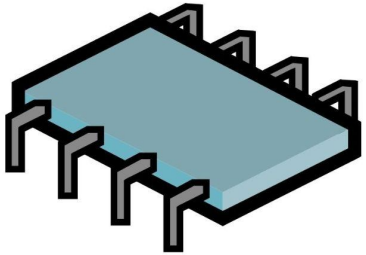
# Maskowanie

## JĘZYK C
## dla mikrokontrolerów



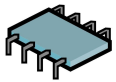
msalamon.pl

# Maskowanie bitów

Maskowanie ma wiele wspólnego z

wyborem i manipulacją bitami





# Maskowanie bitów

W zasadzie to, co omawialiśmy w poprzedniej lekcji

było maskowaniem

# Maska bitowa

# Maska bitowa

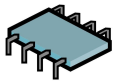
---

Słowo o długości odpowiadającej maskowanemu słowu

służące do operacji na wybranych bitach

Wikipedia





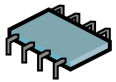
# Maska bitowa

To, co tworzyliśmy poprzez “wybór” bitów to była właśnie maska bitowa!

# Maska bitowa

Maska wraz z odpowiednią operacją bitową pozwalały na ustawianie, kasowanie, lub negację maskowanych bitów





# Maskowanie niepotrzebnych bitów

# Maskowanie niepotrzebnych bitów

Gdy odczytujemy jakiś rejestr to odczytujemy całość

# Maskowanie niepotrzebnych bitów

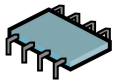
---

Często zależy nam na wartości 1, 2, 3 bitów

# Maskowanie niepotrzebnych bitów

Jak “pozbyć się” reszty?

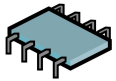




# Maskowanie niepotrzebnych bitów

Zamaskować je!





# Maskowanie niepotrzebnych bitów

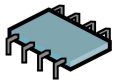
```
uint8_t Value = 0x5B;                       // 0b01011011

Value &= ~(BIT0 | BIT1 | BIT2 | BIT6 | BIT7);

                                  // ((1<<0) | (1<<1) | (1<<2) | (1<<7) | (1<<6))

Value &= ~(0b11000111);                     // 0xC7

Value &= (0b00111000);                      // 0x38

Value == 0x18;                              // 0b00011000
```

# Maskowanie niepotrzebnych bitów

A co gdyby wskazać (podać maskę) od razu tych bitów,

które nas interesują?





# Maskowanie niepotrzebnych bitów

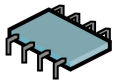
```
uint8_t Value = 0x5B;                    // 0b01011011

Value &= ~(BIT0 | BIT1 | BIT2 | BIT6 | BIT7);

                                         // (~(1<<0) | (1<<1) | (1<<2) | (1<<7) | (1<<6))

Value &= ~(0b11000111);                  // 0xC7

Value &= (0b00111000);                   // 0x38

Value == 0x18;                           // 0b00011000
```

# Maskowanie niepotrzebnych bitów

---

```
uint8_t Value = 0x5B;          // 0b01011011

Value &= (0b00111000);         // 0x38

Value == 0x18;                 // 0b00011000
```





# Maskowanie niepotrzebnych bitów

---

```
uint8_t Value = 0x5B;          // 0b01011011

Value &= 0x38;                 // 0b00111000
            (BIT3 | BIT4 | BIT5)

Value == 0x18;                 // 0b00011000
```

# Wyciągnięcie wartości

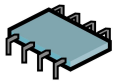
...z zamaskowanych bitów

# Wyciągnięcie wartości z kilku bitów

---

Często takie kilka bitów w środku potrzebujemy, aby miały wartość 0-X





# Wyciągnięcie wartości z kilku bitów

Table 22: Register 0xF4 *"ctrl_meas"*

| Register 0xF4 *"ctrl_meas"* | Name | Description |
|---|---|---|
| Bit 7, 6, 5 | osrs_t[2:0] | Controls oversampling of temperature data. See Table 24 for settings and chapter 3.4.3 for details. |
| Bit 4, 3, 2 | osrs_p[2:0] | Controls oversampling of pressure data. See Table 23 for settings and chapter 3.4.2 for details. |
| Bit 1, 0 | mode[1:0] | Controls the sensor mode of the device. See Table 25 for settings and chapter 3.3 for details. |

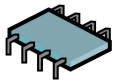
Table 23: register settings *osrs_p*

| osrs_p[2:0] | Pressure oversampling |
|---|---|
| 000 | Skipped (output set to 0x80000) |
| 001 | oversampling ×1 |
| 010 | oversampling ×2 |
| 011 | oversampling ×4 |
| 100 | oversampling ×8 |
| 101, others | oversampling ×16 |

# Wyciągnięcie wartości z kilku bitów

Trzeba ją przesunąć o odpowiednią ilość miejsc





# Wyciągnięcie wartości z kilku bitów

---

```
uint8_t ctrl_meas = 0x4B;   // 0b01001011

ctrl_meas &= 0x1C;          // 0b00011100

ctrl_meas == 0x08;          // 0b00001000

uint8_t osrs_p = crtl_meas >> 2; // 0b00000010
```

Table 22: R

| Register 0xF4 *"ctrl_meas"* | Name |
|---|---|
| Bit 7, 6, 5 | osrs_t[2:0] |
| Bit 4, 3, 2 | osrs_p[2:0] |
| Bit 1, 0 | mode[1:0] |

Table 23:

| osrs_p[2:0] | |
|---|---|
| 000 | |
| 001 | |
| 010 | |
| 011 | |
| 100 | |
| 101, others | |

# Wyciągnięcie wartości z kilku bitów

Table 22: Register 0xF4 *"ctrl_meas"*

| Register 0xF4 *"ctrl_meas"* | Name | Description |
|---|---|---|
| Bit 7, 6, 5 | osrs_t[2:0] | Controls oversampling of temperature data. See Table 24 for settings and chapter 3.4.3 for details. |
| Bit 4, 3, 2 | osrs_p[2:0] | Controls oversampling of pressure data. See Table 23 for settings and chapter 3.4.2 for details. |
| Bit 1, 0 | mode[1:0] | Controls the sensor mode of the device. See Table 25 for settings and chapter 3.3 for details. |

Table 23: register settings *osrs_p*

| osrs_p[2:0] | Pressure oversampling |
|---|---|
| 000 | Skipped (output set to 0x80000) |
| 001 | oversampling ×1 |
| 010 | oversampling ×2 |
| 011 | oversampling ×4 |
| 100 | oversampling ×8 |
| 101, others | oversampling ×16 |

# W drugą stronę

# Ustawienie wartości kilku bitów “w środku”

---

Ustawiać bity w “środku” zmiennej/rejestru też

czasem będziemy potrzebowali robić w zakresie 0-X

# Ustawienie wartości kilku bitów “w środku”

_ _ _

Table 22: Register 0xF4 *“ctrl_meas”*

| Register 0xF4 *“ctrl_meas”* | Name | Description |
|---|---|---|
| Bit 7, 6, 5 | osrs_t[2:0] | Controls oversampling of temperature data. See Table 24 for settings and chapter 3.4.3 for details. |
| Bit 4, 3, 2 | osrs_p[2:0] | Controls oversampling of pressure data. See Table 23 for settings and chapter 3.4.2 for details. |
| Bit 1, 0 | mode[1:0] | Controls the sensor mode of the device. See Table 25 for settings and chapter 3.3 for details. |

Table 23: register settings *osrs_p*

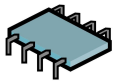
| osrs_p[2:0] | Pressure oversampling |
|---|---|
| 000 | Skipped (output set to 0x80000) |
| 001 | oversampling ×1 |
| 010 | oversampling ×2 |
| 011 | oversampling ×4 |
| 100 | oversampling ×8 |
| 101, others | oversampling ×16 |

# Ustawienie wartości kilku bitów “w środku”

Musimy tak traktować całość, aby nie zmienić

pozostałych bitów





# Ustawienie wartości kilku bitów “w środku”

```
___

uint8_t osrs_p = 2;                 // 0b00000010

uint8_t ctrl_meas = <odczyt>; // 0b???????? 

ctrl_meas &= 0xE3;                  // 0b???000??
                    ~(BIT2 | BIT3 | BIT4)

ctrl_meas |= (osrs_p<<2);           // 0b???010??
```

Table 22: R

| Register 0xF4 *“ctrl_meas”* | Name |
|---|---|
| Bit 7, 6, 5 | osrs_t[2:0] |
| Bit 4, 3, 2 | osrs_p[2:0] |
| Bit 1, 0 | mode[1:0] |

Table 23:

| osrs_p[2:0] | |
|---|---|
| 000 | |
| 001 | |
| 010 | |
| 011 | |
| 100 | |
| 101, others | |

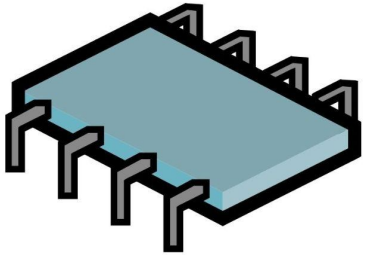
# Podsumowanie

---

- Maskowanie pozwala nam na traktowanie wybranych bitów w całej wartości
- Musimy tak działać, aby nie ruszyć pozostałych bitów
- Do wszystkiego wykorzystujemy maski i operacje bitowe

# Dzięki!

JĘZYK C
dla mikrokontrolerów



msalamon.pl