PROCESORY SYGNAŁOWE

LABOLATORIUM 01 Wprowadzenie do środowiska

Mateusz Kłosiński

July 6, 2025

1 Wprowadzenie do płytki STM32F746G Discovery

Płytka STM32F746G Discovery jest zaawansowanym narzedziem edukacyjnym do nauki programowania mikrokontrolerów z rodziny STM32. Wyposażona jest w mikrokontroler STM32F746NGH6 z rdzeniem ARM Cortex-M7, oferujacym wysoka wydajność i bogaty zestaw peryferiów. Płytka zawiera m.in. ekran dotykowy TFT LCD, pamieć QSPI, interfejsy USB, Ethernet oraz złacza Arduino i STMod+.

1.1 Podstawowe cechy płytki

- **Mikrokontroler**: STM32F746NGH6, 32-bitowy ARM Cortex-M7, 216 MHz.
- **Pamieć**: 1 MB Flash, 320 KB SRAM, 16 MB QSPI Flash.
- **Wyświetlacz**: 4,3-calowy ekran TFT LCD z ekranem dotykowym.
- **Interfejsy**: USB OTG, Ethernet, **SAI (audio)**, CAN, I2C, SPI, UART.
- **Debugger**: Wbudowany ST-Link/V2-1 do programowania i debugowania.

2 Praca z programem STM32CubeIDE

1. Otwórz projekt zamieszczony w folderze `LAB#01`.
2. Zbuduj projekt (`Project` → `Build Project`) i upewnij sie, że nie ma błedów kompilacji.

3. Uruchom debugowanie programu (`Run` → `Debug As` → `STM32 Cortex-M C/C++ Application`).

4. Upewnij sie, że dioda LD1 miga

3 Porównanie czasu działania operacji mnożenia sygnałów

Cel: Zaimplementuj i porównaj czas działania funkcji mnożacych dwa sygnały (tablice) w wersji całkowitoliczbowej i zmiennoprzecinkowej.

3.1 Reprezentacja danych: `int16_t` i `float` w STM32F746G

W mikrokontrolerze STM32F746G (opartym na rdzeniu ARM Cortex-M7) typ `int16_t` to 16-bitowa liczba całkowita ze znakiem. Reprezentowana jest w kodzie uzupełnień do dwóch (two's complement). Zakres możliwych wartości to od −32 768 do +32 767.

```
 0000000000001010   10
 1111111111110101   flip all bits
+0000000000000001   add 1
 1111111111110110   -10
```

Figure 1: Stałoprzecinkowa reprezentacja - kod U2

Z kolei typ `float` to liczba zmiennoprzecinkowa zgodna ze standardem IEEE-754 w formacie 32-bitowym (tzw. *single precision*). Składa sie z 1 bitu znaku, 8-bitowego wykładnika i 23-bitowej mantysy. Umożliwia reprezentowanie liczb w przybliżonym zakresie od 1.18×10^{-38} do 3.4×10^{38} z dokładnością około 6–7 cyfr znaczacych.

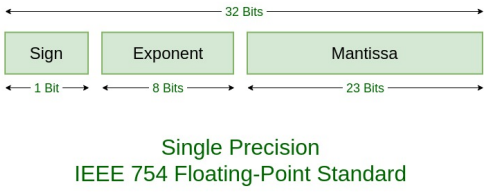

Figure 2: Zmiennoprzecinkowa reprezentacja - float

3.2 Porównanie czasów operacji na różnych typach danych

1. wstaw poniższa funkcje poza int main() w celu "przeciażenia" funkcji systemowej putchar na wysyłanie danych na port szeregowy

```
int __io_putchar(int ch)
{
    if (ch == '\n') {
        uint8_t ch2 = '\r';
        HAL_UART_Transmit(&huart1, &ch2, 1, HAL_MAX_DELAY);
    }
    HAL_UART_Transmit(&huart1, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
    return 1;
}
```

2. Zaimplementuj funkcje do pomiaru czasu działania funkcji użytkownika z użyciem timera `TIM7`. Zmierz czas przed i po wywołaniu funkcji, różnice wypisz przez UART.
3. Zaimplementuj dwie funkcje:
   - operacja MAC (Multiply accumulate) dwóch tablic typu `int16_t`.
   - operacja MAC (Multiply accumulate) dwóch tablic typu `float`.
4. Przetestuj obie funkcje na sygnałach testowych (np. długości 10000 elementów).
5. Zmierz i wypisz przez UART czas działania każdej funkcji.

3.2 Pseudokod funkcji pomiaru czasu i mnożenia sygnałów

Algorithm 1 Pseudokod dla operacji MAC z pomiarem czasu za pomoca timera sprzetowego

```
1: function InicjalizujTablice(sygnalA_int, sygnalB_int, sygnalA_float, sygnalB_float, dlugosc)
2:     for i od 0 do dlugosc - 1 do
3:         sygnalA_int[i] ← LosowaInt16(-32768, 32767)
4:         sygnalB_int[i] ← LosowaInt16(-32768, 32767)
5:         sygnalA_float[i] ← LosowaFloat(-1.0, 1.0)
6:         sygnalB_float[i] ← LosowaFloat(-1.0, 1.0)
7:     end for
8: end function
9: function MacInt16(a, b, wynik, dlugosc)
10:    for i od 0 do dlugosc - 1 do
11:        wynik[i] ← wynik[i] + a[i] × b[i]
12:    end for
13: end function
14: function MacFloat(a, b, wynik, dlugosc)
15:    for i od 0 do dlugosc - 1 do
16:        wynik[i] ← wynik[i] + a[i] × b[i]
17:    end for
18: end function
19: function InicjalizujTimer
20:    Włacz zegar TIM2 do pomiaru mikrosekund dla wywołania funkcji
21:    Ustaw preskaler TIM2, aby uzyskać 1 MHz (1 μs na tik)
22: end function
23: function main
24:    Inicjalizuj ziarno losowe
25:    InicjalizujTablice(sygnalA_int, sygnalB_int, sygnalA_float, sygnalB_float, dlugosc)
26:    InicjalizujTimer
27:    Uruchom TIM2
28:    Wyzeruj licznik TIM2
29:    MacInt16(sygnalA_int, sygnalB_int, wynik_int, dlugosc)
30:    czas_us ← Odczytaj licznik TIM2
31:    Wypisz "Czas wykonania MacInt16: ", czas_us, " μs"
32:    Wyzeruj licznik TIM2
33:    MacFloat(sygnalA_float, sygnalB_float, wynik_float, dlugosc)
34:    czas_us ← Odczytaj licznik TIM2
35:    Wypisz "Czas wykonania MacFloat: ", czas_us, " μs"
36:    Zatrzymaj i dezaktywuj TIM2
37: end function
```

Zastanówmy sie, które wywołanie funkcji okazało sie szybsze: dla liczb całkowitych (`mac_int16`) czy zmiennoprzecinkowych (`mac_float`). Na podstawie przeprowadzonych pomiarów można zauważyć, że...

3.3 Analiza przyczyn

Dlaczego jedno z nich okazało sie szybsze? Możliwe przyczyny obejmuja:

- Szybsze przetwarzanie operacji na liczbach całkowitych dzieki zoptymalizowanym jednostkom arytmetycznym.
- Dodatkowy narzut zwiazany z obsługa operacji zmiennoprzecinkowych, nawet przy właczonej jednostce FPU.

3.4 Wyłaczenie sprzetowego wsparcia obliczeń zmiennoprzecinkowych

Aby porównać czasy wykonania z wyłaczonym wsparciem sprzetowym FPU, wykonaj nastepujace kroki:

1. Otwórz ustawienia projektu: `Project -> Properties`.
2. Przejdź do: `C/C++ Build -> Settings`.
3. W sekcji `MCU GCC Compiler` znajdź opcje `Floating-Point ABI`.
4. Zmień wartość na `Software implementation`.
5. Kliknij `Apply and Close`, a nastepnie przebuduj projekt.

Po tych zmianach uruchom program ponownie i porównaj czasy z poprzednimi obliczeniami.

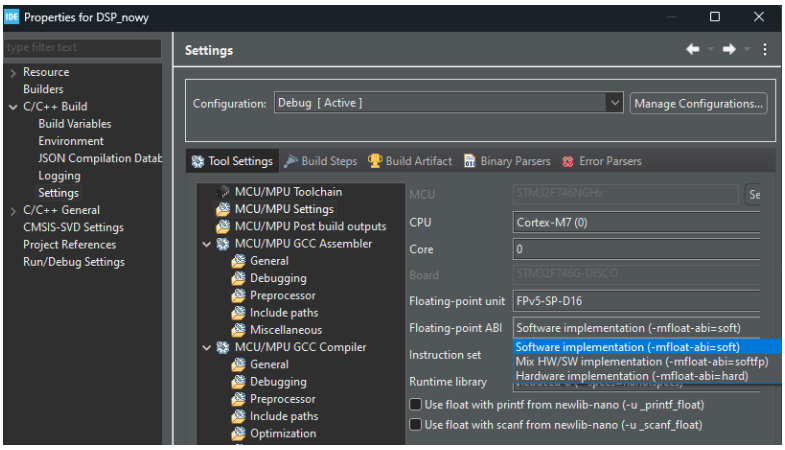

Figure 3: Wylaczenie jednostki FPU

Zgłab architekture naszego procesora STM32F746G i znajdź ile cykli zegara zajmuje mnożenie dla typów użytych w naszym kodzie, oraz czym jest Floating Point Unit i jakie daje nam korzyści

4 DO ZAPAMIETANIA

- Typy reprezentacji danych (uint8_t, uint16_t, float, double, char)
- Typy kodowania znaku (Modulo, U1, U2) czym sie różnia, jak dodać dwie liczby w systemie U2

4 ZADANIE DOMOWE

1. **Dodawanie liczb w kodzie uzupełnień do dwóch (U2):**
   Dodaj dwie liczby -125 oraz $+250$ jako 8-bitowe liczby ze znakiem zapisane w kodzie U2.
   Zadanie:

   - Zapisz obie liczby w postaci binarnej 8-bitowej (U2).
   - Wykonaj dodawanie binarne.
   - Podaj wynik w kodzie U2, a nastepnie odczytaj go jako liczbe dziesietna.
   - Sprawdź, czy wystapiło przepełnienie i uzasadnij.

   Dodaj dwie liczby **100** oraz **50** jako 8-bitowe liczby ze znakiem.
   Zadanie:

   - Przedstaw liczby jako 8-bitowe wartości(u2)
   - Dodaj je
   - Zinterpretuj wynik jako liczbe ze znakiem w kodzie U2 i podaj wartość dziesietna.
   - Wykaż, że wystapiło przepełnienie i opisz jego skutki.