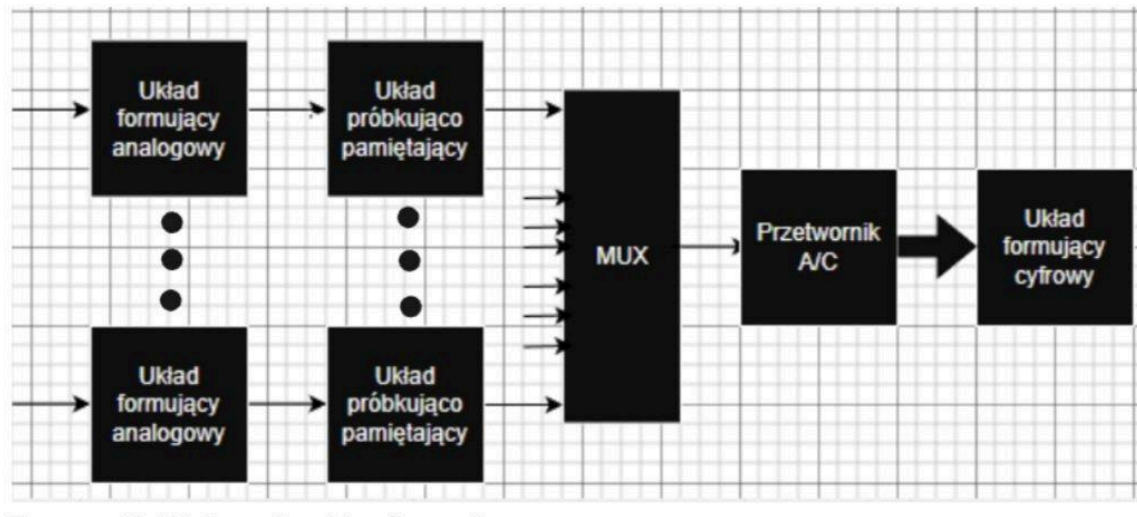


Narysować schemat funkcjonalny bloku akwizycji z mieszanym zbieraniem próbek i objaśnić etapy przetwarzania sygnałów. Porównać tryby pracy bloku i ich konsekwencje. Narysować schemat układu SH i wytłumaczyć jego działanie, tworząc wykres czasowy napięcia wyjściowego dla sygnału sinusoidalnego na wejściu. Kiedy układ SH nie musi być stosowany?



Oto skondensowana wersja „ściągą”, gotowa do szybkiego przerysowania i przepisania:

1. Schemat bloku akwizycji (mieszany/sekwencyjny):

Rysunek: Wejścia (xN) -> [Układy Dopasowujące] -> [MUX] -> [S/H] -> [ADC] -> [Bufor/uC]

Etapy:

1. **Dopasowanie:** Wzmocnienie/tłumienie i filtracja antyaliasingowa.
2. **Multipleksowanie (MUX):** Wybór jednego z wielu kanałów.
3. **Próbkowanie (S/H):** Zamrożenie napięcia na czas pomiaru.
4. **Kwantyzacja (ADC):** Zamiana napięcia na cyfrę.

Porównanie trybów:

- **Sekwencyjny (z MUX):** Tańszy (1 ADC). **Konsekwencja:** Przesunięcie czasowe między próbkami z różnych kanałów (błąd fazowy).
- **Równoległy:** Droższy (wiele ADC/SH). **Konsekwencja:** Idealna synchronizacja (brak przesunięcia).

2. Układ S/H (Sample & Hold):

Schemat: Wejście -> [Bufor] -> Klucz (Tranzystor) -> Punkt A -> [Bufor] -> Wyjście.
(W punkcie A kondensator do masy).

Działanie:

- **Sample (Próbkowanie):** Klucz zwarty -> kondensator ładuje się do napięcia wejściowego.
- **Hold (Pamiętanie):** Klucz rozwarty -> kondensator trzyma napięcie stałe dla ADC.

Wykres: Narysuj sinusoidę, a na niej „schodki”.

- Pionowe/ukośne zbocza schodka = czas *Sample* (śledzenie).
- Poziome płaskie linie = czas *Hold* (trzymanie).

3. Kiedy S/H nie jest potrzebny?

Gdy sygnał wejściowy jest bardzo wolnozmienny. Zmiana napięcia wejściowego w czasie trwania konwersji musi być mniejsza niż rozdzielczość przetwornika ($\Delta U < 1 \text{ LSB}$).

Zad. 2 (4 pkt.) Umieszczony w bloku akwizycji mikrokontroler ATmega16A odbiera z zewnętrznego układu przerwanie INT2. Program główny MAIN mikrokontrolera ma zaczynać się od adresu 0x0030, a podprogram P_INT2 od adresu 0x0400. a) Narysować mapę pamięci programu i zaznaczyć wymienione obszary programów (długość programów umowna) oraz nanieść na mapę te mnemoniki rozkazów procesora, które są niezbędne do prawidłowego działania i restartu kodu. b) Podać kod organizujący takie rozmieszczenie programów z pomocą dyrektyw .org c) Wytłumaczyć dokładnie, kiedy i jak mikrokontroler przechodzi z MAIN do P_INT2 oraz kiedy i jak wraca do MAIN. Jaką rolę pełni rejestr SP?

a) Mapa pamięci programu (Flash) ATmega16

Zgodnie z Twoją prośbą, przygotowałem mapę w formacie tekstowym. W ATmega16 wektor **RESET** znajduje się pod adresem **0x0000**, a wektor przerwania **INT2** pod adresem **0x0024**.

Plaintext		
Adres Flash	Zawartość / Mnemonik	Opis
0x0000	jmp MAIN	Wektor RESET (skok do MAIN)
...	...	(wolne/inne wektory)
0x0024	jmp P_INT2	Wektor przerwania INT2
...	...	(wolne)
0x0030	[Kod programu MAIN]	Początek programu głównego
...	...	
0x0400	[Kod programu P_INT2]	Początek podprogramu obsługi
...	reti	Powrót z przerwania

b) Kod z dyrektywami .org (Assembler)

```

Fragment kodu

.org 0x0000
    jmp MAIN          ; Skok pod adres 0x0030 po starcie/resecie

.org 0x0024           ; Adres wektora INT2 dla ATmega16
    jmp P_INT2        ; Skok do obsługi przerwania

.org 0x0030
MAIN:
    ; --- Inicjalizacja Stosu (obowiązkowa dla przerwań!) ---
    ldi r16, high(RAMEND)
    out SPH, r16
    ldi r16, low(RAMEND)
    out SPL, r16
    ; --- Tutaj reszta kodu MAIN ---
loop:
    rjmp loop         ; Pętla główna

.org 0x0400
P_INT2:
    ; --- Treść obsługi przerwania ---
    ; (np. sbi PORTA, 0)
    reti              ; KONIECZNE: Powrót i odblokowanie przerwań

```

c) Mechanizm przerwania i rola rejestru SP

Kiedy i jak następuje przejście z MAIN do P_INT2?

1. **Warunki:** Na pinie INT2 pojawi się odpowiedni sygnał (zbocze/poziom), a bit globalny przerwań (I w rejestrze SREG) oraz bit maski INT2 są ustawione.
2. **Akcja:** Mikrokontroler kończy obecnie wykonywaną instrukcję w **MAIN**.

3. **Zatwierdzenie:** Na **Stos (SRAM)** zostaje automatycznie odłożony aktualny licznik programu (PC - Program Counter), czyli adres instrukcji, do której procesor ma wrócić.
4. **Skok:** Licznik programu (PC) zostaje załadowany adresem wektora przerwania (`0x0024`), skąd instrukcja `jmp P_INT2` kieruje procesor pod adres `0x0400`.

Kiedy i jak następuje powrót?

1. Następuje po wykonaniu instrukcji **RETI** (Return from Interrupt).
2. Procesor pobiera (zdejmuje) ze Stosu zapisany wcześniej adres powrotu i ładuje go z powrotem do licznika programu (PC).
3. Automatycznie włączana jest globalna flaga przerw (bit I), co pozwala na obsługę kolejnych zdarzeń.

Rola rejestru SP (Stack Pointer - Wskaźnik Stosu):

- **Przechowywanie adresu:** SP wskazuje na ostatnie wolne miejsce w pamięci SRAM (stos), gdzie można zapisać dane.
- **Ratowanie kontekstu:** Jest kluczowy przy przerwaniach, ponieważ to dzięki niemu procesor „pamięta”, gdzie przerwał pracę w programie głównym (przechowuje adres powrotu). Bez zainicjalizowanego SP przerwanie spowoduje zawieszenie się mikrokontrolera.

1. Maksymalna wartość sumy w systemie BIN

Liczba 10-bitowa bez znaku ma zakres od $0\$$ do $2^{10}-1\$$ (czyli $0\$$ do $1023\$$).

- **Wartość dziesiętna:** $1023 + 1023 = 2046\$$.
- **Wartość binarna (11 bitów):** `111 1111 1110`

2. Algorytm działania

1. **Inicjalizacja:** Ustawienie rejestrów kierunku (DDR) dla Portu A i Portu C jako wyjścia.
2. **Kopiowanie danych:** Przepisanie wartości z $\$R17:\$R16$ oraz $\$R19:\$R18$ do innych rejestrów roboczych (np. $\$R21:\$R20$ i $\$R23:\$R22$), aby nie zniszczyć danych wejściowych.
3. **Sumowanie:** Dodanie młodszych bajtów ($\$ADD\$$), a następnie dodanie starszych bajtów wraz z przeniesieniem ($\$ADC\$$).
4. **Wyliczenie średniej:** Ponieważ mamy dwie próbki, średnia arytmetyczna to suma podzielona przez 2. W systemie binarnym odpowiada to **przesunięciu bitowemu w prawo (LSR/ROR)** o jedną pozycję.

5. **Przekazanie wyniku:** Wysłanie otrzymanych bajtów wyniku na rejestry wyjściowe **PORTA** (młodszy bajt) i **PORTC** (starszy bajt).

3. Kod podprogramu (ASM ATmega16A)

Fragment kodu

```
; --- Inicjalizacja portów (część konfiguracji) ---
ldi r20, 0xFF      ; Ładujemy samymi jedynkami
out DDRA, r20      ; Port A jako wyjście (wynik LSB)
out DDRC, r20      ; Port C jako wyjście (wynik MSB)

; --- Podprogram obliczający średnią ---
CALC_AVERAGE:
    ; Kopiujemy dane do rejestrów roboczych, by nie nadpisać wejścia
    mov r20, r16    ; Kopia LSB próbki 1
    mov r21, r17    ; Kopia MSB próbki 1
    mov r22, r18    ; Kopia LSB próbki 2
    mov r23, r19    ; Kopia MSB próbki 2

    ; Sumowanie (Wynik w R21:R20)
    add r20, r22     ; Dodaj młodsze bajty
    adc r21, r23     ; Dodaj starsze bajty + Carry (przeniesienie)

    ; Dzielenie przez 2 (Przesunięcie logiczne w prawo)
    lsr r21          ; Przesuń starszy bajt (bit 0 trafia do flagi C)
    ror r20          ; Przesuń młodszy bajt uwzględniając bit z flagi C

    ; Wysłanie wyniku na porty
    out PORTA, r20   ; Młodsze 8 bitów średniej na Port A
    out PORTC, r21   ; Pozostałe bity (MSB) na Port C

    ret             ; Powrót z podprogramu
```

Wyjaśnienie kluczowych instrukcji:

- **adc (Add with Carry):** Niezbędne przy sumowaniu liczb wielobajtowych – uwzględnia fakt, że suma młodszych bajtów mogła przekroczyć 255.
- **lsr i ror:** Duet tych instrukcji pozwala „przepchnąć” najmłodszy bit starszego bajtu do pozycji najstarszego bitu młodszego bajtu. Dzięki temu dzielimy całą liczbę 16-bitową przez 2.
- **Bezpieczeństwo danych:** Dzięki użyciu **mov** na początku, oryginalne wartości w **R16** – **R19** pozostają niezmienione, co było wymogiem zadania.

