KURS STM32

Wojciech Olech

CZĘŚĆ II: GPIO

PORTY WEJŚCIA/WYJŚCIA

Na mikrokontrolerach STM32 porty I/O są stosunkowo mocno rozbudowane.
Konfiguracja pinów IO w STM32CubeMX odbywa się w zakładce `System Core -> GPIO`
Można też edytować funkcje pinów graficznie, klikając w pin na obrazku mikrokontrolera i wybierając jego funkcję.

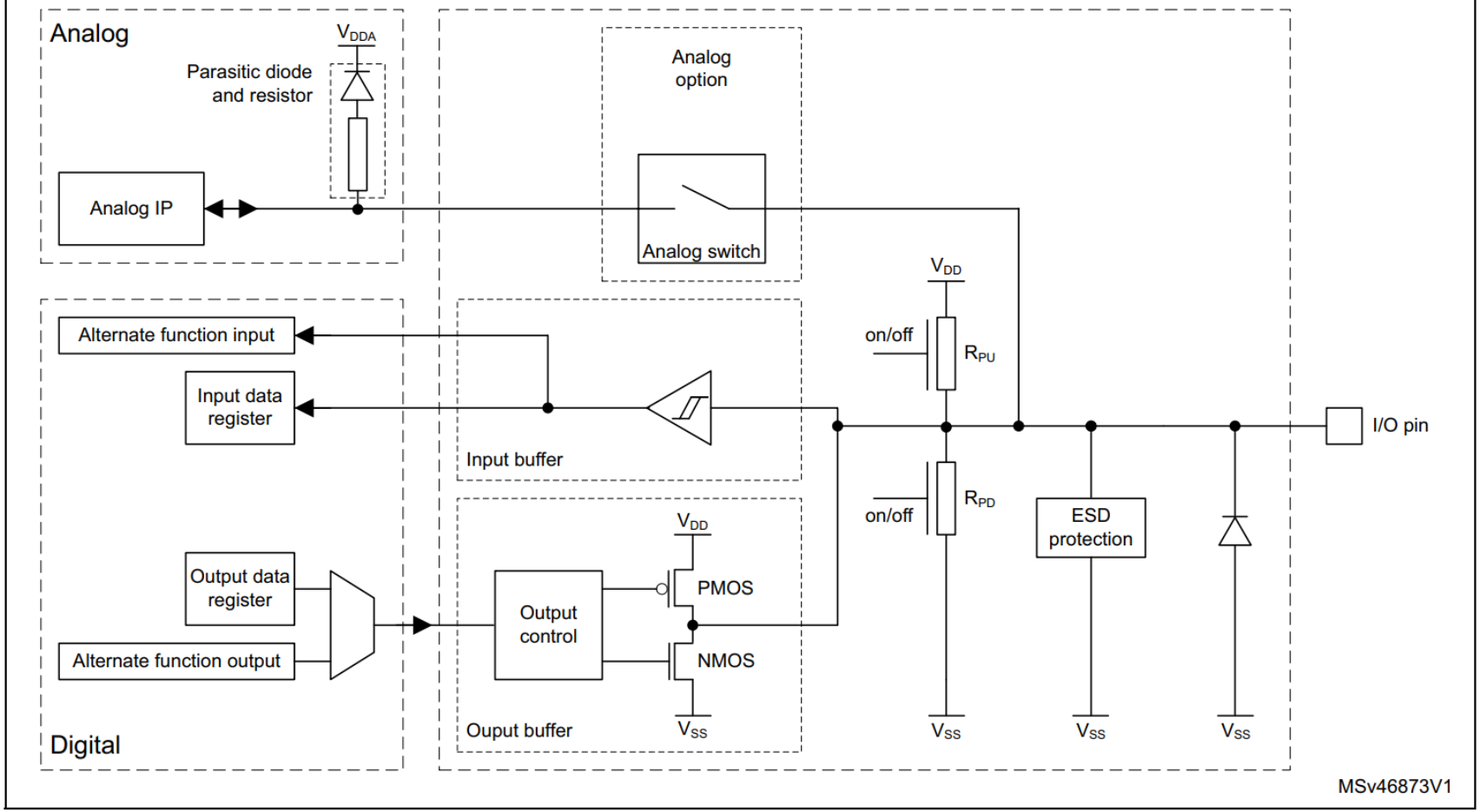
FUNKCJE PORTÓW

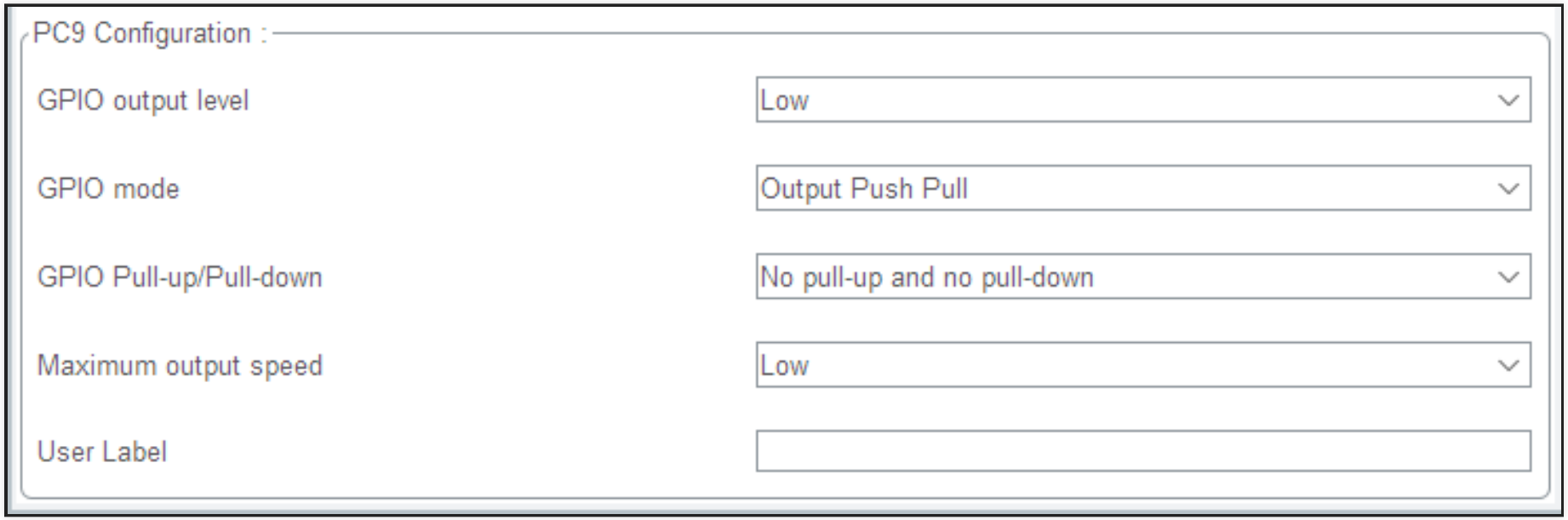
Wszystkie piny GPIO mają co najmniej cztery funkcje

- **Reset_State** - domyślny stan w którym pin jest nieużywany
- **GPIO_Input** - pin jest w trybie wejścia
- **GPIO_Output** - pin jest w trybie wyjścia
- **GPIO_EXTIn** - pin jest używany w trybie przerwań

Dodatkowo, w zależności od pinu i mikrokontrolera, mogą mieć również kilka innych funkcji (pin magistrali komunikacyjnej, pin ADC, pin przerwania, itp.)

Figure 2. Three-volt or five-volt tolerant GPIO structure (TT or FT)




PC9 Configuration :
GPIO output level
Low
GPIO mode
Output Push Pull
GPIO Pull-up/Pull-down
No pull-up and no pull-down
Maximum output speed
Low
User Label

KONFIGUROWANIE PINÓW W STM32CUBEMX

- GPIO Output Level - domyślny stan logiczny pinu
- GPIO Mode - tryb działania pinu
  - Output Push Pull - tryb wyjścia, w stanie niskm, tranzystor ściąga pin do masy. Przy stanie wysokim, tranzystor podciąga pin do Vcc (3.3V). Domyślny tryb wyjściowy.
  - Output Open Drain - tryb wyjścia, w stanie niskim pin wisi w powietrzu (jest w stanie wysokiej impedancji - tri-state, Hi-Z). W stanie wysokim, tranzystor ściąga pin do masy. Używany najczęściej w połączeniu z pull-upami.
  - Input mode - tryb wejścia
- GPIO Pull-up/Pull-down - wybór rezystora podciągającego lub jego braku. Pull-up podciąga pin do Vcc (3.3v), Pull-down podciąga pin do masy. Używane najczęściej w trybie open drain lub trybie wejściowym.
- Maximum output speed - ustawia tzw. *slew rate*, szybkość zmiany stanu pinu z wysokiego na niski i vice-versa, efektywnie zmieniając maksymalną częstotliwość z jaką pin może zmieniać swój stan. Wraz ze zwiększeniem szybkości zwiększa się też pobór prądu i szumy.
- User label - nazwa pinu, jeśli ją ustawimy to zostaną wygenerowane specjalne makra w kodzie dla naszego pinu i portu.

INICJALIZACJA PINÓW I/O

Jeśli używamy STM32CubeMXa, to wygeneruje on automatycznie kod do inicjalizacji pinów. Będzie on wyglądał mniej-więcej następująco:

```
// Struktura z informacjami na temat konfiguracji pinów
GPIO_InitTypeDef GPIO_InitStruct = {0};

// Uruchomienie zegara portu, domyślnie jest on wyłączony by oszczędzać energię.
// x - litera portu
__HAL_RCC_GPIOx_CLK_ENABLE();

// Konfiguracja domyślnego stanu pinu (opcjonalna, domyślnie piny są w stanie niskim)
// Pierwszy argument to port, x to jego litera
// Drugi argument to pin portu, y to jego numer
// Trzeci argument to stan, enumeracja GPIO_PinState: GPIO_PIN_RESET (0) lub GPIO_PIN_SET (1)
HAL_GPIO_WritePin(GPIOx, GPIO_PIN_y, GPIO_PIN_z);

// Konfiguracja pinu
GPIO_InitStruct.Pin = GPIO_PIN_y; // port
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; // tryb
GPIO_InitStruct.Pull = GPIO_NOPULL; // pull-up
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW; // szybkość pinu
// Funkcja inicjalizująca pin
HAL_GPIO_Init(GPIOx, &GPIO_InitStruct);
```

FUNKCJE DO OBSŁUGI PORTÓW I/O

`void` **`HAL_GPIO_WritePin(port, pin, stan)`** - ustawienie stanu pinu

`GPIO_PinState` **`HAL_GPIO_ReadPin(port, pin)`** - odczyt stanu pinu

`void` **`HAL_GPIO_TogglePin(port, pin)`** - zamiana stanu pinu na przeciwny

`void` **`HAL_GPIO_Init(port, init_structure)`** - inicjalizacja pinu

`void` **`HAL_GPIO_DeInit(port, pin)`** - deinicjalizacja pinu

POMOCNICZNE FUNKCJE Z HALA

`void` **`HAL_Delay(czas)`** - usypia procesor na podaną ilość milisekund

`uint32_t` **`HAL_GetTick()`** - podaje ile milisekund minęło od uruchomienia procesora

Domyślnie, przy tworzeniu projektu na Nucleo, jeśli pozwolisz Cube'owi zainicjalizować peryferia w domyślnym trybie to stworzy makra dla diód i przycisków dostępnych na płytce (na podstawie labeli GPIO). Będą one znajdować się w pliku `main.h`

Dla Nucleo-F401RE będą to między innymi makra dla przycisku (`B1_Pin`, `B1_GPIO_Port`), diody (`LD2_Pin`, `LD2_GPIO_Port`) i seriala.

Możemy takie makra tworzyć samodzielnie, nadając pinom labele w ustawieniach GPIO.

ZADANIE 1: NAPISAĆ PROGRAM KTÓRY ZAŚWIECI DIODĘ PO NACIŚNIĘCIU PRZYCISKU

Żeby odczytać stan przycisku należy użyć funkcji `HAL_GPIO_ReadPin`.

```
if (HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_SET) {
    // Zrób coś, jeśli pin jest w stanie wysokim
} else {
    // Zrób coś, jeśli pin jest w stanie niskim
}
```

Należy jednak pamiętać o dwóch rzeczach - w jaki sposób przycisk jest podłączony do pinu GPIO i w jakim trybie jest on skonfigurowany?

Pin PC13, do którego jest podpięty przycisk na płytkach Nucleo, jest domyślnie skonfigurowany jako "No pull-up and no pull-down" w **trybie przerwania na zboczu opadającym**. Oznacza to, że fizycznie pin jest podciągnięty do Vcc i jego naciśnięcie spowoduje spadek napięcia na pinie (do około 0V).

Biorąc to pod uwagę, możemy stwierdzić że żeby zaświecić diodę przy wciśnięciu przycisku należy zrobić to wtedy, kiedy stan na pinie tego przycisku jest niski.

Przykładowe rozwiązanie:

```
if (HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_SET) {
    // Zrób coś, jeśli pin jest w stanie wysokim
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
} else {
    // Zrób coś, jeśli pin jest w stanie niskim
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
}
```

ZADANIE 2: NAPISAĆ PROGRAM KTÓRY ZMIENI STAN ŚWIECENIA DIODY PO KLIKNIĘCIU PRZYCISKU

W tym przypadku, musimy sprawdzać tylko czy przycisk został wciśnięty. Dodatkowo, należy zastosować **debouncing** żeby uniknąć migania diody przy wciskaniu przycisku.

Kiedy klikamy przycisk, na jego stykach pojawiają się "mikrodrgania". Jeśli procesor takie mikrodrganie wykryje, to uważa że przycisk został naciśnięty, mimo tego że user mógł go jeszcze do końca nie wcisnąć. Z racji że procesor pracuje znacznie szybciej niż człowiek, to jest w stanie wykryć kilka takich drgnięć jeszcze przed pełnym wciśnięciem przycisku, co objawia się jako kilka kliknięć.

Debouncing w tym przypadku można uzyskać zapętlając program dopóki użytkownik nie puści przycisku, jednocześnie wstawiając malutki delay który sprawdza czy przycisk jest wciśnięty po wykryciu wciśnięcia lub drgania styków.

Przykładowe rozwiązanie:

```
if (HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET) {
    // Debouncing
    HAL_Delay(50);
    if (HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET) {
        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);

        // Czekaj, dopóki user nie zwolni przycisku
        while(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET);
    }
}
```

DODATKOWE MATERIAŁY

STM32 GPIO configuration for hardware settings and low-power consumption - dokument który opisuje szczegółowo konfigurację portów GPIO, ich możliwości oraz podaje przykłady używania ich między innymi z logiką 3.3V, 5V, sterowaniem LED, triaka i niektórych magistrali komunikacyjnych. Są tam również wzory do obliczania poziomów logicznych i rezystorów podciągających. Must-read.
https://www.st.com/content/ccc/resource/technical/document/application_note/group0/13/c0/f6/6c/29/3b/47/b3/