EMBEDDED

л.12
DMA

Палій Святослав

Український Католицький Університет
APPLIED SCIENCES FACULTY

Полінг

Опис: Полінг передбачає постійне опитування (перевірку) певного регістру або прапорця, щоб визначити, чи настав момент появи сигналу. Програма перебуває в циклі очікування, постійно перевіряючи стан пристрою.

Переваги:

- **Простота:** Легко реалізувати, не потребує складної логіки.
- **Передбачуваність:** Можна встановити часові інтервали опитування, що дає змогу контролювати затримку.

Недоліки:

- **Високе навантаження на CPU:** Постійне опитування займає процесорні цикли, навіть якщо подія відбувається рідко.
- **Неефективність:** У системах із декількома завданнями може призводити до блокування інших процесів, оскільки CPU «засиджується» у циклі перевірок.

```
/*
  Приклад: Опитування (Polling)
  - Читаємо стан кнопки.
  - Якщо кнопка натиснута, вмикаємо світлодіод.
*/

const int buttonPin = 2;   // пін, до якого підключена кнопка
const int ledPin    = 13;  // пін, до якого підключено світлодіод

void setup() {
  pinMode(buttonPin, INPUT_PULLUP);  // Підтягуємо до “1”
  pinMode(ledPin, OUTPUT);
}

void loop() {
  int buttonState = digitalRead(buttonPin);

  if (buttonState == LOW) {   // кнопка натиснута (LOW)
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
}
```

Переривання

Опис: Переривання дозволяють пристрою сигналізувати про появу потрібної умови без необхідності постійного опитування. Коли подія виникає, апаратний засіб відправляє сигнал (IRQ), після чого процесор переходить до виконання спеціальної процедури обробки – Interrupt Service Routine (ISR).

Переваги:

- **Ефективне використання CPU:** Процесор може виконувати інші завдання, поки не отримає сигнал про подію.
- **Швидкість реакції:** Подія може оброблятися негайно, як тільки вона відбувається, без затримок, пов’язаних із циклічними перевірками.

Недоліки:

- **Контекстне перемикання:** Обробка переривань вимагає перемикання контексту, що може вносити певну затримку основної програми.
- **Складність реалізації:** Необхідність встановлення правильних пріоритетів.

```
/*
  Приклад: Обробка переривань
  - При натисканні кнопки генерується переривання, ISR встановлює прапорець.
  - Основна функція loop() реагує на прапорець і керує світлодіодом.
*/

const int buttonPin = 2;    // пін, що підтримує переривання (на Arduino Uno це пін 2 або 3)
const int ledPin    = 13;
volatile bool buttonPressed = false;  // прапорець, встановлюється в ISR

void setup() {
  pinMode(buttonPin, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(buttonPin), buttonISR, FALLING);
}

void loop() {
  if (buttonPressed) {
    digitalWrite(ledPin, HIGH);
    delay(100);            // світлодіод світиться коротке число часу
    digitalWrite(ledPin, LOW);
    buttonPressed = false; // скидаємо прапорець після обробки
  }
}

// Обробник переривання
void buttonISR() {
  buttonPressed = true;
}
```

Direct **M**emory **A**ccess (DMA)

Опис: DMA – це апаратний механізм, який дозволяє безпосередньо передавати дані між пристроєм та оперативною пам’яттю, не навантажуючи процесор. DMA зазвичай застосовується при передачі великих обсягів даних. Після завершення операції DMA часто генерує переривання, щоб сповістити CPU про завершення передачі.

Переваги:

- **Мінімальне навантаження на CPU:** Після налаштування DMA працює автономно, звільняючи процесор для інших завдань.
- **Висока ефективність при передачі великих даних:** Особливо корисно у мультимедійних застосуваннях або при роботі з дисками.

Недоліки:

- **Складність налаштування:** Використання DMA вимагає ретельного управління каналами та ресурсами, що може ускладнювати розробку.
- **Обмеженість ресурсів:** DMA-каналів у системі може бути обмежено кілька, тому важливо їх ефективно розподіляти серед пристроїв.

```
/*
  Приклад: Використання DMA (Direct Memory Access)
*/
#include <DMAChannel.h>

DMAChannel dmaChannel;  // створюємо об'єкт для роботи з DMA

volatile bool dmaTransferComplete = false;
const int BUFFER_SIZE = 32;
volatile uint8_t srcBuffer[BUFFER_SIZE] = {
  0, 1, 2, 3, 4, 5, 6, 7,
  8, 9, 10, 11, 12, 13, 14, 15,
  16, 17, 18, 19, 20, 21, 22, 23,
  24, 25, 26, 27, 28, 29, 30, 31
};
volatile uint8_t dstBuffer[BUFFER_SIZE];

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

  // Налаштовуємо DMA: задаємо буфер-джерело та буфер-призначення
  dmaChannel.disable();
  dmaChannel.sourceBuffer(srcBuffer, BUFFER_SIZE);
  dmaChannel.destinationBuffer(dstBuffer, BUFFER_SIZE);

  // Налаштовуємо переривання DMA
  dmaChannel.attachInterrupt(dmaISR);

  // Запускаємо DMA трансфер
  dmaChannel.enable();
  Serial.println("DMA трансфер розпочато");
}

void loop() {
  if (dmaTransferComplete) {
    Serial.println("DMA трансфер завершено. Дані в dstBuffer:");
    for (int i = 0; i < BUFFER_SIZE; i++) {
      Serial.print(dstBuffer[i]);
      Serial.print(" ");
    }
    Serial.println();

    dmaTransferComplete = false;

    // Для демонстрації: затримка перед повторним запуском DMA
    delay(5000);
    Serial.println("Повторно запускаємо DMA трансфер");
    dmaChannel.enable();
  }
}

// DMA ISR: викликається після завершення передачі
void dmaISR() {
  dmaChannel.clearInterrupt();
  dmaTransferComplete = true;
}
```

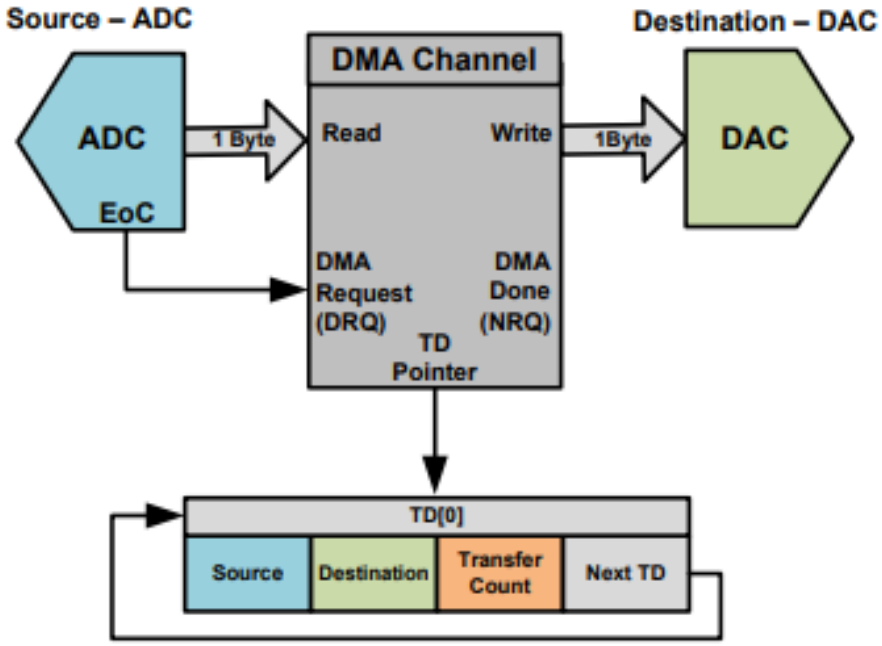
Метод	Витрати CPU	Латентність	Складність реалізації	Сфери застосування
Полінг	Високе (постійне опитування)	Передбачувана, проте може бути повільною через інтервали опитування	Простий у реалізації	Системи з високою частотою подій або коли ресурс процесора не є обмеженим
Переривання	Низькі (CPU займається іншими завданнями)	Швидка реакція, з невеликим накладним часом на контекстне перемикання	Помірна – вимагає належного управління пріоритетами та синхронізації	Багатозадачні системи, де події відбуваються не надто часто або вимагають негайної реакції
DMA	Мінімальні (автономна передача даних)	Залежить від налаштування; сам процес передачі – швидкий, завершення часто сигналізується через переривання	Висока – потребує налаштування апаратних каналів та ресурсів	Передача великих обсягів даних (медіа, дискові операції) та системи з критичними вимогами до ефективності CPU

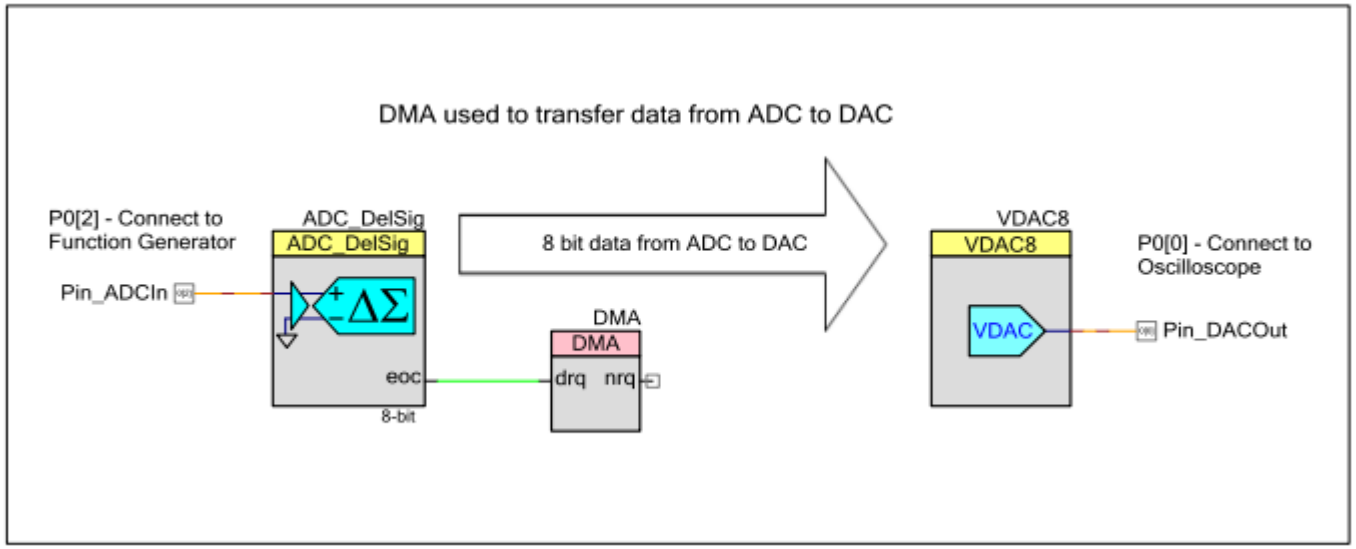
Приклад 1: Моніторинг вбудованих сенсорів
Приклад 2: Взаємодія з користувачем на недорогих пристроях

Приклад 1: Периферійні пристрої комп’ютера
Приклад 2: Автомобільні системи безпеки

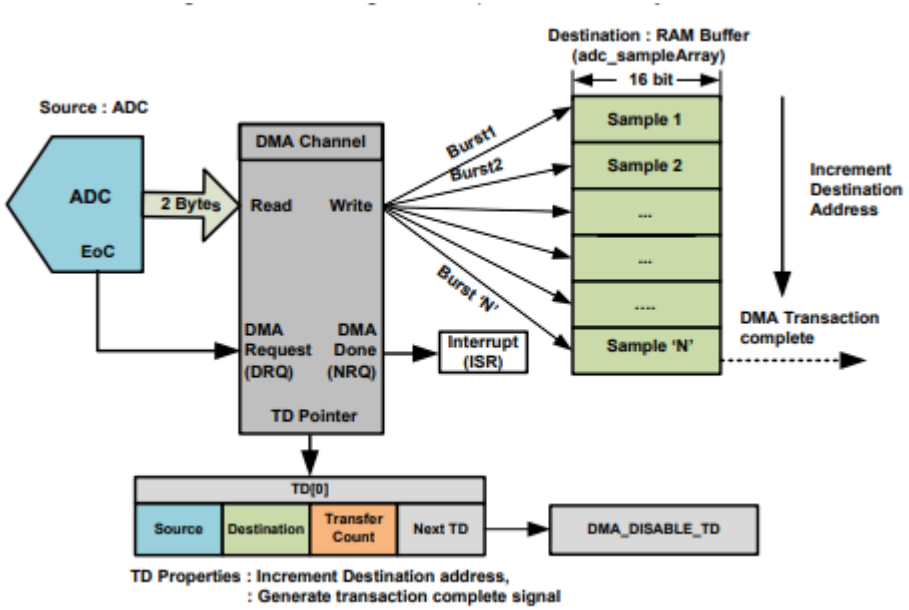
Приклад 1: Мультимедіа, відео- та звук
Приклад 2: Мережеві адаптери

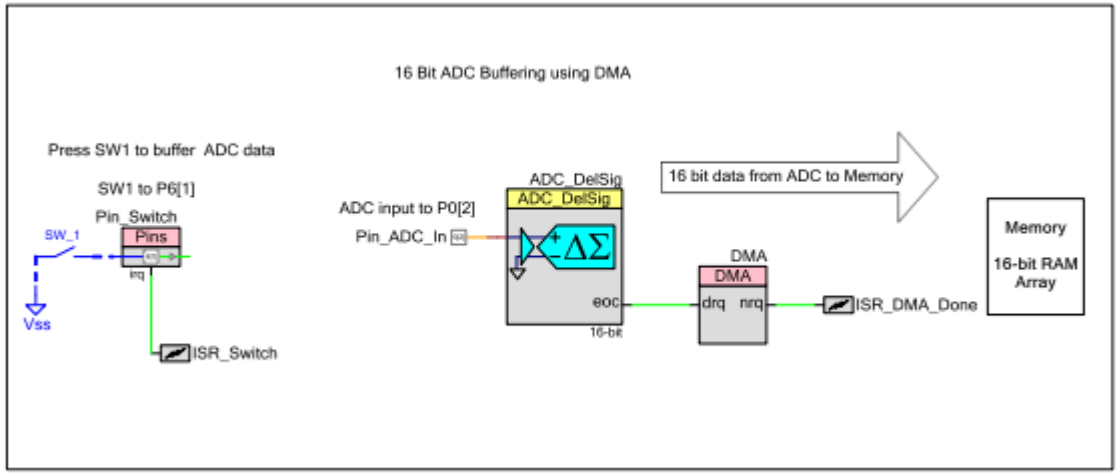
Приклади на PSoC





```
/* Визначення для конфігурації DMA */
#define DMA_BYTES_PER_BURST 1
#define DMA_REQUEST_PER_BURST 1
#define DMA_SRC_BASE (CYDEV_PERIPH_BASE)
#define DMA_DST_BASE (CYDEV_PERIPH_BASE)

/* Змінна DMA_Chan використовується для зберігання дескриптора DMA каналу */
uint8 DMA_Chan;

/* Оскільки в цьому прикладі використовується лише один TD, масив DMA_TD містить тільки один елемент */
uint8 DMA_TD[1];

/* Крок 1 - Ініціалізація DMA для обох каналів:
 * Кількість даних за один імпульс (Burst count) = 1, Кількість запитів за один імпульс (Request per burst) = 1
 * Старший байт адреси джерела = регістр даних ADC, Старший байт адреси призначення = регістру даних VDAC8
 * Змінна DMA_Chan містить дескриптор каналу, повернутий функцією 'DmaInitialize'. Він використовується для подальших звернень до каналу */
DMA_Chan = DMA_DmaInitialize(DMA_BYTES_PER_BURST, DMA_REQUEST_PER_BURST, HI16(DMA_SRC_BASE), HI16(DMA_DST_BASE));

/* Крок 2 - Виділення TD для DMA каналу:
 * DMA_TD[0] — це змінна, що містить дескриптор TD, повернутий функцією виділення TD. Він використовується для подальших звернень */
DMA_TD[0] = CyDmaTdAllocate();

/* Крок 3 */
/* Налаштування TD[0]:
 * Кількість переданих байтів = 1 (загальна кількість байтів для передачі від ADC до DAC)
 * Наступний TD = DMA_TD[0]. Один і той самий TD має повторюватися для кожного завершення перетворення ADC (EoC)
CyDmaTdSetConfiguration(DMA_TD[0], 1, DMA_TD[0], 0);

/* Крок 4 */
/* Налаштування адрес TD:
 * Адреса джерела = нижні 16 біт регістру даних ADC
 * Адреса призначення = нижні 16 біт регістру даних VDAC8 */
CyDmaTdSetAddress(DMA_TD[0], LO16((uint32)ADC_DelSig_DEC_SAMP_PTR), LO16((uint32)VDAC8_Data_PTR));

/* Крок 5 */
/* Призначення TD для DMA каналу */
CyDmaChSetInitialTd(DMA_Chan, DMA_TD[0]);

/* Крок 6 */
/* Увімкнення каналу:
 * Канал увімкнено з параметром "Preserve TD", встановленим у 1.
 * Це зберігає оригінальну конфігурацію TD та перевантажує її після завершення передачі,
 * що дозволяє повторно використовувати цей TD */
CyDmaChEnable(DMA_Chan, 1);
```





Parameter	Project Setting
Upper Source Address	HI16(CYDEV_PERIPH_BASE)
Upper Destination Address	HI16(CYDEV_SRAM_BASE)
Burst Count	2 (Two bytes)
Request Per Burst	1 (True)
Initial TD	TD[0]
Preserve TD	1 (True)

Додаткові приклади:
https://www.infineon.com/dgdl/Infineon-AN52705_PSoC_3_and_PSoC_5LP_Getting_Started_with_DMA-ApplicationNotes-v12_00-EN.pdf?fileId=8ac78c8c7cdc391c017d0724c3544966

Щиро дякую!