## Лекція 9

### 1. Таймери

МК AVR мають на борту різні реалізації таймерів. Перш за все це 8-ми та 16-ти розрядні таймери/лічильники. Останні, відповідно, можуть відраховувати точніші та довші інтервали часу. Також присутні в моделях AVR і сторожові таймери (Watchdog Timer), що призначені для контролю за несанкціонованим зациклюванням програми.

**8-розрядні таймери/лічильники.** У МК AVR реалізовано три виконання 8-розрядних таймерів/лічильників, що відрізняються набором виконуваних функцій. Перше виконання має реалізовані лише восьмирозрядний лічильник та лічильник зовнішніх подій. Друге виконання має додатково 8-розрядний ШІМ (широтно-імпульсний генератор) та формувач сигналів. Третє виконання ще додає керування таймером/лічильником в асинхронному режимі (годинник реального часу). У моделях AVR можуть бути присутні 2 восьмирозрядні таймери, які позначаються відповідно T0 та T2. Кількість таймерів та їхнє виконання залежать від конкретної моделі. Також від виконання залежить і кількість переривань, що можуть бути згенеровані таймером.


Рис. 1. Регістри вводу/виводу 8-розрядного таймера T0 ATmega32A

Як 8- так і 16 -розрядні таймери AVR використовують у своїй роботі 10-розрядний **попередній подільник частоти** (рис. 1). Він використовується для всіх наявних таймерів в моделі МК, окрім таймера з керуванням в асинхронному режимі, який має свій окремий попередній подільник.

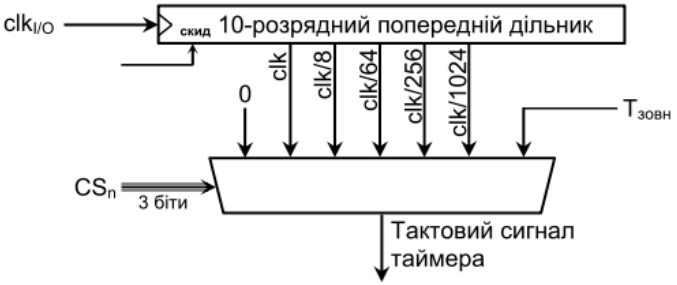



Рис. 2. Блок попереднього дільника таймера

На виході мультиплексора дільника частоти вхідна тактова частота, від якої тактується МК, розбита на ряд підчастот, які зменшені у вказане число раз. Наприклад, clk/64 означає, що відраховується 64 тактів МК, після чого посилається імпульс для таймера. Таким чином, ми можемо сповільнити роботу нашого таймера.

Варто мати на увазі, що попередній подільник працює незалежно від таймера. У результаті чого виникає невизначений проміжок часу (від 1 до значення дільника) між дозволом таймера та його першим відліком при сумісній роботі з дільником частоти. Для цього передбачена можливість скиду значень попереднього дільника частоти шляхом запису лог. 1 у відповідний біт регістра SFIOR.

Таймери, що не мають асинхронного режиму роботи, можуть тактуватися також від зовнішнього сигналу $T_{зовн}$. При цьому можна налаштувати відлік або по наростанню фронту сигналу , або по його спаду. Частота зовнішнього сигналу повинна бути у 2,5 рази меншою за частоту тактового генератора МК.

Попередній подільник частоти для асинхронного таймера може тактуватися як від основного генератора тактової частоти МК, так і від автономного генератора із своїм зовнішнім кварцем. Цей автономний генератор оптимізований під годинниковий кварц 32 768 Гц. Асинхронний подільник має додаткові коефіцієнти поділу 32 та 128, однак відсутній підрахунок від зовнішніх імпульсів.

**Вибір режиму тактування** таймера здійснюється за допомогою трьох бітів $CS_n$, що розміщені у регістрі $TCCR_n$ (табл. 1).

Таблиця 1. Вибір тактового сигналу для таймерів виконання 1, 2

| $CS_n2$ | $CS_n1$ | $CS_n0$ | Джерело тактового сигналу |
|---|---|---|---|
| 0 | 0 | 0 | Таймер зупинений |
| 0 | 0 | 1 | $clk_{I/O}$ |
| 0 | 1 | 0 | $clk_{I/O}/8$ |
| 0 | 1 | 1 | $clk_{I/O}/64$ |
| 1 | 0 | 0 | $clk_{I/O}/256$ |
| 1 | 0 | 1 | $clk_{I/O}/1024$ |
| 1 | 1 | 0 | Рахунок імпульсів на виводі $T_n$ за спадаючим фронтом |
| 1 | 1 | 1 | Рахунок імпульсів на виводі $T_n$ за наростаючим фронтом |

Таймери/лічильники можуть працювати у **4-х режимах**:

– Нормальний режим. 8-розрядний лічильний регістр нараховує 256 тактів таймера, виникає переповнення, і рахунок продовжується зі значення 0. При виникненні пере-повнення виставляється у регістрі TIFR прапор переповнення TOV та генерується переривання (якщо це переривання дозволене у регістрі TIMSK прапорцем TOIE, а також вистав-лений загальний дозвіл у регістрі стану SREG).

– Скид при співпадінні. У цьому режимі наш лічильний регістр здійснює рахунок тактів таймера до вказаного 8-роз-рядного значення у регістрі порівняння OCR, після чого скидається у значення 0. Під час обнулення лічильного регістра виставляється прапор співпадіння OCF та генерується переривання, якщо звісно є загальний дозвіл на переривання у SREG та виставлений прапор дозволу OCIE у регістрі TIMSK.

– Швидкий ШІМ.

– ШІМ з точною фазою.

Деталі останніх 2-х режимів будуть розкриті пізніше.


Рис. 3. Переривання таймера за значенням лічильника

Таблиця 2. Режими роботи 8-розрядних таймерів

| $WGM_n1$ | $WGM_n0$ | Режими роботи таймера |
|---|---|---|
| 0 | 0 | Нормальний ( по переповненню) |
| 0 | 1 | ШІМ з точною фазою |
| 1 | 0 | Скид при співпадінні |
| 1 | 1 | Швидкий ШІМ |

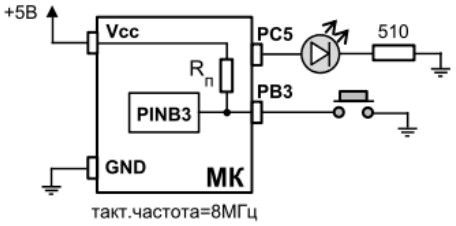
Для нормального режиму та скиду при співпадінні можемо налаштувати поведінку виводу таймера $OC_n$ (табл. 3).

Таблиця 3. Керування виводом ОС у норм. реж. та скиду при співп.

| $COM_n1$ | $COM_n0$ | Поведінка |
|---|---|---|
| 0 | 0 | Таймер відключений від виводу ОС |
| 0 | 1 | Стан виводу змінюється на протилежний |
| 1 | 0 | Вивід скидується в «0» |
| 1 | 1 | Вивід скидується в «1» |

При необхідності стан виводу $OC_n$ може бути змінений примусово, шляхом запису лог. 1 у розряд $FOC_n$ регістра $TCCR_n$. Перери-вання при цьому не генерується.

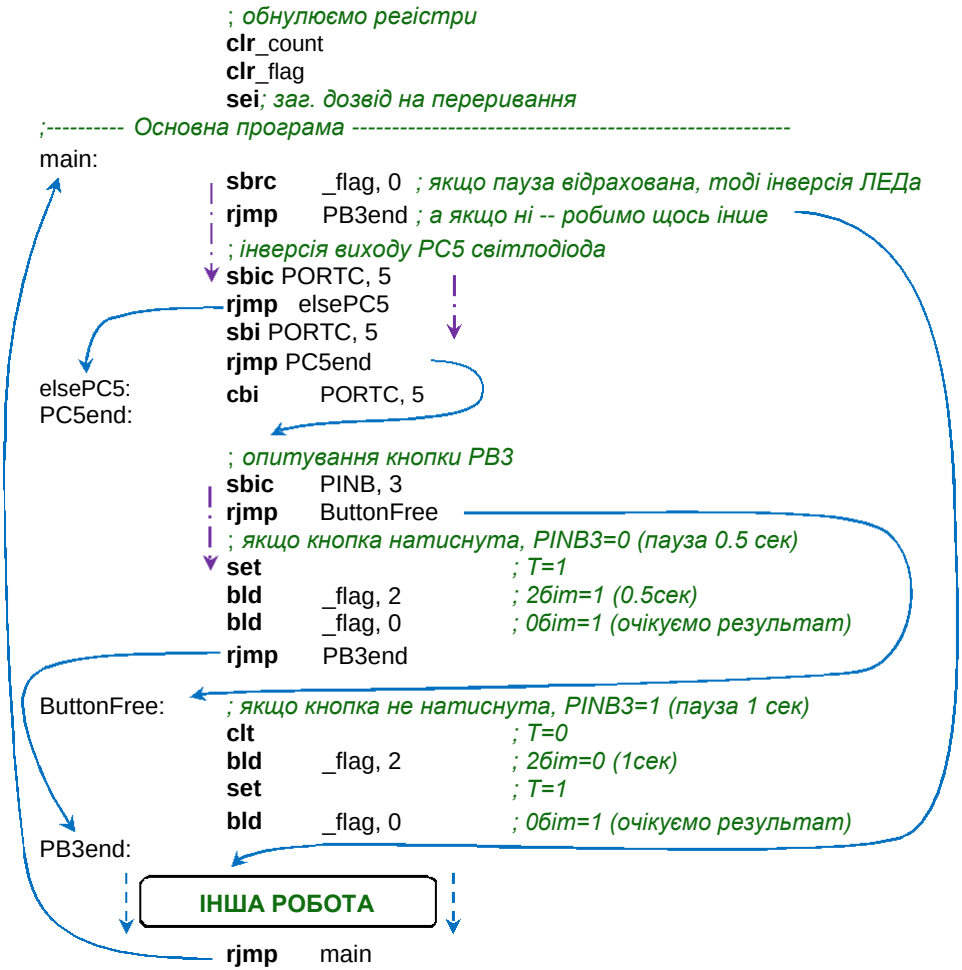
**Приклад 1.** Використовуючи таймер 0, реалізувати блимаючий світлодіод. У пасивному режимі світлодіод блимає з інтервалом приблизно 1 сек., тобто 1 сек. світиться, а на 1 сек. гасне. При натисненні кнопки, світлодіод починає блимати у 2 рази частіше.




При максимальному значенні коефіцієнта попереднього подільника частоти рівному 1024 та тактовій частоті 8 МГц, таймер 0 може відраховувати максимальні інтервали часу 32.768 мсек. Тому для реалізації заданих інтервалів (1 та 0,5 сек) необхідно буде використовувати певне число послідовних відліків часу таймера 0. Для точності інтервалів налаштуємо наш таймер 0 у режим «скид при співпадінні» на час 25 мсек.

```
.include "m32Adef.inc"
.def    _flag  = r0         ; 0біт - пауза =0 (пораховано), =1(ще рахує)
                            ; 2біт - тривалість паузи =0(1сек), =1(0.5сек)
.def    _count = r21
                .CSEG
                .org    $000
                jmp     RESET           ; Reset Handler
                .org    $014
                jmp     TIM0_COMP       ; Timer0 Compare Handler
                .org    $028
                reti                    ;Store Program Memory Ready Handler
;підпрограма переривання по співпадінню Таймера 0 ----------
TIM0_COMP:      sbrs    _flag, 0        ; якщо відлік дозволений
                rjmp    Task0end
                inc     _count          ; _count++
                sbrs    _flag, 2        ; якщо 2біт=1, тоді рахуємо до 20
                rjmp    sec1
                cpi     _count, 20      ; порівнюємо з числом 20
                brne    Task0end        ; на кінець, якщо не рівне
                rjmp    ready
sec1:           cpi     _count, 40      ; порівнюємо з числом 40
                brne    Task0end        ; на кінець, якщо не рівне
ready:          clt                     ; T=0
                bld     _flag, 0        ; 0біт=1 (пауза порахована)
                clr     _count          ; _count=0
Task0end:
T0end:          reti

;-----------------------------------------------------------------------------------
RESET:          ldi     r16, Low(RAMEND)
                out     SPL, r16
                ldi     r16, High(RAMEND)
                out     SPH, r16
                ldi     r16, 0x00
                ldi     r17, 0xFF
                out     DDRB, r16       ; порт B на вхід
                out     PORTB, r17
                out     DDRC, r17       ; порт C на вихід
                out     PORTC, r16
; таймер_0 скид по співпадінню, 25msec, Prescaler=1024, OCR=0xC2
                ldir16, (1<<WGM01)|(1<<CS02)|(1<<CS00)
                out     TCCR0, r16      ; OCR=0xC2
                ldi     r16, 0xC2
                out     OCR0, r16       ; OCR=0xC2
                ldi     r16, (1<<OCIE0)
                out     TIMSK, r16      ; дозвіл на перер. по співпадінню
```

Ініціалізація стеку

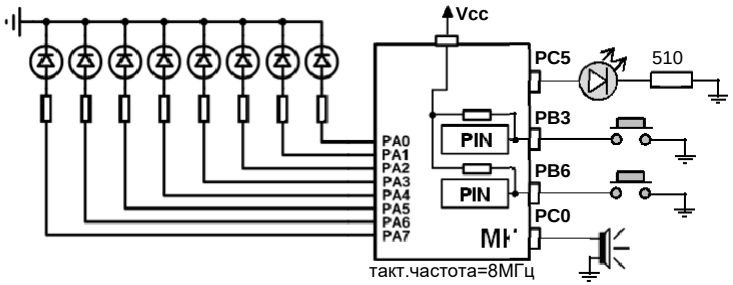



**Опис програми.** Використання таймера дає нам можливість одночасно відраховувати потрібні інтервали часу та виконувати іншу додаткову роботу. Програмна логіка для відліку пауз між блиманнями світлодіода реалізована на логічних прапорцях (окремих бітах) регістра загального призначення R0 (псевдонім _flag) . На початку основної програми здійснюємо перевірку нульового біта R0. Якщо він скинутий (=0), тоді здійснюємо інверсію виводу МК, до якого підключений світлодіод. Далі опитуємо ногу МК, до якої підключена кнопка. Якщо кнопка не натиснута, тоді на вході логічна «1» (на початку програми ми ініціалізували цей порт на вхід з підтягуючими

резисторами). Далі скидаємо 2 біт R0 в логічний «0» (вказівка для таймера відрахувати 1 сек), та встановлюємо в логічну «1» нульовий біт (очікування).

Задані інтервали часу відраховуються у підпрограмі переривання таймера 0 по співпадінню. Співпадіння налаштовані на час 25 мсек. Відповідно, для інтервалу в 1 сек. необхідно відрахувати 40 таких переривань таймера 0, а для 0,5 сек. – 20 переривань. Відлік кількості переривань ведеться у регістрі R21 (псевдонім _count), і на кожному перериванні виконується порівняння із заданою константою.

**Приклад 2.** Для попереднього прикладу додамо ще одну задачу, яка буде працювати паралельно з основною. Для цього приєднаємо ще одну кнопку PB6, а на порт A вісім світлодіодів. При натиску кнопки регістр порту A збільшуватиметься на 1. Значення регістра буде відображатися у бінарному вигляді на лінійці світлодіодів. При кожному натиску кнопки звуковипромінювач, під'єднаний до виводу PC0, має видавати короткий сигнал.




Задачу підрахунку кількості натискань кнопки розмістимо у програмний блок «інша робота» попереднього програмного лістингу, а також додамо невеликі фрагменти коду у сегменти макровизначень, ініціалізації та у підпрограму переривань таймера 0.

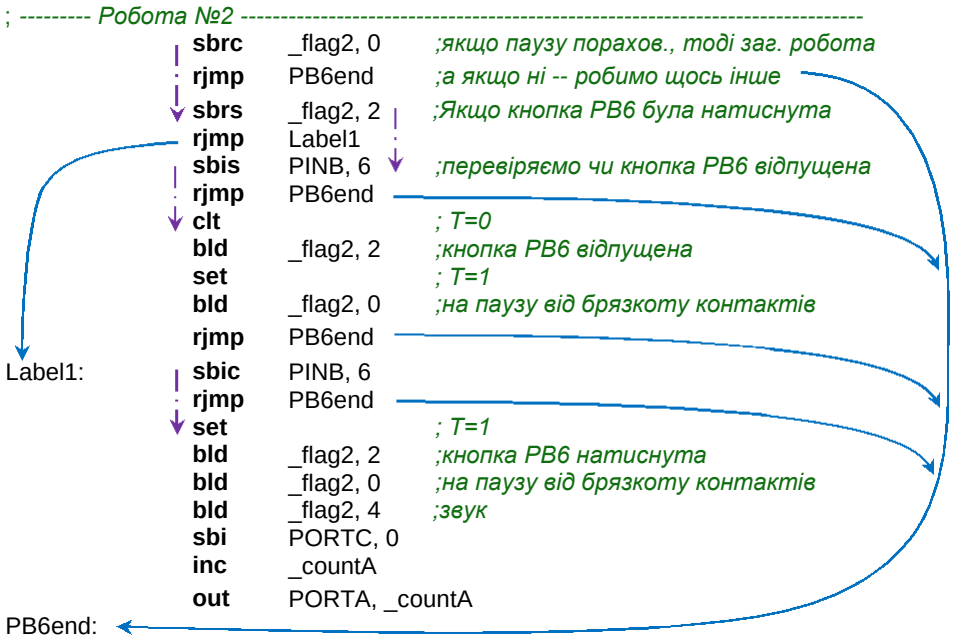
У сегменті з макровизначеннями додамо:

```
.def    _flag2 = r1   ; 0біт - пауза =0 (пораховано), =1(ще рахує)
                      ; 2біт - стан кнопки PB6 =0(відпущена), =1(натиснута)
                      ; 4біт - звуковипромінювач =0(тихо), =1(пищить)
.def    _count2= r22
.def    _countA= r23
.def    _countBuz = r24
```

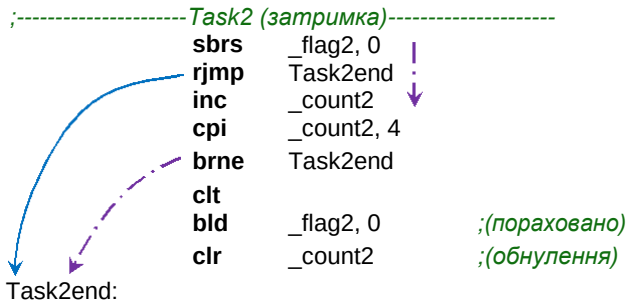
У блок ініціалізації периферії:

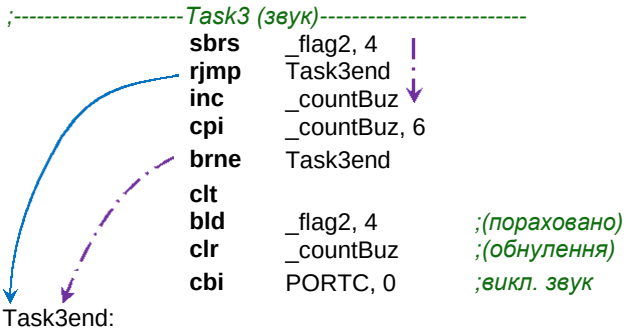
```
out     DDRA, r17
out     PORTA, r16

clr     _count2
clr     _countA
clr     _countBuz
clr     _flag2
```

У сегменті основної програми у блоці «інша робота»:




У кінець підпрограми таймера перед командою виходу reti додамо такий код:






Таким чином, наші задачі можуть паралельно працювати незалежно одна від одної, відраховуючи необхідні інтервали часу та використовуючи при цьому лише один таймер.

**2. Використання асинхронних таймерів.**

МК сімейства Mega у більшості випадків також мають в себе на борту і таймер/лічильник третього виконання, що може працювати в асинхронному режимі, тобто у якості годинника реального часу. Тактовий генератор такого таймера може працювати від окремого зовнішнього кварцу, підключеного до виводів TOSC1 та TOSC1, або від зовнішнього сигналу, що подається на вивід TOSC1. Хоча тактовий генератор таймера і налаштований на частоту 32 768 Гц, але частота кварцового резонатора чи зовнішнього сигналу може знаходитися у межах 0…256 кГц.

Для роботи таймера в асинхронному режимі призначений регістр ASSR. Для переведення таймера в асинхронний режим необхідно встановити біт $AS_n$ в 1. Регістр ASSR має ще три біти, що призначені для контролю стану обновлення регістрів таймера при переключенні в асинхронний режим. У табл. 4 наведено режими тактування асинхронного таймера. Решта налаштувань такі ж, як у звичайних 8-розрядних таймерах.

При початковій ініціалізації таймера в асинхронному режимі слід дотримуватися рекомендованої виробником процедури:

- витримати 1 секунду паузи для запуску тактового генератора таймера/лічильника;
- заборонити переривання від таймера/лічильника;
- переключити його в асинхронний режим;
- записати необхідні значення в регістри $TCNT_n$, $OCR_n$ і $TCCR_n$.

- очікувати, поки не будуть скинуті прапорці $TCN_nUB$, $OCR_nUB$, $TCR_nUB$ регістру ASSR.
- скинути прапорці переривань таймера/лічильника;
- дозволити переривання.

Таблиця 4. Вибір тактового сигналу для таймерів виконання 3

| $CS_n2$ | $CS_n1$ | $CS_n0$ | **Джерело тактового сигналу** |
|---|---|---|---|
| 0 | 0 | 0 | Таймер зупинений |
| 0 | 0 | 1 | clk |
| 0 | 1 | 0 | clk/8 |
| 0 | 1 | 1 | clk/32 |
| 1 | 0 | 0 | clk/64 |
| 1 | 0 | 1 | clk/128 |
| 1 | 1 | 0 | clk/256 |
| 1 | 1 | 1 | clk/1024 |

** Вибір джерела тактування залежить від значення біта $AS_n$*

**Приклад.** Для моделі ATmega32 на основі асинхронного таймера T2 з кварцовим резонатором 32 768 Гц реалізувати секундомір з відліком секунд та хвилин, та їхнім відображенням на 4-х семисегментних елементах зі спільним анодом.

Відлік часу секундоміра реалізується згідно блок-схеми, зображеної на рис. 4.

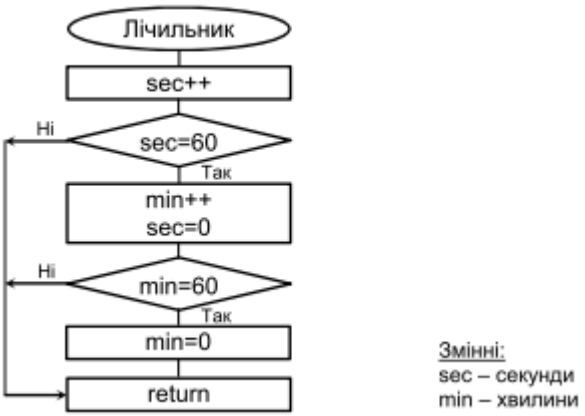



Рис. 4. Блок-схема підпрограми відліку часу

На рис. 5 зображена принципова схема для нашого прикладу. Для керування напругою на спільних анодах семисегментних елементів використовуються ключі на основі pnp-транзисторів. Для спрощення симуляції в пакеті Proteus ці ключі замінюємо на логічні інвертори.

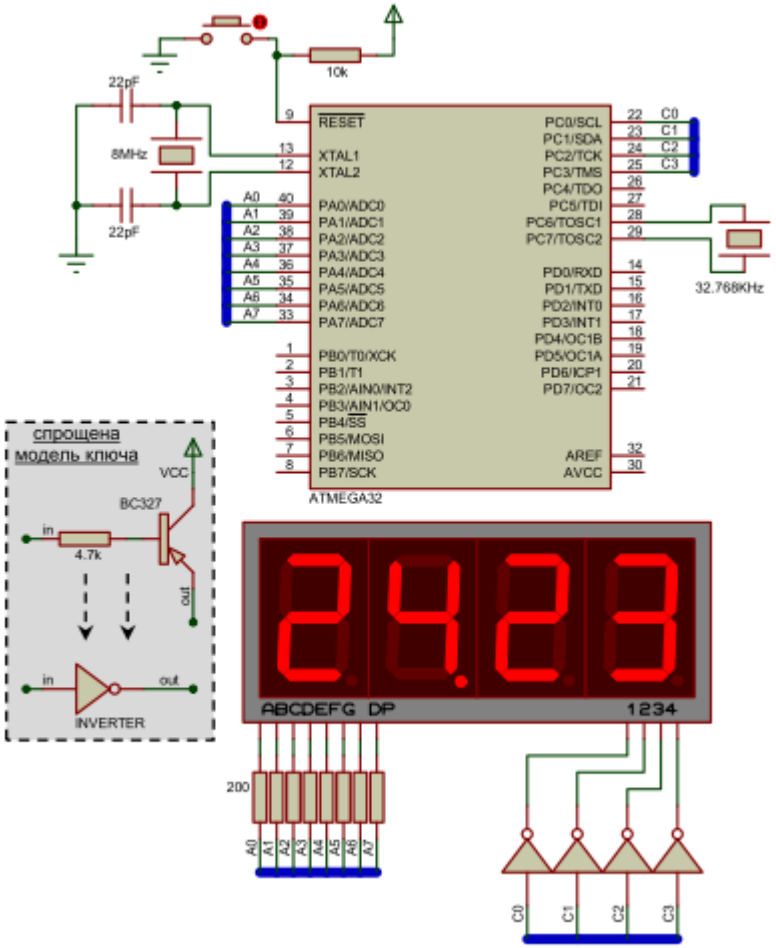


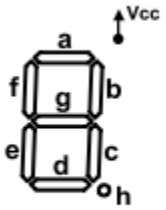

Рис. 5. Принципова схема секундоміра

Для відображення чисел на нашому 4-х елементному семисегментнику ми задіяли порт A для засвічування окремих сегментів та половину порту C для керування ключами при подачі напруги живлення на елементи.

Наш дисплей працює в режимі динамічної індикації, суть якої полягає у тому, що семисегментні елементи

засвічуються по черзі, тобто в конкретний момент часу світиться лише один семисегментний елемент. Для ефекту постійного свічення, не відчутного для людського ока, кожен елемент повинен засвічуватися з частотою не нижче 60-70 Гц. Оскільки у нас 4 елементи, то загальна частота перемикання ключів подачі живлення на семисегментні елементи повинна складати близько 70×4=280 Гц. Задачу перемикання ключів живлення та подачу байта числа, що має засвічуватися, для кожного семисегментного елемента реалізуємо у підпрограмі переривання таймера T0.

Для відображення чисел на семисегментних елементах потрібно скласти таблицю відповідностей, аналогічну табл. 4. Для подачі напруги живлення Vcc на семисегментний елемент слід подати на ключ (pnp-транзистор) низький рівень (лог. «0») з відповідного виводу порту C. Далі, для засвічування сегментів, на їхні виводи катодів через обмежувальні резистори подаються низькі рівні з виводів порту A.




| Зобр. | | h | g | f | e | d | c | b | a | 10-знач. | 16-знач. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0b | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 192 | C0 |
| **1** | 0b | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 249 | F9 |
| **2** | 0b | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 164 | A4 |
| **3** | 0b | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 176 | B0 |
| **4** | 0b | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 153 | 99 |
| **5** | 0b | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 146 | 92 |
| **6** | 0b | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 130 | 82 |
| **7** | 0b | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 248 | F8 |
| **8** | 0b | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 128 | 80 |
| **9** | 0b | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 144 | 90 |

Рис. 6. Кодування семисегментного індикатора зі спільним анодом

Розрахуємо значення для ініціалізації наших таймерів.

**T2** (асинхр. таймер) для відліку інтервалів часу 1 сек.:

$$f_{сигн.T2} = \frac{f_{такт.генер.T2}}{256 \cdot K_{подільнT2}}$$

$$f_{подільнT2} = \frac{f_{такт.генер.T2}}{256 \cdot K_{сигн..T2}} = \frac{32768\ \ Гц}{256 \cdot 1\ \ Гц} = 128$$

T0 для задання частоти переключення ключів живлення:

$$f_{сигн.T0} = \frac{f_{такт.генер.T0}}{(OCR0+1) \cdot K_{подільнT0}}$$

$$OCR0 = \frac{f_{такт.генер.T0}}{K_{подільнT0} \cdot K_{сигн..T0}} - 1 = \frac{8\ \ МГц}{1024 \cdot 280\ \ Гц} - 1 \approx 27 = 0x1B$$

```
.include "m32def.inc"
;Імена для регістрів загального призначення-------------------------------------
.def    _temp1      =r16
.def    _temp2      =r17
.def    _temp3      =r18
.def    _segment    =r18

.def    _sec        =r20
.def    _min        =r21
.def    _secL       =r22
.def    _secH       =r23
.def    _minL       =r24
.def    _minH       =r25

;Константи для сегментів дисплею ------------------------------------------------
.equ    Fig_0 = 0xC0    ; 0
.equ    Fig_1 = 0xF9    ; 1
.equ    Fig_2 = 0xA4    ; 2
.equ    Fig_3 = 0xB0    ; 3
.equ    Fig_4 = 0x99    ; 4
.equ    Fig_5 = 0x92    ; 5
.equ    Fig_6 = 0x82    ; 6
.equ    Fig_7 = 0xF8    ; 7
.equ    Fig_8 = 0x80    ; 8
.equ    Fig_9 = 0x90    ; 9

;Підпрограма Двійково-Десяткового Кодування (до 99, незапаковане)
.macro  BCD     ;@0-Bin, @1-BCDL, @2-BCDH
                mov     @1, @0          ; BCDL <- Bin
                clr     @2
BCD_1:          subi    @1, 10          ; BCDL = BCDL - 10
                brcs    BCD_2           ; якщо BCDL < 0
                inc     @2              ; BCDH++
                rjmp    BCD_1
BCD_2:          subi    @1, -10         ; BCDL = BCDL + 10

                clr     _temp1
                ldi     ZL, low(SegTable*2)  ; Z ← адреса мітки SegTable
                ldi     ZH, high(SegTable*2) ;       у байтах
                add     ZL, @1
                adc     ZH, _temp1      ; Z = Z + BCDL (зміщення)
                lpm     @1, Z           ; BCDL ← FLASH(Z)

                ldi     ZL, low(SegTable*2)  ; Z ← адреса мітки SegTable
                ldi     ZH, high(SegTable*2) ;       у байтах
                add     ZL, @2
                adc     ZH, _temp1      ; Z = Z + BCDH (зміщення)
                lpm     @2, Z           ; BCDH ← FLASH(Z)
.endmacro
```

```
                .CSEG
                .org    $000
                jmp     reset
                .org    $00A
                jmp     TIM2_OVF        ; Timer2 Overflow Handler
                .org    $014
                jmp     TIM0_COMP       ; Timer0 Compare Handler

;Переривання при співпадінні таймера T0
TIM0_COMP:      ldi     _temp1, 0xFF
                eor     _temp1, _segment        ; інверсія бітів segment
                out     PORTC, _temp1           ; вмикаємо один з ключів
                sbrc    _segment, 0             ; якщо segment.bit0<>0
                out     PORTA, _minH            ; тоді PORTA←ст. хвилини
                sbrc    _segment, 1             ; якщо segment.bit1<>0
                out     PORTA, _minL            ; тоді PORTA←мол. хвилини
                sbrc    _segment, 2             ; якщо segment.bit2<>0
                out     PORTA, _secH            ; тоді PORTA←ст. секунди
                sbrc    _segment, 3             ; якщо segment.bit3<>0
                out     PORTA, _secL            ; тоді PORTA←мол. секунди

                lsl     _segment                ; segment << 1
                cpi     _segment, (0b10000)
                brne    endCOMP                 ; якщо segment = 0b10000
                ldi     _segment,1              ; тоді segment = 0b00001
endCOMP:        reti

; Переривання по переповненню таймера T2
TIM2_OVF:       inc     _sec                    ; sec++
                cpi     _sec, 60
                brne    endOVF                  ; якщо sec <> 60
                inc     _min                    ; min++
                clr     _sec                    ; sec=0
                cpi     _min, 60
                brne    endOVF                  ; якщо min <> 60
                clr     _min                    ; min=0
endOVF:         BCD     _sec, _secL, _secH      ; розбивка сек. на 2 числа
                BCD     _min, _minL, _minH      ; розбивка хвил. на 2 числа
                andi    _minL, 0b01111111       ; засвічуємо роздільну точку

                reti                            ; між хвил. та сек.
;
reset:
                ;ініціалізація стека
                ldi     _temp1, Low(RAMEND)
                out     SPL, _temp1
                ldi     _temp1, High(RAMEND)
                out     SPH, _temp1
;Очікуємо стабілізації зовнішнього кварца 32.768 кГц ~1секунду
                call    delay1sec
```

```
                ;ініціалізація асинхронного таймера
                ldi     _temp1, 0x00
                out     TIMSK, _temp1  ;заборона переривань таймерів
                ldi     _temp1, (1<<AS2)
                out     ASSR, _temp1   ;перехід в асинхронний режим
                ldi     _temp1, (1<<CS22)|(1<<CS20)
                out     TCCR2, _temp1 ;поділ=128; по переповненню

loop:           in      _temp1, ASSR
                andi    _temp1, (1<<TCN2UB)|(1<<OCR2UB)|(1<<TCR2UB)
                brne    loop        ;очікуємо готовності асинхр. таймера T2

                ldi     _temp1, 0x00                ;скидаємо прапорці
                out     TIFR, _temp1                ;переривань таймера

                ;ініціалізація таймера T0 (~280Гц)
                ldi     _temp1, (1<<WGM01)|(1<<CS02)|(1<<CS00)
                out     TCCR0, _temp1 ;скид при співпад., подільн=1024
                ldi     _temp1, 0x1B
                out     OCR0, _temp1    ; OCR0=0x1B

                ;ініціалізація портів вводу/виводу
                ldi     _temp1, 0x00
                ldi     _temp2, 0xFF
                ;Порт A -- сегменти (катод)
                out     DDRA, _temp2                ;порт працює на вихід
                out     PORTA, _temp2       ;+5V (семисегм. не світять)
                ;Порт C -- керування живленням (анод)
                out     DDRC, _temp2                ;порт працює на вихід
                out     PORTC, _temp2      ;+5V (транзист. ключ вимкнений)

                clr     _sec                        ; sec=0
                clr     _min                        ; min=0
                BCD     _sec, _secL, _secH          ; розбивка сек. на 2 числа
                BCD     _min, _minL, _minH          ; розбивка хвил. на 2 числа
                ldi     _segment, 1
                ;дозвіл на переривання по переповн. T2 та співпадінню T0
                ldi     _temp1, (1<<TOIE2)|(1<<OCIE0)
                out     TIMSK, _temp1

                sei     ;загальний дозвіл на переривання
;основний програмний цикл
main:
                rjmp    main

;Вектор даних
SegTable: .db   Fig_0, Fig_1, Fig_2, Fig_3, Fig_4, Fig_5, Fig_6, Fig_7, Fig_8, Fig_9
        ;цифри 0       1       2       3       4       5       6       7       8       9
```

```
;Підпрограма затримки 1 секунда
delay1sec:      ldi     _temp1, 0x00
                ldi     _temp2, 0x6A
                ldi     _temp3, 0x18
sec1:           subi    _temp1, 1
                sbci    _temp2, 0
                sbci    _temp3, 0
                brne    sec1
                ret
```

### 3. ШІМ-модуляція

Широтно-імпульсна модуляція (PWM – Pulse Width Modulation) представляє собою спосіб заддання аналогового сигналу цифровим методом: змінюючи ширину (тривалість) прямокутних імпульсів сигналу визначеної частоти. Сформувати ШІМ-сигнал ми можемо, подавши на певний вивід МК у певній послідовності у часі високі та низькі рівні напруги (одиниці та нулі). Пропустивши отриманий нами ШІМ-сигнал через фільтр низьких частот, на виході фільтра отримаємо рівень напруги, що лінійно пропорційний шпаруватості (відношення тривалості періоду до тривалості імпульсу) імпульсів ШІМ. Призначення фільтра – не пропускати несучу частоту ШІМ. Фільтр може складатися з простої інтегруючої RC-ланки, або бути відсутнім взагалі, якщо кінцеве навантаження має достатню інерцію. Таким чином, маючи у розпорядженні лише два логічні рівня «1» та «0», можна отримати будь-яке проміжне значення аналогового сигналу. За допомогою ШІМ можемо керувати яскравістю освітлення (світлодіоди, лампи розжарення), швидкістю обертання двигунів та формувати звукові сигнали.

У МК AVR формування ШІМ-сигналів є однією з функцій таймерів/лічильників.

8-розрядні МК AVR можуть апаратно генерувати ШІМ-сигнали у двох режимах:

- Fast PWM – швидкодіючий ШІМ;
- Phase Correct PWM – ШІМ з точною фазою;

Режим **Fast PWM** генерує високочастотний ШІМ-сигнал у такий спосіб. Лічильник рахує від нуля до максимального значення, тобто 256, після чого лічильний регістр скидається і цикл повторюється. Якщо відповідний вивід $OC_n$ програмно підключений до таймера, тоді на початку лічби вивід встановлюється в «1» (вис. рівень), а при співпадінні лічильного регістра $TCNT_n$ та регістра порівняння $OCR_n$ вивід таймера/лічильника скидається в нуль (рис. 7 а). Також

можемо програмно встановити інверсний режим ШІМ-сигналу, тоді на початку лічби виставляється низький рівень, а при співпадінні лічильного регістра та регістра порівняння встановлюється високий рівень на виході $OC_n$.

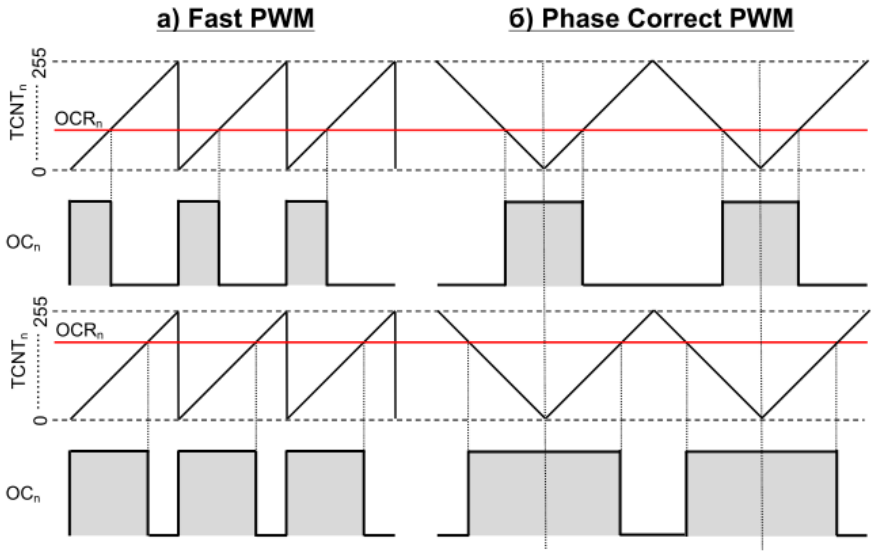



Рис. 7. Формування ШІМ-сигналів у режимах Fast та Phase Correct

Для уникнення виникнення несиметричних імпульсів сигналу на виході $OC_n$ передбачена подвійна буферизація запису у регістр $OCR_n$. Необхідне число спершу зберігається у спеціальному буферному регістрі, а вже запис у регістр порівняння $OCR_n$ здійснюється при досягненні ним значення 255.

Варто звернути увагу на випадок, коли у $OCR_n$ встановлені значення 0 та 255:

- $OCR_n$=0 – на виводі $OC_n$ при кожному 255+1 такті сигналу таймера буде спостерігатися короткий викид імпульсу. Тому мабуть краще у такому випадку відключати вивід від таймера.
- $OCR_n$=255 – вивід $OC_n$ буде встановлений у стійкий стан.

Частота ШІМ-сигналу на виводі $OC_n$ розраховується так:

$$f_{OCnPWM} = \frac{f_{такт.генер.Tn}}{256 \cdot K_{подільнTm}}$$

При 8 МГц тактового генератора МК максимально можемо отримати 31 250 Гц ШІМ-сигнал.

Також для таймера у режимі Fast PWM можуть бути запрограмовані переривання по переповненню та співпадінню. Прапорець $TOV_n$ (по переповненню) виставиться при досягненні лічильником максимального значення (255). Прапорець $OCF_n$ (по співпадінню) виставиться при рівності лічильного регістра та регістра порівняння.

Режим **Phase Correct PWM** генерує ШІМ-сигнали з фіксованою фазою. У цьому режимі лічильник спершу рахує від 0 до 255, а потім здійснює відлік у зворотному порядку від 255 до 0. Таким чином, максимальна частота вихідного сигналу буде у 2 рази меншою, аніж у режимі Fast PWM. При співпадінні лічильного регістра та регістра порівняння у прямому напрямку лічби (від 0 до 255) вихід $OC_n$ скидається у низький рівень, а при співпадінні цих регістрів у зворотному напрямку лічби (від 255 до 0) вивід $OC_n$ встановлюється у високий рівень (рис. 3.23 б). При використанні інверсного ШІМ-сигналу логіка зворотна: спершу встановлюється високий рівень, а потім скидається у низький.

Для запису значення у регістр порівняння $OCR_n$ також використовується попередня буферизація, а запис у цей регістр відбувається лише у момент досягнення лічильником значення 255.

Якщо у регістр порівняння $OCR_n$ записане значення 0 чи 255, то вивід $OC_n$ переключиться в одне зі стійких станів.

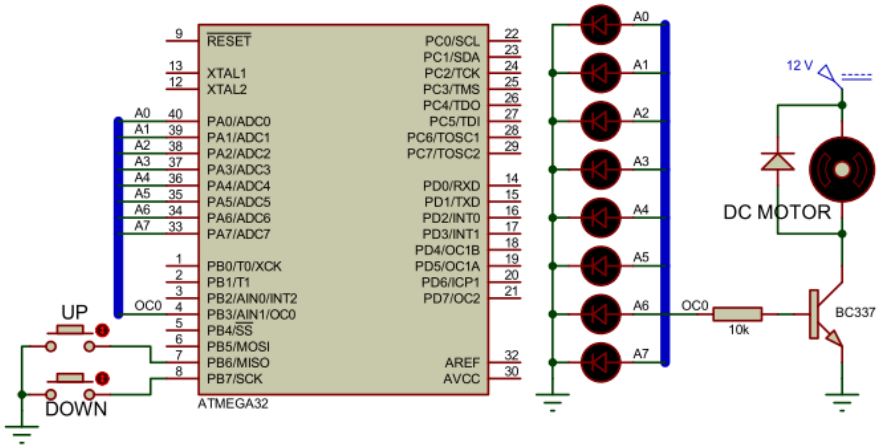
Частота ШІМ-сигналу на виводі $OC_n$ розраховується так:

$$f_{OCnPWM} = \frac{f_{такт.генер.Tn}}{256 \cdot K_{подільнТп}} \qquad f_{OCnPWM} = \frac{f_{такт.генер.Tn}}{510 \cdot K_{подільнТп}}$$

Прапорець переривання по співпадінню $OCF_n$ виставляється при кожному співпадінню лічильного регістра та регістра порівняння, а по переповненню $TOV_n$ при досягненні лічильником значення нуль.

Режим Phase Correct PWM використовується для формування багатофазних ШІМ-сигналів (центри яких співпадають у різних каналах), які часто використовуються для керування моторами.

**Приклад.** Реалізувати керування мотором постійного струму (комп'ютерним вентилятором) за допомогою ШІМ-сигналу у режимі Fast PWM з виводу $OC_0$ таймера T0. ШІМ-сигнал дискретно розбити на 8 рівнів. Збільшення чи зменшення ширини імпульсу виконувати за допомогою 2 -х кнопок, а поточне значення виводити на лінійку з 8-ми світлодіодів, підключених до порту А. Керування мотором виконується за допомогою транзисторного ключа. Для захисту транзисторного ключа паралельно з мотором підключений діод Шоткі.




** на схемі опущені обмежувальні резистори для світлодіодів*

Рис. 8. Принципова схема керування мотором постійного струму

```
.include "m32def.inc"
; Імена для регістрів загального призначення--------------------------------------
.def    _temp1      =r16
.def    _temp2      =r17
.def    _temp3      =r18
.def    _power      =r19        ; значення для виводу в OCR0
.def    _leds       =r20        ; значення індикації світлодіодів
.def    _val32      =r21        ; константа, =32

        .CSEG
        ; ініціалізація стека
        ldi             _temp1, Low(RAMEND)
        out             SPL, _temp1
        ldi             _temp1, High(RAMEND)
        out             SPH, _temp1

        ; ініціалізація портів вводу/виводу
        ldi             _temp1, 0x00
        ldi             _temp2, 0xFF
        ; Порт A працює на вихід
        out             DDRA, _temp2
        out             PORTA, _temp1

        ldi             _temp1, 0x0F
        ldi             _temp2, 0xF0
        ; Порт B – 0..3 вивід на вихід, 4..7 на вхід
        out             DDRB, _temp1
        out             PORTB, _temp2
```

```
        ; ініціалізація таймера T0 -- fast PWM; fPWM = 31 250 Гц
        ldi     _temp1, (1<<WGM01)|(1<<WGM00)|(1<<CS02)|(1<<COM01)
        out     TCCR0, _temp1       ; подільн=1
        clr     _power
        out     OCR0, _power        ;OCR0=0

        clr     _leds               ; leds = 0
        ldi     _val32, 32          ; val32 = 32

main:
B6:     ; button UP
        sbic    PINB, 6             ; якщо натиснута кнопка UP
        rjmp    B7
        add     _power, _val32      ; power =power + 32
        brcc    B6a                 ; якщо power <= 255
        ldi     _power, 255         ; power = 255
B6a:    out     OCR0, _power        ; OCR0=power
        lsl     _leds               ; leds = leds <<1
        set                         ; T=1
        bld     _leds,0             ; leds.bit0=1
        out     PORTA, _leds        ; PORTA← leds
        rcall   Pause               ; виклик підпрограми затримки

B7:     ; button DOWN
        sbic    PINB, 7             ; якщо натиснута кнопка DOWN
        rjmp    end
        sub     _power, _val32      ; power =power - 32
        brcc    B7a                 ; якщо power >= 0
        ldi     _power, 0           ; power = 0
B7a:    out     OCR0, _power        ; OCR0=power
        lsr     _leds               ; leds = leds >>1
        out     PORTA, _leds        ; PORTA← leds
        rcall   Pause               ; виклик підпрограми затримки

end:    rjmp    main

; Підпрограма затримки 0,5 сек
Pause:  ldi     _temp1, 0x00
        ldi     _temp2, 0x35
        ldi     _temp3, 0x0C

delay:  subi    _temp1, 1
        sbci    _temp2, 0
        sbci    _temp3, 0
        brne    delay
        ret
```

Оскільки ШІМ-сигнал генерується апаратно, то для логіки кнопок використовуємо просту програмну затримку.

**16-розрядні таймери/лічильники** МК AVR можуть апаратно генерувати ШІМ-сигнали у трьох режимах:

- Fast PWM – швидкодіючий ШІМ;
- Phase Correct PWM – ШІМ з точною фазою;
- Phase and Frequency Correct PWM – ШІМ з точною фазою та частотою.

На відміну від 8-ми розрядних таймерів/лічильників, 16-розрядні дають певну гнучкість та додаткову функціональність при формуванні ШІМ-сигналів.

Режими Fast PWM і Phase Correct PWM повністю ідентичні аналогічним режимам 8-ми розрядних таймерів/лічильників. Однак у них передбачена можливість генерації ШІМ-сигналів різної розрядності:

- 8 біт, лічба до 255 (0xFF);
- 9 біт, лічба до 511 (0x1FF);
- 10 біт, лічба до 1023 (0x3FF);
- від 2 до 16 біт, лічба від 3 до 0xFFFF (границя лічби фіксується в регістрі $OCR_nA$ або $ICR_n$).

При роботі з фіксованою частотою ШІМ-сигналу для змінної границі лічби рекомендується використовувати регістр захоплення $ICR_n$. Однак, якщо часто потрібно змінювати частоту генерованого сигналу, тоді рекомендується використовувати для запису граничного значення лічби регістр порівняння $OCR_nA$, оскільки регістр захоплення не має подвійної буферизації запису. Це усуває появу несиметричних імпульсів ШІМ-сигналу на виході модулятора. Однак через це регістр $OCR_nA$ не зможе бути використаний для формування ШІМ- сигналу. У цьому випадку ми можемо налаштувати вихід OCnA на генерацію сигналу меандру, записавши у розряди COMnA значення «01».

Режим Phase and Frequency Correct PWM за своєю дією подібний до режиму Phase Correct PWM. Основна його відмінність – це момент оновлення значення регістру порівняння. У цьому режимі оновлення регістру порівняння відбувається у момент досягнення лічильником значення нуля (а не максимального значення). За рахунок цього кожен період сигналу є повністю симетричним.

**Звертання до 16- розрядних регістрів.** Кожен 16-ти розрядний регістр ($TCNT_n$, $OCR_nA$, $OCR_nB$, $ICR_n$) фізично розміщується у двох 8-ми розрядних регістрах. Відповідно, для доступу до них необхідно здійснювати по дві операції запису чи читання. Для того, щоб запис чи читання обох байтів відбувався одночасно, у складі кожного 16-розрядного таймера є спеціальний 8-ми розрядний регістр TEMP, що призначений для тимчасового зберігання значення старшого байта.

Тому для запису у 16-ти розрядний регістр таймера необхідно спершу виконати запис у старший байт (який поміститься у TEMP), а потім у молодший. При запису значення у молодший, воно об'єднається зі значенням регістра TEMP, та обоє байти одночасно (протягом одного і того ж машинного циклу) запишуться у 16-розрядний регістр.

При читанні послідовність навпаки, спершу зчитується молодший байт. При його читанні значення старшого байта буде записане у регістр TEMP, і при наступному читанні старшого байта буде повернуте значення, що знаходиться у регістрі TEMP.

**4. Сторожовий таймер (WatchDog).**

Сторожовий таймер призначений для захисту мікроконтролера від збоїв та зависань у ході виконання програми. Він має незалежний тактовий генератор, який активний навіть у сплячому режимі мікроконтролера, та працює на частоті в межах 1 МГц.

Якщо сторожовий таймер включений, то через вибрані проміжки часу він виконує скид МК. Тому для нормальної роботи програми цей таймер необхідно регулярно скидати командою wdr через інтервали часу, які менші за його період. Для задання різних періодів спрацювання сторожовий таймер має попередній подільник для власного тактового генератора. Вибір коефіцієнту поділу визначаються бітами WDP у регістрі керування WDTCR.

Таблиця 5. Задання періоду сторожового таймера

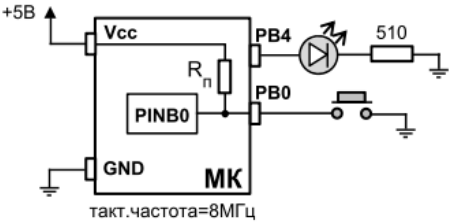
| WDP2 | WDP1 | WDP0 | Кількість тактів генератора | Період спрацювання сторож. таймера* |
|---|---|---|---|---|
| 0 | 0 | 0 | 16K (16 384) | 16,3 - 17,1 мсек |
| 0 | 0 | 1 | 32K (32 768) | 32,5 - 34,3 мсек |
| 0 | 1 | 0 | 64K (65 536) | 65,0 - 68,5 мсек |
| 0 | 1 | 1 | 128K (131 072) | 0,13 - 0,14 сек |
| 1 | 0 | 0 | 256K (262 144) | 0,26 - 0,27 сек |
| 1 | 0 | 1 | 512K (524 288) | 0,52 - 0,55 сек |
| 1 | 1 | 0 | 1024K (1 048 576) | 1,0 - 1,1 сек |
| 1 | 1 | 1 | 2048K (2 097 152) | 2,1 - 2,2 сек |

** тривалість спрацювання залежить від напруги живлення, темпер. і т.п.*

Активація сторожового таймера здійснюється бітом WDE у регістрі керування WDTCR («1» – включений).

Вимкнення сторожового таймера виконується послідовністю команд: спершу записати «1» у біти WDE та WDTOE, а потім протягом наступних 4-х машинних циклів записати «0» у біт WDE.

**Приклад.** Для демонстрації роботи сторожового таймера запрограмуємо кнопку на виконання простої програмної затримки 0,5 сек, а таймер на період спрацювання 0,26-0,27 сек. Одразу після опитування кнопки (та затримки, якщо кнопка натиснута) відбувається скид сторожового таймера та засвічується світлодіод.




```
.include "m32def.inc"
        .CSEG
        ; ініціалізація стека
        ldi     r16, Low(RAMEND)
        out     SPL, r16
        ldi     r16, High(RAMEND)
        out     SPH, r16
        ; Порт B
        ldi     r16, 0xF0
        ldi     r17, 0x0F
        out     DDRB, r16
        out     PORTB, r17
        ; ініціалізація WatchDog (0.26-0.27 sec)
        wdr; скид сторожового таймера
        ldi     r16, (1<<WDE)|(1<<WDP2)
        out     WDTCR, r16

main:   sbis    PINB, 0     ; пропустити, якщо кнопка відпущена
        rcall   Pause       ; виклик програми затримки 0,5 сек
        wdr                 ; скид сторожового таймера
        sbi     PORTB, 4    ; засвічування світлодіода
        rjmp    main
; Підпрограма затримки 0,5 сек
Pause:  ldi     r16, 0x00
        ldi     r17, 0x35
        ldi     r18, 0x0C
delay:  subi    r16, 1
        sbci    r17, 0
        sbci    r18, 0
        brne    delay
        ret
```

Якщо кнопка натиснута, тоді світлодіод ніколи не засвітиться, бо сторожовий таймер буде здійснювати скид МК ще до завершення підпрограми замтримки 0,5 сек, оскільки він запрограмований на період спрацювання 0,26 сек.