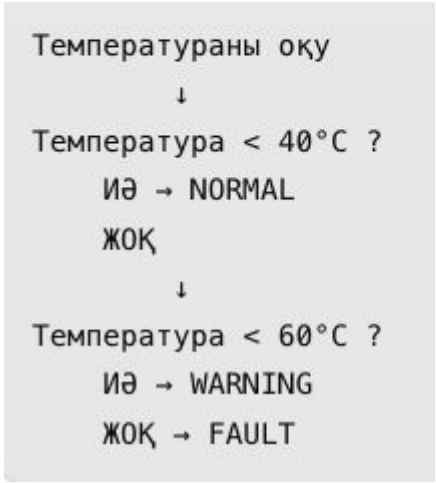
Мысалы, электромобильдің оқу моделіндегі батарея температурасын бақылау:

```c
#include <stdio.h>

int main(void)
{
    float temperature = 48.0f;

    if (temperature < 40.0f)
    {
        printf("NORMAL\n");
    }
    else if (temperature < 60.0f)
    {
        printf("WARNING\n");
    }
    else
    {
        printf("FAULT\n");
    }

    return 0;
}
```




40°C және 60°C — тек оқу тапсырмасына арналған шартты шектер. Нақты EV батареясының қорғаныс шектері өндіруші талаптарына тәуелді.

# Функциялар (Functions) — мысал

Күрделі embedded-бағдарламаны бір ғана `main()` функциясына жазу ыңғайсыз. Сондықтан бағдарлама бірнеше функцияға бөлінеді. Функциялар бағдраламаны оқуға, тестілеуге және өзгертуге көмектеседі.

## С коды

```c
float read_temperature(void)
{
    return 35.5f;
}

bool check_temperature(float temp)
{
    return temp < 60.0f;
}

int main(void)
{
   float temperature = read_temperature();
   bool temperature_ok = check_temperature(temperature);

   return 0;
}
```

## 1. read_temperature()

- **Түрі:** `float`
- **Параметр:** жоқ (`void`)
- **Қызметі:** температураны оқу (мысалы, ADC арқылы)
- **Қайтару мәні:** 35.5 (°C)



Температура датчигі

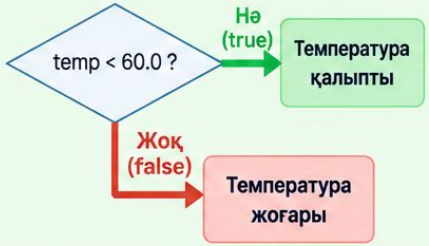
**read_temperature()**

**35.5 °C**

## 2. check_temperature()

- **Түрі:** `bool`
- **Параметр:** `float temp`
- **Қызметі:** температура шекті мәннен төмен бе, соны тексеру
- **Қайтару мәні:** `true` (егер < 60°C), `false` (егер ≥ 60°C)

**check_temperature(temp)**




## 3. main() — негізгі бағдарлама

**Температураны оқу**
temperature = read_temperature();

→

**Тексеру**
temperature_ok = check_temperature (temperature);

→

**Нәтижені қолдану**
(мысалы, LED, журнал немесе қауіпсіздік логикасы)

## Автомобильдегі қолдану мысалы (VCU)



**VCU**
(Vehicle Control Unit)

- Датчиктерден (температура, педаль, кернеу және т.б.) деректерді оқу
- Қауіпсіздік шарттарын тексеру
- CAN хабарламаларын дайындау
- Басқару логикасын орындау

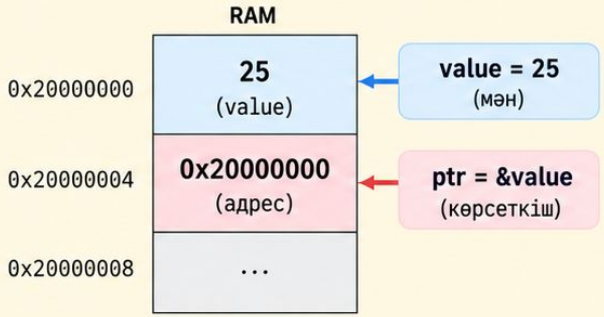
# Pointer (көрсеткіш) — мысал

**Pointer (көрсеткіш)** — жадтағы белгілі бір объектінің адресін сақтайтын айнымалы.

## 1. Код мысалы

```
1  int value = 25;
2  int *ptr = &value;
3
4  // Көрсеткіш арқылы мәнді өзгерту
5  *ptr = 50;
6
```

## 2. Жадтағы көрінісі (RAM)




## 3. Айнымалылардың түсіндірмесі

| Айнымалы | Мәні | Түсіндірме |
|---|---|---|
| value | 25 | Нақты мән (жадта сақталады) |
| &value | 0x20000000 (мысал) | value айнымалысының адресі |
| ptr | 0x20000000 | value айнымалысының адресін сақтайды |
| *ptr | 25 | Көрсеткіш арқылы мәнге қол жеткізу (value мәні) |

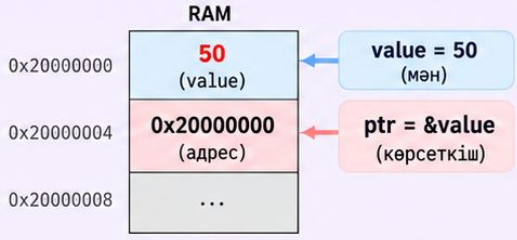
## 4. Көрсеткіш арқылы мәнді өзгерту

```
1  int value = 25;
2  int *ptr = &value;
3  *ptr = 50;
```

**Нәтижесі:**

value айнымалысының мәні 50 болады.

### Өзгерістен кейінгі жадтағы көрінісі



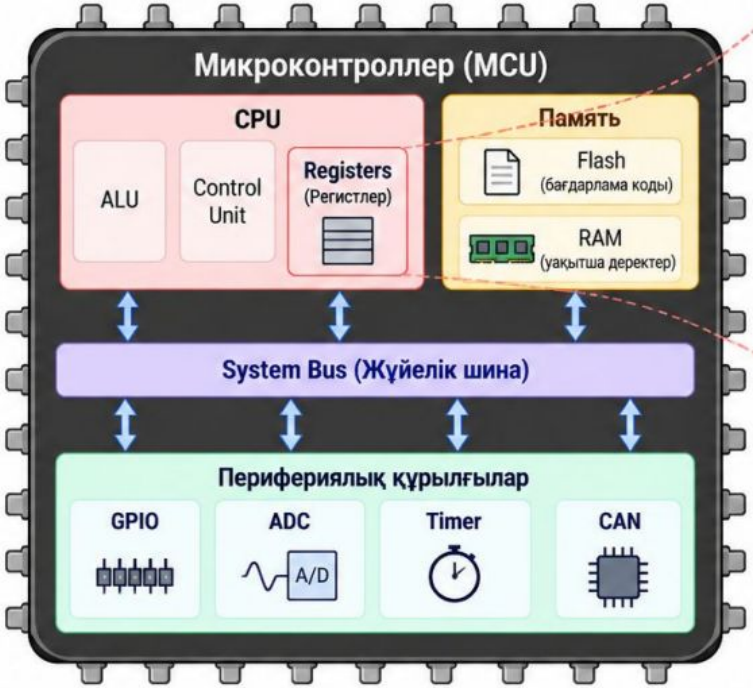

## 5. Негізгі тұжырымдар

- Көрсеткіш жадтағы адресі сақтайды.
- & — адрес алу операторы (мысалы, &value).
- * — көрсеткіш арқылы мәнге қол жеткізу.
- *ptr = 50; → көрсеткіш арқылы мәнді өзгерту.
- Embedded C тілінде көрсеткіштер регистрлермен, буферлермен және аппараттық адрестермен жұмыс істеген кезде маңызды.

Embedded C тілінде pointer регистрлермен, буферлермен және аппараттық адрестермен жұмыс істеу кезінде маңызды

# 10. Регистр дегеніміз не?

**Регистр** — CPU немесе перифериялық құрылғы тікелей жұмыс істейтін өте жылдам сақтау элементі. Микроконтроллердің көптеген аппараттық блоктары регистрлер арқылы басқарылады.




## CPU регистрлері (мысал)

| | |
|---|---|
| R0 | 0x2000 0010 |
| R1 | 0x0000 003C |
| R2 | 0x0000 0001 |
| R3 | 0x0800 1234 |
| ... | ... |

## Периферийлық регистрлер (мысалдар)

| | |
|---|---|
| GPIO Register | 0x4002 0000 |
| ADC Register | 0x4001 2400 |
| Timer Register | 0x4000 1000 |
| CAN Register | 0x4000 6400 |

## Регистермен жұмыс істеу мысалы

```
// GPIO регистріне жазу
#define GPIO_BASE    0x40020000
#define GPIO_ODR     (GPIO_BASE + 0x14)

uint32_t value = 0x01;      // LED қосу
*(volatile uint32_t *)GPIO_ODR = value;
```




## Бағдарлама регистр арқылы аппараттық блоктың жұмысын өзгертеді



**Software** (Бағдарлама) → **Регистерге жазу** (оқу/жазу) →  **Аппараттық блок** (периферия) → **Нәтиже** (LED қосу, таймерді іске қосу, деректерді оқу және т.б.)

Hardware (аппараттық қамтамасыз ету) — микроконтроллердің құрамындағы барлық физикалық электрондық компоненттер мен құрылғылар.

Peripherals (перифериялық модульдер) — сигналдарды өлшеу, кіріс және шығыс желілерін басқару немесе деректерді жіберу және қабылдау сияқты нақты қызметтерді орындайтын микроконтроллердің арнайы аппараттық модульдері.

Hardware Registers (аппараттық регистрлер) — процессордың (CPU) немесе перифериялық модульдердің құрамында орналасқан, деректерді, басқару параметрлерін және құрылғылардың күйін сақтауға арналған регистрлер.

| Аппараттық блок | Қандай қызмет атқарады? |
| --- | --- |
| CPU | Бағдарламаның командаларын орындайды |
| GPIO | Цифрлық сигналдарды қабылдайды және шығарады |
| ADC | Аналогтық сигналды цифрлық мәнге түрлендіреді |
| Timer | Уақытты есептейді |
| CAN | Хабарламалар алмасуды қамтамасыз етеді |
| UART | Деректерді тізбекті түрде жібереді және қабылдайды |

Бұл блоктардың барлығы — жай ғана бағдарламалық функциялар емес, микроконтроллердің ішінде орналасқан нақты электрондық схемалар.

# Микроконтроллердің регистрлері — жалпы түсінік

## 1. Регистр не үшін қажет?

- Регистр — микроконтроллердің ішінде орналасқан, ақпаратты уақытша сақтайтын жад ұяшықтары.
- Регистер арқылы құрылғыларды баптауға, олардың жұмысын басқаруға және бақылауға болады.
- STM32 микроконтроллерінде көптеген регистерлер бар (GPIO, ADC, Timer, UART, CAN және т.б.).
- Бағдарлама белгілі регистрдің биттерін өзгерту арқылы микроконтроллердің шығыс пинін басқарады.

## 2. Жалпы жұмыс схемасы




## 3. Регистрлердің мысалдары (STM32 құрылымында)

| Модуль | Регистрлер (мысалдар) |
|---|---|
| GPIO | MODER, ODR, IDR, BSRR ... |
| ADC | CR1, CR2, DR ... |
| Timer | CNT, ARR, PSC, CR1 ... |
| UART | BRR, CR1, DR ... |
| CAN | MCR, MSR, TSR ... |
| Басқалар | Көптеген регистрлер бар |

## 4. 8-биттік регистр мысалы (электромобильді басқару)




| Бит | Қызметі (мысал) |
|---|---|
| Bit 0 | READY LED (дайындық индикаторы) |
| Bit 1 | FAULT LED (ақаулық индикаторы) |
| Bit 2 | Cooling Fan (салқындатқыш желдеткіш) |
| Bit 3 | Enable Signal (жүйені қосу) |
| Bit 4–7 | Қолданылмайды / резерв |

## 5. Биттік операциялар (C тілінде)

| Оператор | Аталуы | Қызметі |
|---|---|---|
| & | Биттік AND | Екі биттің AND |
| \| | Биттік OR | Екі биттің OR |
| ^ | Биттік XOR | Екі биттің XOR |
| ~ | Биттерді терістеу | Барлық биттерді инверсиялау |
| << | Солға жылжыту | Биттерді солға жылжыту |
| >> | Оңға жылжыту | Биттерді оңға жылжыту |

### Мысал: биттерді басқару

```
1  uint8_t outputs = 0;        // 00000000
2  outputs |= (1U << 0);       // READY LED қосу
3  outputs |= (1U << 2);       // Fan қосу
4  outputs &= ~(1U << 0);      // READY LED өшіру
5  // Нәтиже: outputs = 0b00000100 (тек Fan қосулы)
```

## 6. Нақты мысалдың нәтижесі




**Нәтиже:** outputs = 0b00000100
(тек Cooling Fan қосулы)

# Биттік операциялар (Bit Operations) — қорытынды түсініктеме

## 1. Негізгі идея

- Микроконтроллер регистрлері жеке биттер арқылы басқарылады.
- Әр биттің өз қызметі бар (шығыс/кіріс, режим, флаг және т.б.).
- Биттік операциялар нақты бір битті оқуға, өзгертуге көмектеседі, басқа биттерді өзгертпей.

## 2. 8-биттік регистр мысалы




## 3. Нақты мысал (электромобиль)

| Бит | Қызметі |
|---|---|
| Bit 0 | READY LED (дайындық индикаторы) |
| Bit 1 | FAULT LED (ақаулық индикаторы) |
| Bit 2 | Салқындатқыш желдеткіш (Cooling Fan) |
| Bit 3 | Enable Signal (жүйені қосу) |
| Bit 4–7 | Қолданылмайды / резерв |

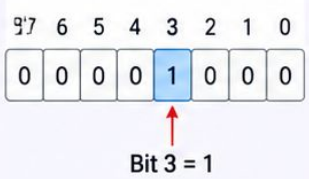
## 4. C тіліндегі негізгі биттік операторлар

| Оператор | Аталуы | Қызметі | Мысал (A = 0b00001010) | Нәтиже |
|---|---|---|---|---|
| & | Биттік AND | Екі биттің AND | A & 0b00000001 | 0 |
| \| | Биттік OR | Екі биттің OR | A \| 0b00000001 | 0b00001011 |
| ^ | Биттік XOR | Екі биттің XOR | A ^ 0b00000001 | 0b00001011 |
| ~ | Биттерді терістеу | Барлық биттерді инверсиялау | ~A | 0b11110101 |
| << | Солға жылжыту | Биттерді солға жылжыту | 1 << 3 | 0b00001000 (8) |
| >> | Оңға жылжыту | Биттерді оңға жылжыту | A >> 1 | 0b00000101 |

## 5. Бір битті 1-ге орнату (Set bit)

```
uint8_t reg = 0b00000000;
reg |= (1U << 3); // Bit 3 = 1
```

**Нәтиже:** 0b00001000 (8)




## 6. Бір битті 0-ге (Clear bit)

```
reg &= ~(1U << 3);   // Bit 3 = 0
```




## 7. Практикалық мысал (шығыстарды басқару)

```
uint8_t outputs = 0;
#define READY_LED   (1U << 0)
#define FAULT_LED   (1U << 1)
#define COOLING_FAN (1U << 2)

outputs |= READY_LED;      // Bit 0 = 1
outputs |= COOLING_FAN;    // Bit 2 = 1
outputs &= (uint8_t)~READY_LED;  // Bit 0 = 0
```

**Нәтижесі:**

```
outputs = 0b00000100
```

Тек Cooling Fan қосулы (Bit 2 = 1)

## Memory-Mapped I/O

Memory-Mapped I/O — процессордың (CPU) GPIO, ADC, Timer, CAN және басқа да перифериялық модульдердің регистрлеріне арнайы адрестер арқылы қатынау, олардан деректерді оқу және оларға деректерді жазу тәсілі.

Memory-Mapped I/O — это способ, при котором процессор (CPU) обращается к регистрам периферийных модулей (GPIO, ADC, Timer, CAN и других) через специальные адреса.

# Embedded-бағдарламаны әзірлеу процесі

(Мысал: STM32 микроконтроллері және жарықдиод)

## 1 Source Code
(Бастапқы код)

main.c

```
#include "stm32f1xx.h"

int main(void) {
    while(1) {
        GPIOB->ODR |= (1<<0);
    }
}
```

Студент C тілінде бағдарлама жазады.

## 2 Compilation
(Компиляциялау)

C Compiler
(arm-none-eabi-gcc)

.o

Object file

Компилятор C кодын машиналық кодқа түрлендіреді.

## 3 Object Files
(Объектілік файлдар)

main.o

startup.o

stm32f1xx.o

...

Әрбір .c файлы үшін объектілік файл (.o) жасалады.

## 4 Linking
(Байланыстыру)

.o файлар

Кітапханалар
(Libraries)

Linker
(arm-none-eabi-ld)

Linker объектілік файлдар мен қажетті кітапханаларды біріктіріп, адрестерді орналастырады.

## 5 Executable File
(Орындаушы файл)

.elf

.bin

.hex

Нәтижесінде .elf файлы жасалады. Қажет болса .bin немесе .hex файлы алынады.

## 6 Flashing
(Бағдарламаны жүктеу)

Programming Tool
(ST-Link, OpenOCD, STM32CubeProgrammer)

Flash Memory

Бағдарлама микроконтроллердің Flash жадына жазылады.

## 7 MCU Execution
(Бағдарламаның орындалуы)

STM32

Программа орындалады

Микроконтроллер жадтағы бағдарламаны орындайды.

## 8 Debugging / Testing
(Тексеру және қателерді түзету)

- Breakpoints
- Variable watch
- Step-by-step
- Serial Monitor
- Қателерді түзету

Бағдарламаның жұмысы тексеріліп, қателер анықталып түзетіледі.



### Даму ортасы (Development Environment)

- STM32CubeIDE
- Keil MDK
- IAR Embedded Workbench
- VS Code + PlatformIO

USB / Debug Interface
(ST-Link)

Бағдарламаны жүктеу



### Микроконтроллер (STM32)

- Flash жадыда бағдарлама
- Бағдарлама орындалады
- GPIO шығысын басқарады

Жарықдиод қосу

**1. Source Code**

Бағдарламашы C тілінде код жазады.

**2. Compilation**

Компилятор C кодын мақсатты процессорға арналған машиналық кодқа түрлендіреді.

**3. Linking**

Linker әртүрлі объектілік файлдар мен қажетті кітапханаларды біріктіріп, адрестерді орналастырады.

**4. Executable File**

Нәтижесінде .elf сияқты орындалатын файл жасалады. Қажет болса .hex немесе .bin файлы алынады.

**5. Flashing**

Бағдарлама микроконтроллердің Flash жадына жазылады.

**6. Debugging**

Бағдарламаның орындалуы тексеріліп, қателер анықталады.

# 15. Компиляция және микроконтроллерге жазу

Мысалы, STM32 микроконтроллеріне арналған бағдарлама жасалады

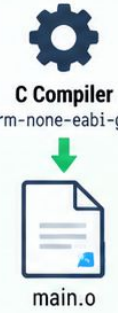
## 1 Бастапқы код (main.c)

```c
// main.c
#include "stm32f1xx.h"

int main(void) {
  while(1) {
    GPIOB->ODR |= (1<<0);
  }
}
```

Студент С тілінде бағдарлама жазады.

## 2 Компиляция (Arm Compiler)



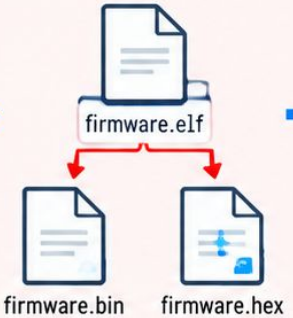

С коды машиналық кодқа түрлендіріліп, объектілік файл (.o) жасалады.

## 3 Байланыстыру (Linker)




Объектілік файлдар мен кітапханалар біріктіріліп, адрестер орналастырылады.

## 4 Орындаушы файл (Executable)




Нәтижесінде .elf файлы жасалады. Қажет болса .bin немесе .hex файлы алынады.

## 5 Жүктеу (ST-LINK / Bootloader)




Бағдарлама ST-LINK арқылы микроконтроллердің Flash жадына жазылады.

## 6 Микроконтроллер (STM32)




Бағдарламаның орындалуы

STM32 микроконтроллері Flash жадындағы бағдарламаны орындайды.

## Мысал: STM32 үшін компиляциялау (команда жолы)

```
arm-none-eabi-gcc -c main.c -o main.o
# Компиляция

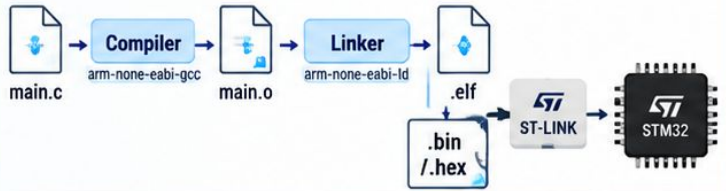
arm-none-eabi-gcc main.o startup.o \
  -T stm32f103.ld -o firmware.elf
# Байланыстыру (Linking)

arm-none-eabi-objcopy -O binary firmware.elf firmware.bin
# .bin файлын жасау
```

## Файлдардың рөлі

| Файл | Сипаттамасы |
|---|---|
| main.c | Бастапқы код (С тілі) |
| main.o | Объектілік файл (бір модульдің машиналық коды) |
| firmware.elf | Орындаушы файл (адрестері орналастырылған) |
| firmware.bin | Flash жадына жазуға арналған екілік файл |
| firmware.hex | Альтернативті формат (Intel HEX) |

## Процестің жалпы сызбанұсқасы



# 16. Debugging — бағдарламаны жөндеу

Debugging — бағдарламаның орындалуын тексеру және қателерді анықтау процесі.

## 1 Breakpoint
(Үзу нүктесі)

```
1  int main(void) {
2      System_Init();
3      while (1) {
4          Read_Sensors();
5          Control();
6          HAL_Delay(100);
7      }
8  }
9
```

Бағдарламаның орындалуын белгілі бір жерде тоқтатады.

## 2 Step Into / Step Over
(Қадаммен орындау)




```
void Read_Sensors(void) {
    uint16_t adc = Read_ADC();
    temperature = Convert_Temp(adc);
}
```

Бағдарламаны қадам бойынша орындауға мүмкіндік береді.

## 3 Watch Variables
(Айнымалыларды бақылау)

| Name | Value | Type |
|---|---|---|
| battery_voltage | 382 | int |
| motor_temp | 42 | int |
| accelerator | 35 | int |
| system_state | READY | enum |

Айнымалылардың ағымдағы мәндерін көрсетеді.

## 4 Registers View
(Регистрлерді көрсету)

| Register | Value (Hex) |
|---|---|
| R0 | 0x20001000 |
| R1 | 0x0000017E |
| R2 | 0x0000002A |
| R3 | 0x00000023 |
| R4 | 0x08001234 |
| PC | 0x080003F0 |
| LR | 0x0800012C |
| SP | 0x2001FF80 |

Микроконтроллер регистрлерінің мәндерін көрсетеді.

## 5 Serial Monitor
(Сериалдық монитор)

```
COM3 (115200 baud)
Battery Voltage: 382 V
Motor Temperature: 42 C
Accelerator: 35 %
System State:  READY
>
```

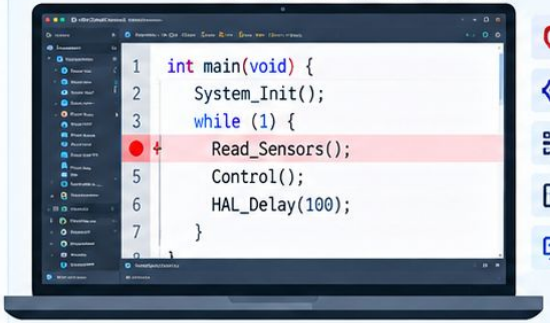
Бағдарламадан жіберілген мәтіндік және диагностикалық ақпаратты көрсетеді.

## 6 Қорытынды
(Нәтиже)

- ✓ Бағдарламаның дұрыс орындалуын тексеру
- ✓ Қателерді анықтау
- ✓ Айнымалылар мәнін бақылау
- ✓ Аппараттық конфигурация қателерін табу
- ✓ Жүйенің логикасын тексеру

## Орындалу процесі (Debug режимінде)

STM32CubeIDE (Debug Perspective)




- Breakpoints
- Variables
- Registers
- Memory
- Console

## Жұмыс істеу сызбанұсқасы




## Нәтиже мысалы (Serial Monitor)

```
COM3 (115200 baud)
Battery Voltage: 382 V
Motor Temperature: 42 C
Accelerator: 35 %
System State:  READY
```

Компиляция сәтті аяқталғаны бағдарламаның міндетті түрде дұрыс жұмыс істейтінін білдірмейді.

Бағдарламада логикалық қателер, уақыттық қателер немесе аппараттық конфигурация қателері болуы мүмкін.

# 17. Datasheet және Reference Manual

Микроконтроллерді бағдарламалау кезінде өндірушінің техникалық құжаттары өте маңызды.

## Datasheet — микроконтроллердің жаллы сипаттамасы

Datasheet — микроконтроллердің неігзгі сипаттамаларын, электрлік параметрлерін және жұмыс шарттарын сипаттайтын құжат.

### Datasheet құжатында көбінесе:

- Микроконтроллердің неігзгі сипаттамалары (ядро, жиілік, жад көлемі)
- Қоректендіру кернеуі (VDD), электрлік параметрлер
- Жұмыс температурасы
- Pinout (шығыстардың орналасуы)
- Қуат тұтынуы, электрлік параметрлер
- Жад көлемі (Flash, RAM)
- Периферияльқ мүмкіндіктер (ADC, UART, SPI, CAN және т.б.)




## Reference Manual — периферияльқ модульдердің толық сипаттамасы

Reference Manual — микроконтроллердің ішкі құрылымын, регистрлерін және периферияльқ модульдердің жұмысын толық сипаттайтын құжат.

### Reference Manual құжатында:

- GPIO регистрлері және конфигурациясы
- ADC модулінің жұмысы
- Timer модулінің жұмысы
- Interrupt жүйесі (NVIC)
- Clock configuration (RCC)
- Communication peripherals (UART, SPI, I2C, CAN, USB және т.б.)
- Әр регистрдің биттік өрістері және жұмысы




## Мысал: STM32F103C8T6




## Datasheet-тағы pinout мысалы




## Reference Manual-дағы GPIO регистрі мысалы

**GPIOx_MODER** (x = A..G) **Адрес:** 0x4001 0800 (GPIOA)

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | ... | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MODER15 | | | | ... | | | | | | MODER1 | | MODER0 |

| Биттер | Атауы | Сипаттамасы |
|---|---|---|
| 31:30 | MODER15 | Pin 15 режимі |
| ... | ... | ... |
| 3:2 | MODER1 | Pin 1 режимі (00 – Input, 01 – Output, 10 – Alternate, 11 – Analog) |
| 1:0 | MODER0 | Pin 0 режимі |

# 18. Нақты автомобильдік микроконтроллерлер және бағдарламалау құралдары

Автомобиль өнеркәсібінде әртүрлі өндірушілердің микроконтроллерлері қолданылады. Оларды бағдарламалау үшін өндірушінің арнайы әзірлеу ортасы мен SDK құралдары пайдаланылады.

## 1 NXP S32K344

(S32K3 отбасы)

Автомобильдік басқару жүйелеріне арналған 32-биттік Arm Cortex-M7 микроконтроллері.



- Arm Cortex-M7 дейін 160 MHz
- Автомобильдік стандарт (ISO 26262)
- Қауіпсіздік модульдері (HSE, SHE)
- Көп коммуникация (CAN, LIN, FlexRay, Ethernet)

### Бағдарламалау құралдары

**S32 Design Studio**

- S32 Design Studio (IDE)
- S32 SDK (драйверлер, мысалдар)
- Compiler, Debugger, Trace
- Configuration Tools

## 2 Infineon AURIX TC375

(AURIX TriCore)

AURIX TriCore архитектурасына негізделген автомобильдік жоғары өнімді MCU.



- TriCore (3 ядроға дейін) 300 MHz+
- Автомобильдік қауіпсіздік (ISO 26262, ASIL)
- Көлік жүйелеріне арналған (ECU, ADAS, Powertrain)
- CAN, LIN, FlexRay, Ethernet

### Бағдарламалау құралдары

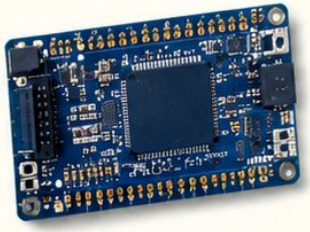
**AURIX Development Studio**

- AURIX Development Studio (IDE)
- iLLD (драйвер кітапханасы)
- Compiler, Debugger, Trace
- Configuration Tools (MemTool)

## 3 Renesas RH850

(RH850 отбасы)

Автомобильдік ECU жүйелерінде қолдануға арналған жоғары сенімді MCU отбасы.



- RH850 (32 биттік) жоғары өнімділік
- Автомобильдік стандарт (ISO 26262)
- Қозғалтқыш, трансмиссия, шасси жүйелері
- CAN, LIN, FlexRay, Ethernet, автомобильдік I/O

### Бағдарламалау құралдары

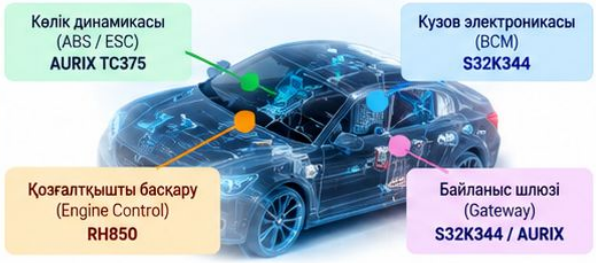
**e² studio** (Renesas)

- e² studio (IDE)
- RH850 Toolchain (Compiler/Debugger)
- FSP (Flexible Software Package)
- Configuration Tools

## Негізгі сипаттамаларды салыстыру

| Параметр | NXP S32K344 | Infineon AURIX TC375 | Renesas RH850 |
|---|---|---|---|
| Архитектура | Arm Cortex-M7 | TriCore (3 ядро) | RH850 (32-bit) |
| Жұмыс жиілігі | дейін 160 MHz | 300+ MHz | 100–240 MHz |
| Қолдану саласы | Жалпы ECU, аймақтық контроллерлер | Қауіпсіздік маңызды жүйелер (ADAS, Powertrain) | ECU (Engine, Transmission, Chassis) |
| Коммуникация | CAN, LIN, FlexRay, Ethernet | CAN, LIN, FlexRay, Ethernet | CAN, LIN, FlexRay, Ethernet |
| IDE / Tools | S32 Design Studio | AURIX Development Studio | e² studio |
| Сәйкестік | ISO 26262 | ISO 26262 (ASIL) | ISO 26262 |

## Автомобильде қолдану мысалдары




## Жалпы қорытынды

- Автомобильдік микроконтроллерлердің регистрлері, кітапханалары және бағдарламалық ортасы әртүрлі болады.
- С тілі, айнымалылар, функциялар, pointer, биттік операциялар және компиляция принциптері ортақ.
- Әр өндіруші өз әзірлеу ортасын (IDE) және SDK ұсынады.
- Жобаның талаптарына сәйкес нақты микроконтроллер мен құралдар таңдалады.

# SDK (Software Development Kit) не?

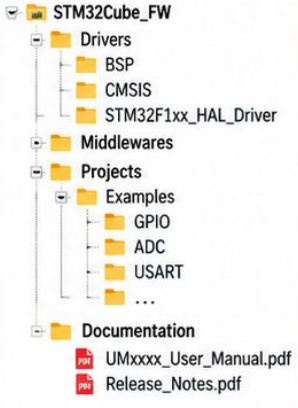
SDK — бағдарламалық қамтамасыз етуді әзірлеуге арналған дайын құралдар, кітапханалар және мысалдар жиынтығы.

## 1 SDK дегеніміз не?

**SDK** — микроконтроллерге арналған драйверлер, кітапханалар, мысалдар және әзірлеуге қажетті қосымша ресурстар жиынтығы.

Қарапайым тілмен айтқанда, **SDK** — бағдарламашының жұмысын жеңілдететін дайын бағдарламалық құралдар жинағы.

## 2 SDK құрамында не болады?




**Drivers**
GPIO, ADC, CAN сияқты модульдерді басқару

**Libraries**
Дайын бағдарламалық функциялар

**Examples**
Бағдарламалардың дайын мысалдары

**Documentation**
Қолдану нұсқаулықтары

**Configuration Tools**
Микроконтроллер модульдерін баптау (мысалы, STM32CubeMX)

## 3 STM32 мысалы: LED қосу

**SDK немесе дайын кітапхана болмаса (төмен деңгейде)**

```
// PB0 пинін шығыс етіп баптау
RCC->AHB2ENR |= (1 << 1);
// GPIOB тактін қосу

GPIOB->MODER &= ~(3 << (0*2));
GPIOB->MODER |= (1 << (0*2));
// PB0 – шығыс

GPIOB->BSRR = (1 << 0);
// PB0 – жоғары деңгей (LED ON)
```

GPIO регистрлерін, адрестерін және биттерін өзіңіз баптайсыз.

**Дайын HAL кітапханасын қолдансаңыз (SDK)**

```
#include "stm32f1xx_hal.h"

int main(void)
{
    HAL_Init();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    HAL_GPIO_WritePin(GPIOB,
                      GPIO_PIN_0,
                      GPIO_PIN_SET);

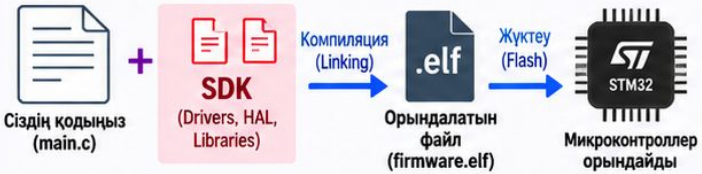
    while (1) { }
}
```

Бір ғана функция арқылы PB0 шығысында жоғары логикалық деңгей орнатылады.




STM32 микроконтроллері

## 4 SDK жұмыс істеу принципі




SDK драйверлері микроконтроллердің перифериялық модульдерімен жұмыс істеуге арналған дайын функцияларды (API) ұсынады.

## 5 SDK, IDE және Toolchain айырмашылығы

**IDE (Integrated Development Environment)**

- Код жазу
- Жоба басқару
- Компиляциялау
- Debug
- Ыңғайлы интерфейс

Мысал:
STM32CubeIDE
S32 Design Studio
AURIX Development Studio

**Toolchain**

- Compiler
- Assembler
- Linker
- Debugger
- Build tools

Мысал:
Arm GNU Toolchain
e² studio toolchain

**SDK**

- Драйверлер
- Кітапханалар
- Мысалдар
- Құжаттама
- Конфигурация құралдары

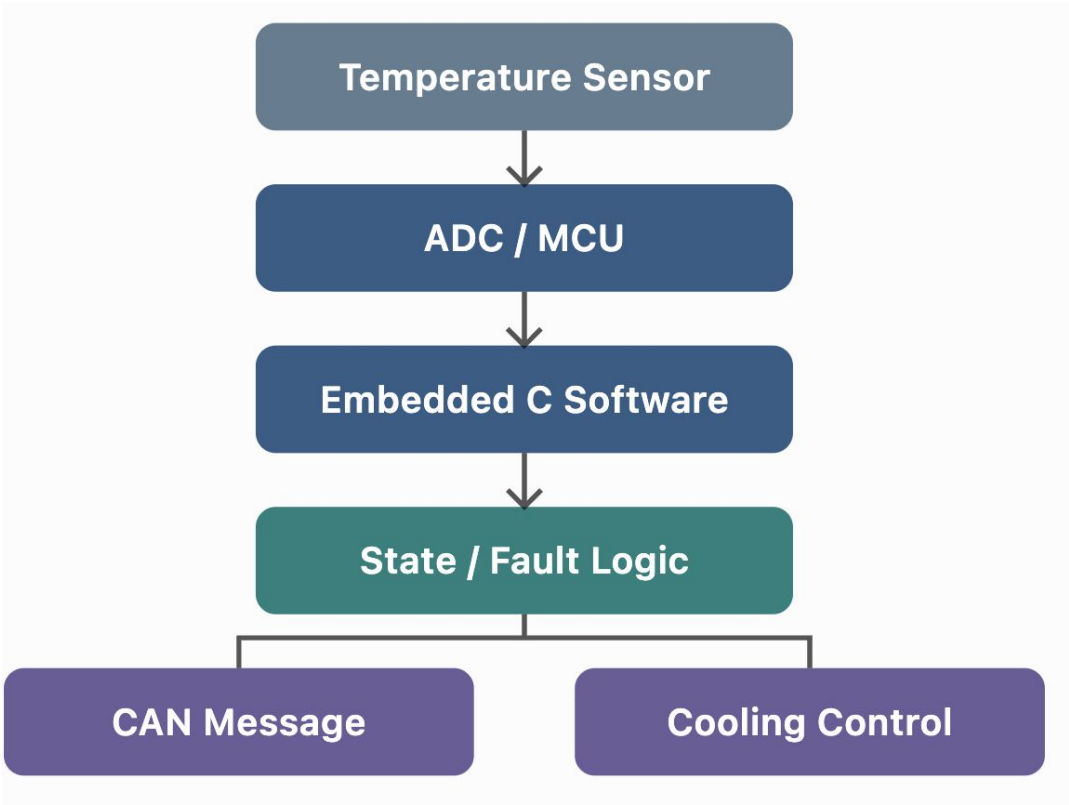
Мысал:
STM32Cube HAL
S32K3 RTD
AURIX MCU Software

## 6 Нақты әлемдегі мысалдар

| Өндіруші | Микроконтроллер | IDE | SDK / Драйверлер |
|---|---|---|---|
| ST | STM32F103 | STM32CubeIDE | STM32Cube HAL (Drivers, Examples) |
| NXP | S32K344 | S32 Design Studio | S32K3 RTD (Real-Time Drivers) |
| Infineon | AURIX TC375 | AURIX Development Studio | iLLD (Infineon Libraries) |
| RENESAS | RH850 | e² studio | RH850 Toolchain (FSP) |

**Есте сақтаңыз:** SDK — микроконтроллердің ішіндегі физикалық құрылғы емес. Ол — бағдарламашы қолданатын бағдарламалық құралдар жиынтығы.

# Embedded бағдарламаны электромобиль жүйесімен байланыстыру




Осы жүйеде:

- Sensor физикалық шаманы өлшейді;
- ADC аналогтық сигналды цифрлық мәнге түрлендіреді;
- MCU Embedded C бағдарламасын орындайды;
- Software деректерді тексереді және шешім қабылдайды;
- CAN ақпаратты басқа ECU-ға жібереді;
- Output Driver атқарушы құрылғыны басқара алады.

Бұл — аппараттық және бағдарламалық қамтамасыз етудің өзара байланысы.

# 19. Электромобильге арналған бағдарламаның оқу мысалы

Міндет: аккумулятордың температурасын бақылау және жүйе күйін анықтау.

## 1 С тіліндегі бағдарлама

```c
#include <stdio.h>
#include <stdint.h>

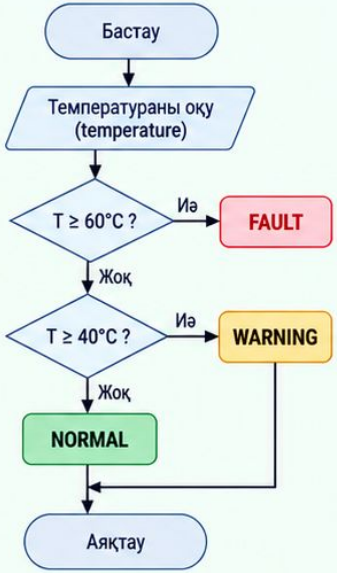
typedef enum
{
    BATTERY_NORMAL,
    BATTERY_WARNING,
    BATTERY_FAULT
} BatteryState;

BatteryState check_battery(float temperature)
{
    if (temperature >= 60.0f)
        return BATTERY_FAULT;
    else if (temperature >= 40.0f)
        return BATTERY_WARNING;
    else
        return BATTERY_NORMAL;
}

int main(void)
{
    float temperature = 52.5f;
    BatteryState state = check_battery(temperature);

    if (state == BATTERY_NORMAL)
        printf("NORMAL\n");
    else if (state == BATTERY_WARNING)
        printf("WARNING\n");
    else
        printf("FAULT\n");
    return 0;
}
```

## 2 Алгоритмнің блок-схемасы




## 3 Компьютерде орындау мысалы

Кіріс деректер:

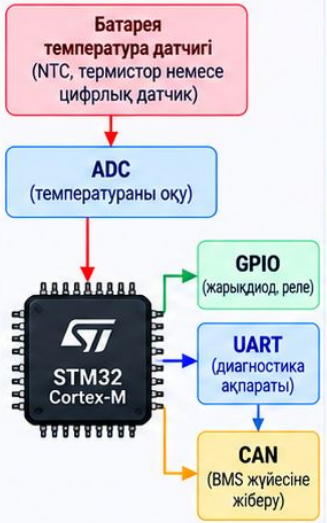
```
temperature = 52.5f
```




Бағдарлама нәтижесі:

```
WARNING
```

## 4 STM32-де қолдану (нақты жүйе)

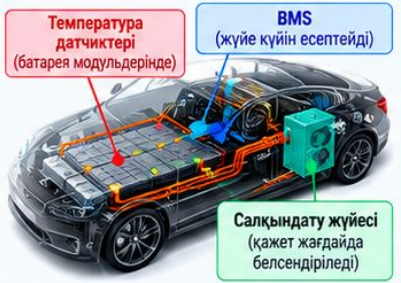



## 5 Жүйе жұмысының мысалы

Температура және жүйе күйі

| Температура | Жүйе күйі | Іс-әрекет |
|---|---|---|
| T < 40°C | NORMAL | Қалыпты жұмыс |
| 40°C ≤ T < 60°C | WARNING | Ескерту, салқындату жүйесін белсендіру |
| T ≥ 60°C | FAULT | Қорғаныс режимі, жүйені өшіру |

## 6 Нақты көлік жүйесіндегі мысал




## 7 STM32-де мүмкін болатын нақты код (фрагмент)

```c
// ADC арқылы температураны оқу (NTC термистор)
uint16_t adc_value = HAL_ADC_GetValue(&hadc1);
float temperature = adc_to_temperature(adc_value);

BatteryState state = check_battery(temperature);

// Нәтижені UART арқылы шығару
if (state == BATTERY_NORMAL)
    printf("[BMS] Temp: %.1f C, State: NORMAL\r\n", temperature);
else if (state == BATTERY_WARNING)
    printf("[BMS] Temp: %.1f C, State: WARNING\r\n", temperature);
else
    printf("[BMS] Temp: %.1f C, State: FAULT\r\n", temperature);
```

## 8 Нақты жүйедегі қосымша функциялар (BMS)

- Бірнеше датчиктен температураны оқу
- Уақыттық тексерулер (фильтрация, орташа мән)
- Датчик диагностикасы (ашық/қысқа тұйықталу)
- Салқындату жүйесін басқару
- Қауіпсіздік күйлері (Overtemperature, Shutdown)
- Басқа параметрлерді бақылау (кернеу, ток, SOC)

## 9 Қорытынды

- Осы мысалда C тілі, функциялар, enum, шартты операторлар қолданылды.
- Бағдарлама компьютерде де орындалады.
- STM32-де нақты датчиктен температураны ADC арқылы оқу қажет.
- Нәтижені GPIO, UART немесе CAN арқылы шығару мүмкін.
- Нақты BMS жүйесінде қосымша диагностика, қорғаныс алгоритмдері және жүйені басқару логикасы қолданылады.

```c
#include <stdio.h>
#include <stdint.h>

typedef enum
{
    BATTERY_NORMAL,
    BATTERY_WARNING,
    BATTERY_FAULT
} BatteryState;

BatteryState check_battery(float
temperature)
{
    if (temperature >= 60.0f)
    {
        return BATTERY_FAULT;
    }
    else if (temperature >= 40.0f)
    {
        return BATTERY_WARNING;
    }
    else
    {
        return BATTERY_NORMAL;
    }
}

int main(void)
{
    float temperature = 52.5f;

    BatteryState state =
        check_battery(temperature);

    if (state == BATTERY_NORMAL)
    {
        printf("NORMAL\n");
    }
    else if (state == BATTERY_WARNING)
    {
        printf("WARNING\n");
    }
    else
    {
        printf("FAULT\n");
    }

    return 0;
}
```

Embedded C — микроконтроллердің аппараттық мүмкіндіктерін бағдарламалық түрде басқаруға арналған негізгі құралдардың бірі.

Микроконтроллерге арналған бағдарламаны әзірлеу кезінде бағдарламашы C тілінің синтаксисін ғана емес, аппараттық архитектураны, жадты, регистрлерді, перифериялық құрылғыларды және әзірлеу құралдарын да білуі қажет.

Дәрістің негізгі логикалық байланысы:

C бағдарламасы → Компиляция → Машиналық код → MCU Flash → CPU → Перифериялық құрылғылар → Электромобиль жүйесі

## Бақылау сұрақтары

1. Embedded C дегеніміз не?
2. Embedded C мен кәдімгі C бағдарламасының айырмашылығы қандай?
3. Микроконтроллер бағдарламасында main() функциясы қандай қызмет атқарады?
4. while(1) циклі не үшін қолданылады?
5. Initialization дегеніміз не?
6. uint8_t және uint16_t түрлерінің айырмашылығы қандай?
7. Pointer дегеніміз не?
8. & және * операторлары көрсеткіштермен жұмыс істегенде қандай қызмет атқарады?
9. Биттік операциялар не үшін қажет?
10. Memory-Mapped I/O дегеніміз не?

11. HAL және Register-Level Programming арасындағы айырмашылық қандай?
12. Compiler не істейді?
13. Linker қандай қызмет атқарады?
14. Flashing дегеніміз не?
15. Debugging не үшін қажет?
16. Breakpoint дегеніміз не?
17. Toolchain құрамына қандай құралдар кіреді?
18. Datasheet пен Reference Manual арасындағы айырмашылық қандай?