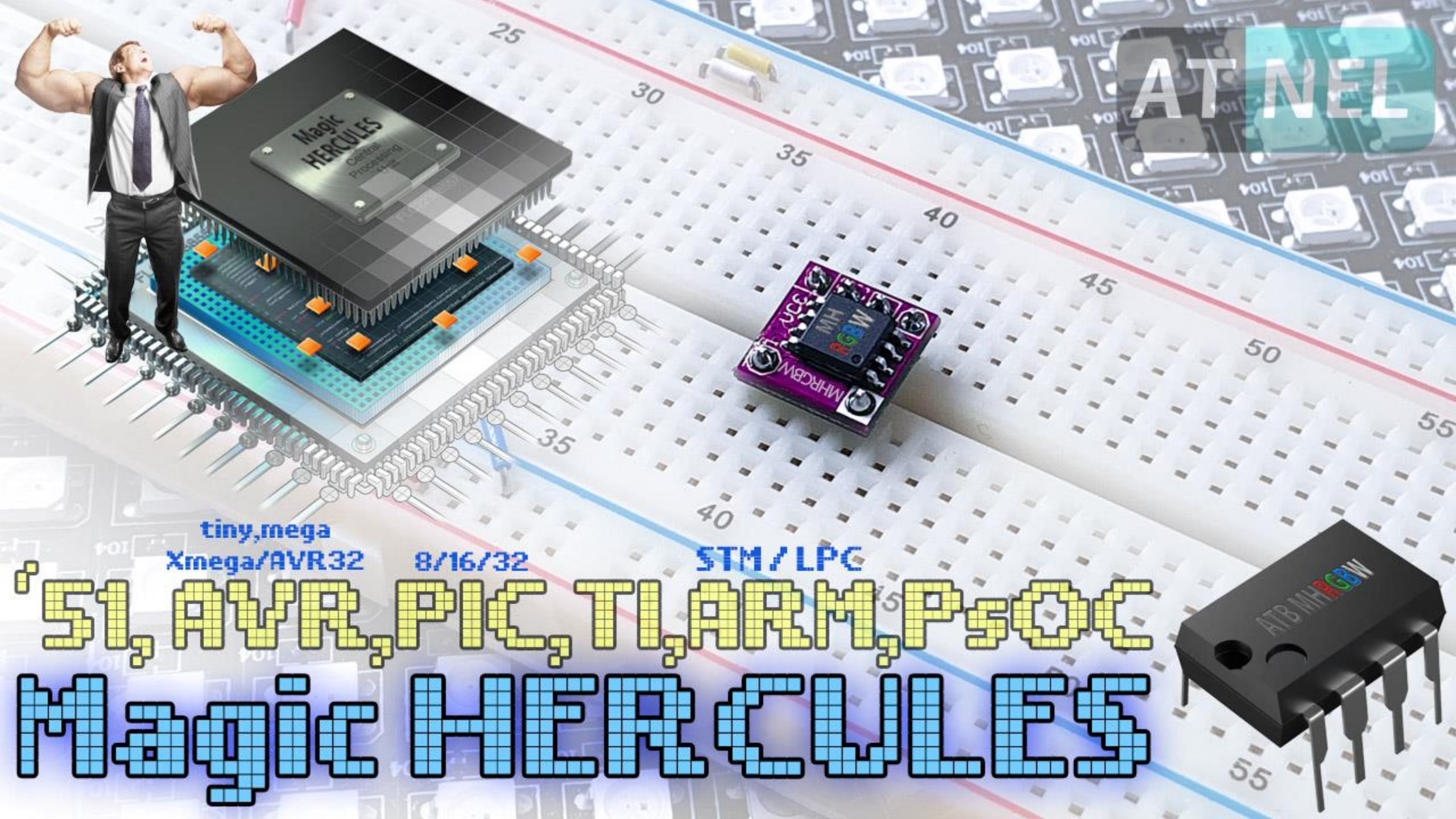

Magic
HERCULES
Central
Processing
Unit
ATNEL
tiny,mega
Xmega/AVR32
8/16/32
STM/LPC
'51,AVR,PIC,TI,ARM,PsOC
Magic HERCULES
ATBMHRGBW

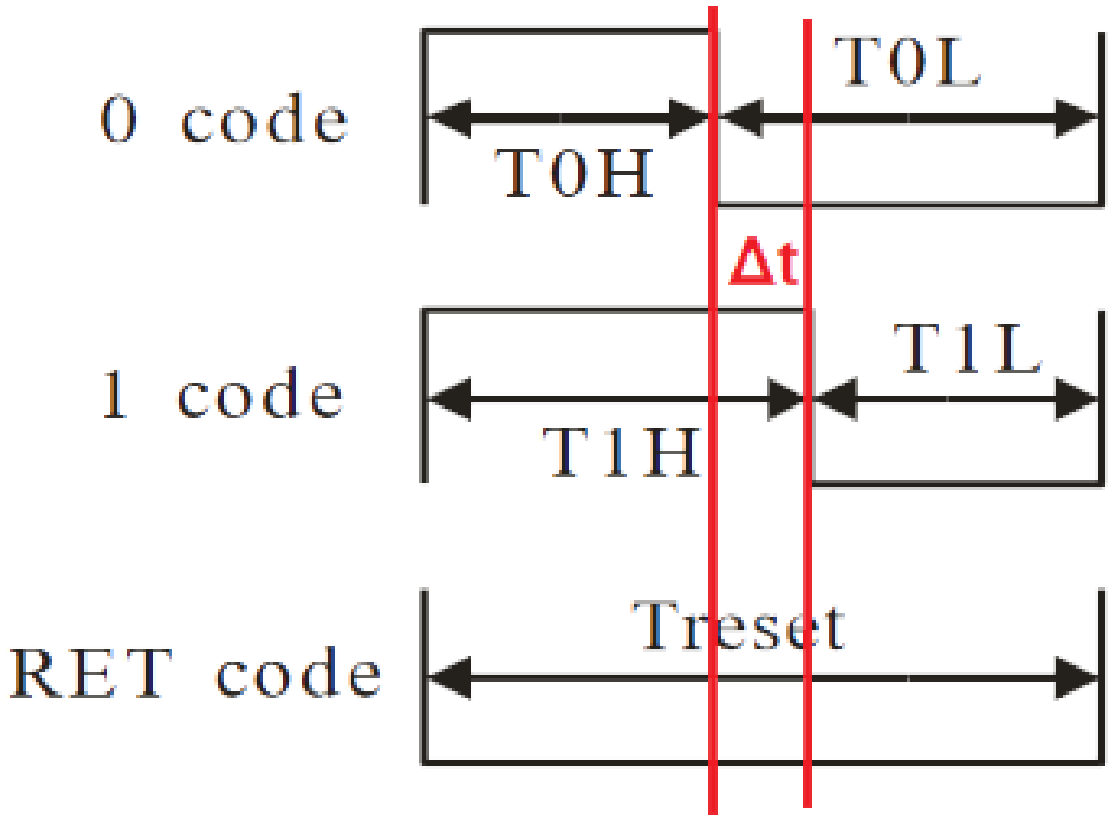
Zanim narodził się Magic Hercules…

- Na świecie był używany tylko protokół NZR do diod Magic LED ;)

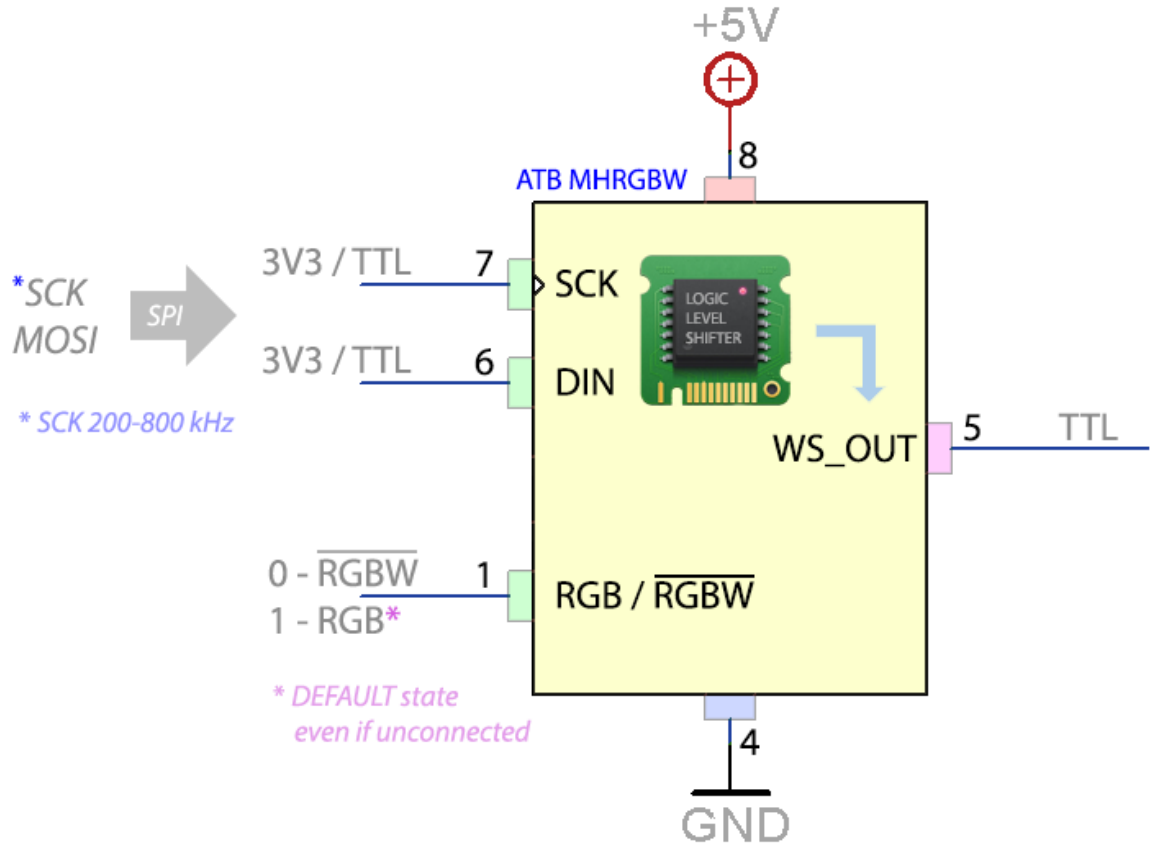
- Jest on bardzo trudny do oprogramowania na 8-bitowym µC *(ze względu na zależności czasowe i bardzo krókie impulsy)*
- Wymaga użycia wstawek kodu w czystym asemblerze.

- Napisanie optymalnego kodu nawet na 32-bitowe mikrokontrolery posiadające DMA, timery sprzętowe nie jest zbyt optymalne.

- Mikrokontrolery ARM i inne zasilane napięciem 3,3V wymagają translacji poziomów jeśli chodzi o sygnał wyjściowy do diod Magic LED.



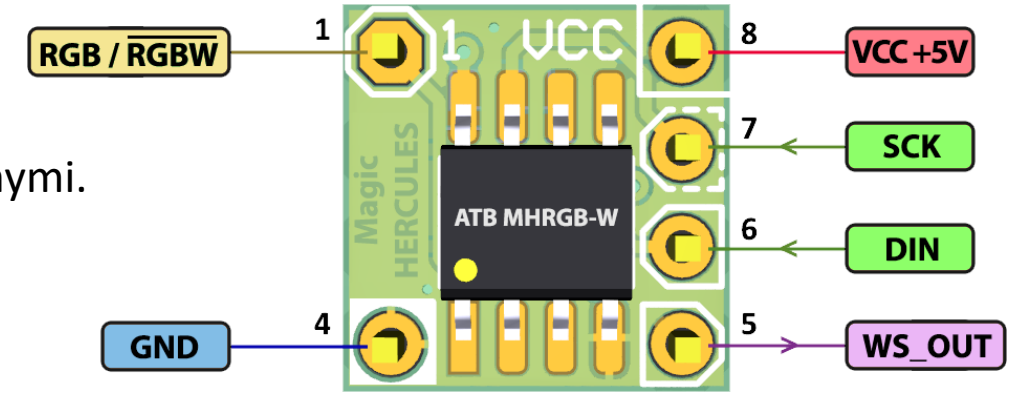
Miło nam zaprezentować nasz najnowszy moduł elektroniczny o nazwie **Magic Hercules**.
Czym jest ten moduł?
Gdyby trzeba było określić to w kilku słowach, to:



Jest to miniaturowy moduł DIP który spełnia funkcje:

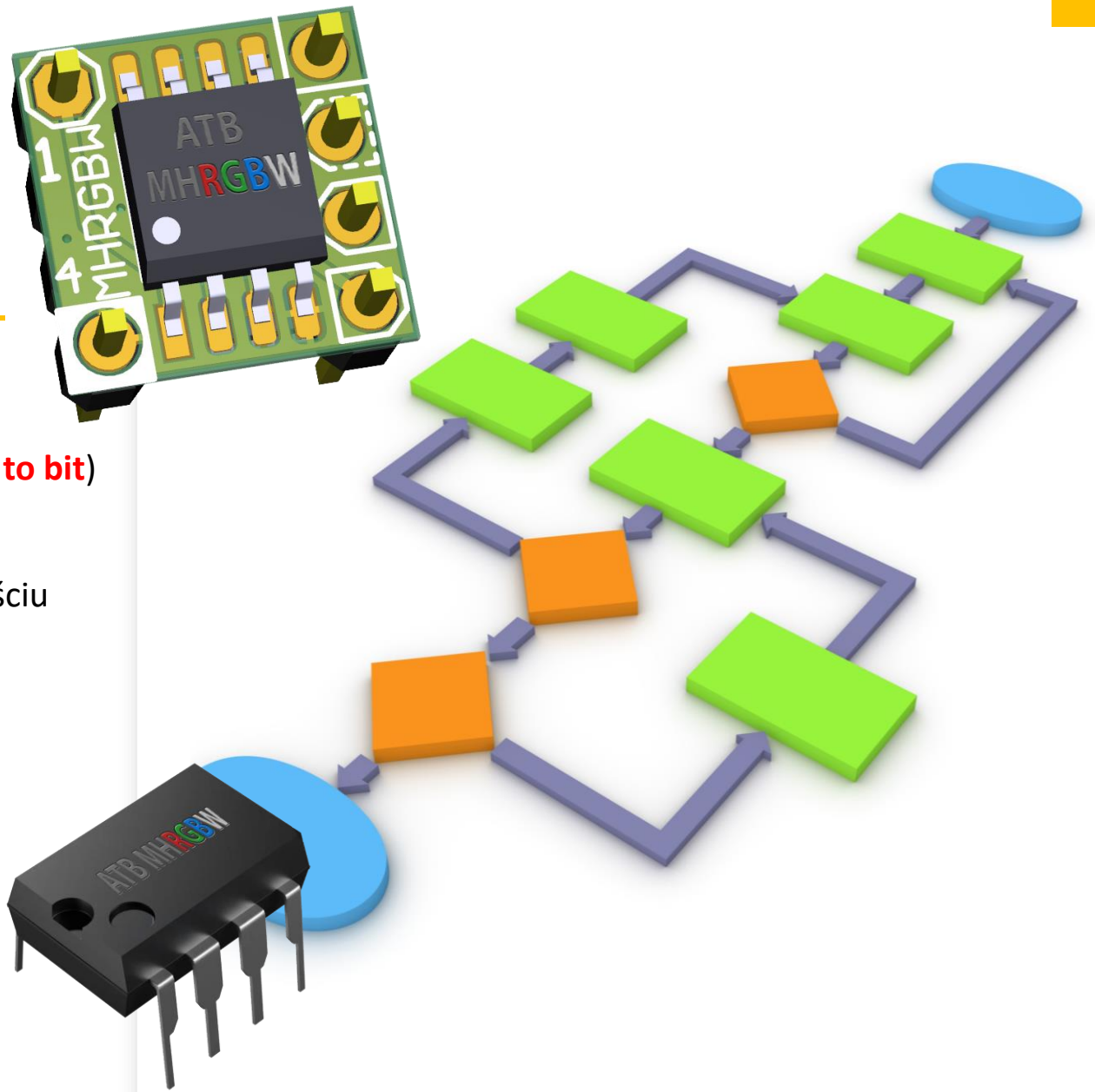
- **Konwertera** SPI do NZR (ponieważ NZR jest protokołem Magic LED).

  można zatem powiedzieć że jest to konwerter SPI do Magic LED
- **TESTER** Magic LED.
- **Konwerter poziomów**: SPI 3V3 do TTL



- Pracuje z: WS2811, WS2812(B), WS2813, WS2815, SK6812 i podobnymi.
- Pracuje z każdymi taśmami LED zasilanymi **+5V** lub **+12V**.
- Obsługuje różne sekwencje kolorów: RGB, GRB, BGR … a także **RGBW**.

Jak działa Magic Hercules?

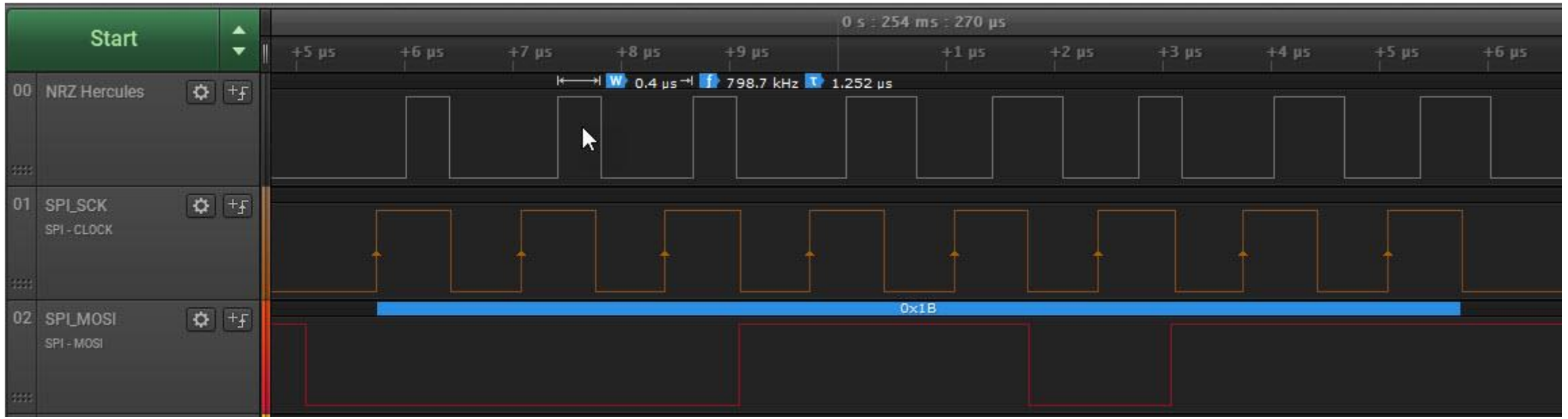


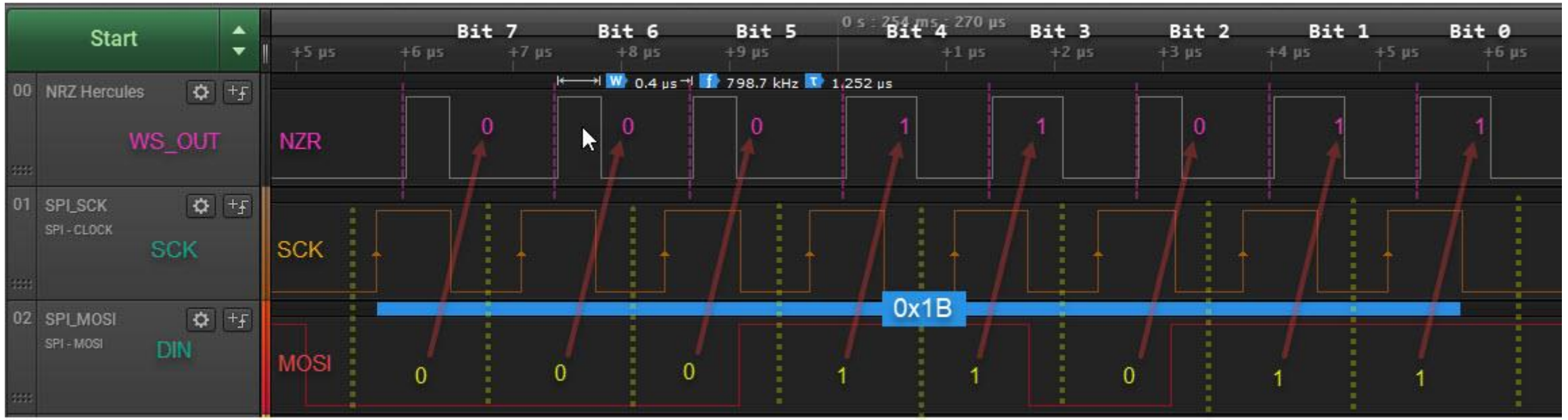
- Konwertuje SPI bezpośrednio na protoków NZR (**bit to bit**)
- SPI clock może być od 200 kHz aż do 800 kHz
- Konwertuje sygnały SPI 3V3 do poziomu TTL na wyjściu

- Zasilanie modułu to: +5V
- Poziomy sygnałów SPI: od **3V3** do **5V** (TTL)
- Wyjście NZR: TTL

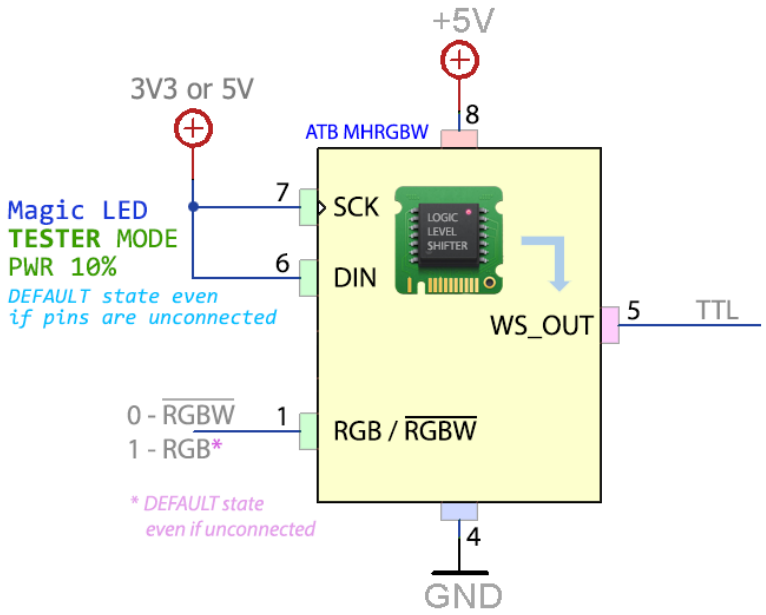
- Może pełnić rolę TESTERA diod Magic LED

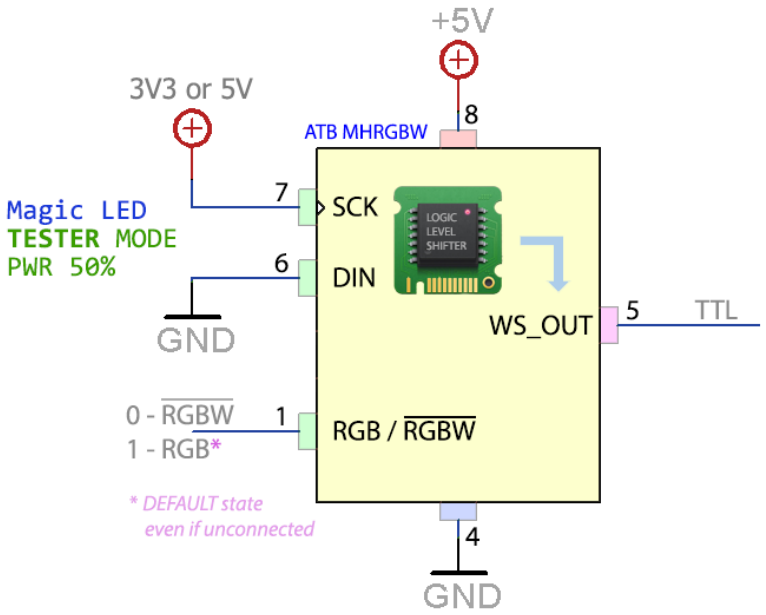
Widok konwersji – SPI do NZR bit to bit

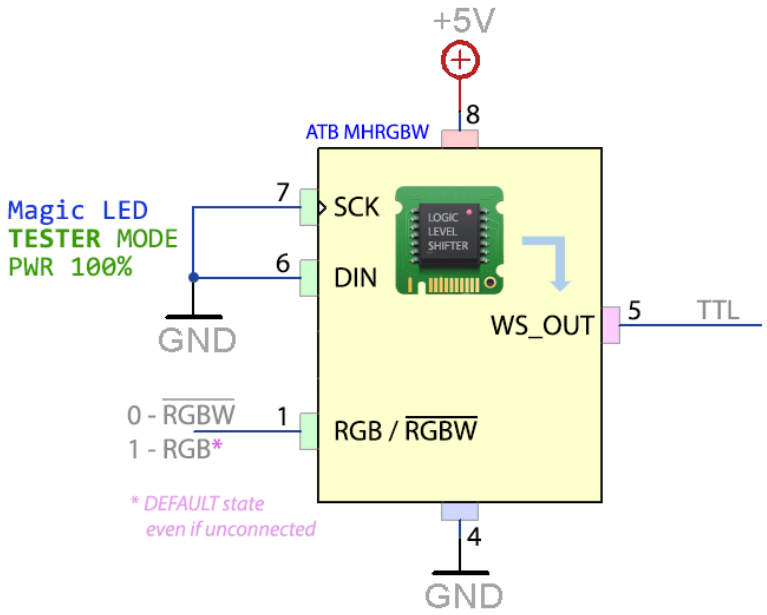




Magic Hercules jako Magic LED TESTER





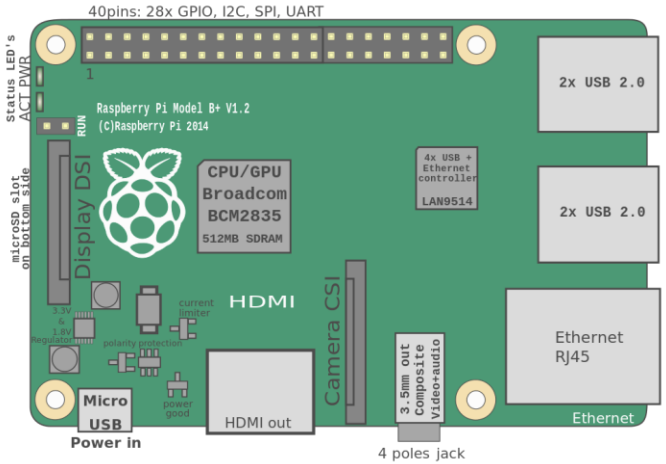


- Wysyła wzorce do taśm LED *(lub ekranów LED)* dla **1024** diod LED!
- Posiada TRZY tryby testowe: 10% , 50% and 100% *(różna jasność diod LED - w zależności od wydajności posiadanego zasilacza)*
- Potrafi testować zarówno diody **RGB** jak i **RGBW** – w zależności od stanu pinu 1 *(RGB/RGBW)*
- Pierwszy pojawiający się wzorzec testowy pozwala od razu rozpoznać sekwencję kolorów *(RGB, BRG, GRB, RGBW, itp)*
- Kolejne wyświetlane testy pozwalają określić prawidłowe działania wszystkich diod na taśmie lub ekranie *(każdy kolor z osobna i kombinacje łączenia kolorów)*

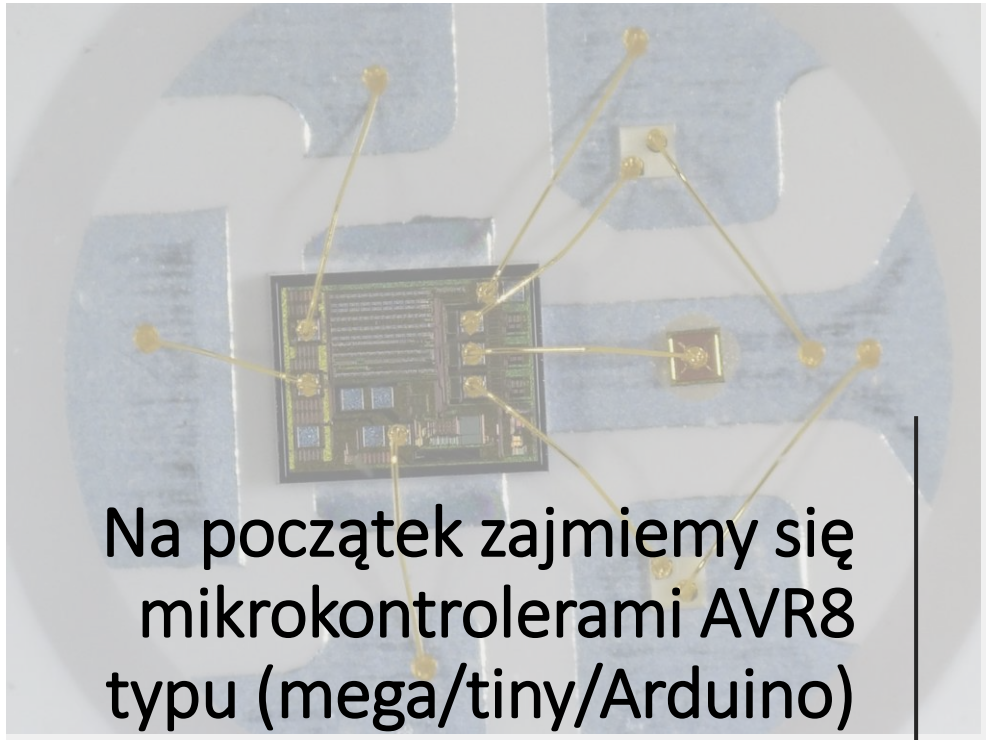
Z jakimi mikrokontrolerami i urządzeniami może pracować Magic Hercules jako konwerter?

Jest użyteczny dla:

- Mikrokontrolerów 8-bitowych jak np **AVR** (ATmega, ATtiny, Xmega, Arduino) oraz serii 80’51
- Wszystkich z rodziny **PIC** : 8/16/32 bit
- Wszystkich z rodziny **ARM** cortex: STM, NXP LPC, Cypress PSoC, AVR32
- Z każdym modułem **Raspberry PI** który posiada sprzętowe wyjście SPI



- Jest szczególnie przydatny gdy µC posiada **DMA**



Na początek zajmiemy się mikrokontrolerami AVR8 typu (mega/tiny/Arduino) bez DMA

Można używać taktowania z zewn. kwarca lub **wewnętrznego oscylatora!**

Taktowanie może być: od 1 MHz do 20 MHz *(24 MHz jeśli chcemy nieco przetaktować µC)*

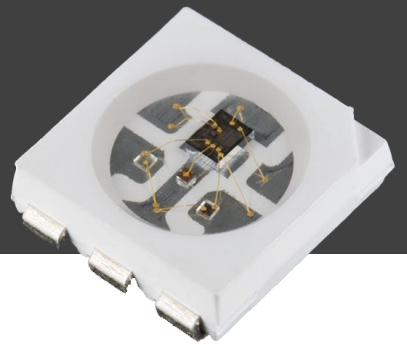
Używamy najprostszego protokołu SPI
*(tylko **dwie linie** SCK i MOSI)*

Możemy używać niektórych przerwań podczas transmisji *(jest to troszkę trudne do wyjaśnienia na tym etapie)*

Możemy korzystać zarówno ze sprzętowego jak i programowej wersji **SPI** !!!

Nie potrzebujemy żadnych wstawek asemblera!
Używamy tylko czystego języka C !

Magic HERCULES (mega/tiny/Arduino) ze sprzętowym SPI



Hardware SPI

F_CPU clock from ext. Quartz or even int. Oscilator

F_CPU Hz	Hardware **SPI** prescaler	**Magic LED** **max** FREQ Hz	**Magic LED type** *due to the reset time*	Hardware **SPI** prescaler	**Magic LED** **min** FREQ Hz
1 000 000	2	500 000	only SK6812	4	250 000
2 000 000	4	500 000	only SK6812	8	250 000
4 000 000	8	500 000	WS281x and SK6812	16	250 000
8 000 000	16	500 000	WS281x and SK6812	32	250 000
11 059 200	16	691 200	WS281x and SK6812	32	345 600
12 000 000	16	**750 000**	WS281x and SK6812	32	375 000
16 000 000	32	500 000	WS281x and SK6812	64	250 000
18 432 000	32	576 000	WS281x and SK6812	64	288 000
20 000 000	32	625 000	WS281x and SK6812	64	312 500
24 000 000	32	**750 000**	WS281x and SK6812	64	375 000

WS281x *= WS2811, WS2812(B)*

- Szeroki zakres częstotliwośći F_CPU.

Można używać także nietypowych jak:

6.4MHz (ATtiny85) z preskalerem SPI = 8 -> NZR 800 kHz

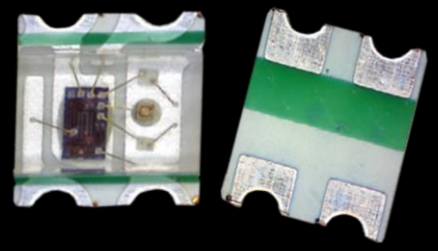
9.6MHz (ATtiny13) z preskalerem SPI = 16 -> NZR 600 kHz

- Szeroki zakres częstotliwości NZR.

- W tabeli przykłady zależności pomiędzy częstotliwością NZR a taktowaniem procesora

Magic HERCULES
(mega/tiny/Arduino)
z programowym SPI

- Mniejszy zakres taktowania µC ale za to możliwość osiągania większych częstotliwości NZR.
- Wersja programowa pozwala na zupełną dowolność w doborze pinów.



Software SPI

F_CPU clock from ext. Quartz or even int. Oscilator

F_CPU Hz	Magic LED max FREQ ~ Hz	Magic LED type *due to the reset time*
4 000 000	203 000	only SK6812
8 000 000	402 000	WS281x and SK6812
11 059 200	553 000	WS281x and SK6812
12 000 000	600 000	WS281x and SK6812
16 000 000	725 000	WS281x and SK6812
18 432 000	770 000	WS281x and SK6812
20 000 000	770 000	WS281x and SK6812
24 000 000	775 000	WS281x and SK6812
		WS281x = WS2811, WS2812(B)

Źródło w C *(AVR8 bez DMA)* – nie można prościej !

Sprzętowe SPI

```
char magic_buffer[768];                 // RAM buffer for 256 RGB Magic LEDs

static void spi_send_byte( uint8_t data ) {
        SPDR = data;                            // send byte to SPDR register
        while( !(SPSR & (1<<SPIF)) );           // wait for data shifting
}

void spi_send_buf( void * buf_out, int size ) {

        for( int i = 0; i < size; i++ )
                spi_send_byte( *buf_out++) );

}
```


Źródło w C *(AVR8 bez DMA)* – nie można prościej !

Sprzętowe SPI

```
char magic_buffer[768];                 // RAM buffer for 256 RGB Magic LEDs

static void spi_send_byte( uint8_t data ) {
        SPDR = data;                            // send byte to SPDR register
        while( !(SPSR & (1<<SPIF)) );           // wait for data shifting
}

void spi_send_buf( void * buf_out, int size ) {
 ----> cli();    // disable interrupts
        for( int i = 0; i < size; i++ )
                spi_send_byte( *buf_out++) );
 ----> sei();    // enable interrupts
}
```

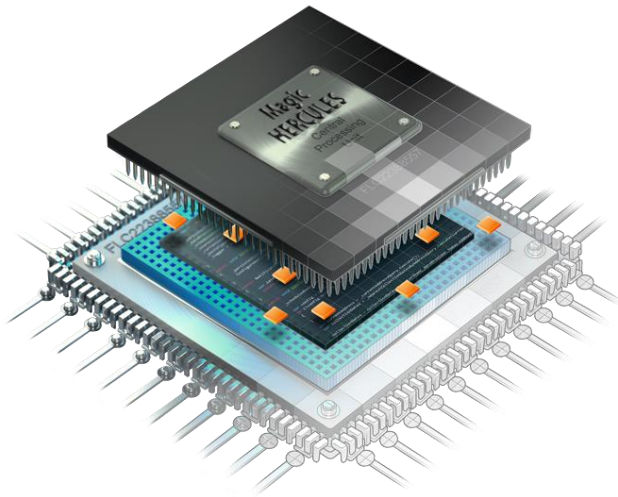

Źródło w C *(AVR8 bez DMA)* – nie można prościej !

Programowe SPI

```
char magic_buffer[768];                    // RAM buffer for 256 RGB Magic LEDs

static void spi_send_byte( uint8_t byte ) {
        uint8_t cnt;
        cnt = 0x80;
        while( cnt ) {
                if( byte & cnt ) MOSI_HI; else MOSI_LO;
                SCK_HI;
                cnt >>= 1;
                asm( "nop" );     // for F_CPU >= 16 MHz
                SCK_LO;
                asm( "nop" );     // for F_CPU >= 16 MHz
        }
        MOSI_HI;
}
```


Podsumowanie – AVR bez DMA



- Kod napisze nawet całkowicie początkujący!
- Nie potrzeba wstawek asemblerowych
- Bardzo szeroki zakres taktowania procesora
- Można używać zarówno zewn. kwarca jak i wewn. oscylatora
- Można używać nietypowych częstotlowości NZR np: 558 kHz, 273 kHz, itp
- Można korzystać z przerwań w czasie transmisji *(będzie wyjaśnione później)*
- Można korzystać ze sprzętowej albo programowej implementacji SPI
- Tylko dwu przewodowa komunikacja pomięcy CPI i Magic Herculesem
- Możemy zasilać procesory napięciem 3,3V i zawsze będziemy mieli prawidłową translację sygnałów!
- Perfekcyjne rozwiązanie dla edukacji. *(dla nauczycieli zajmujących się nauką elektronik i programowania do prowadzenia zajęć)*

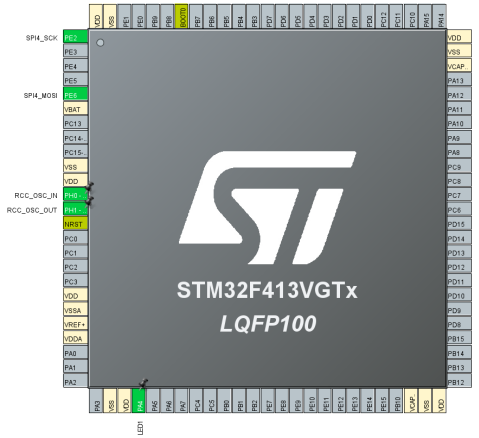
Jeśli posiadasz µC z **DMA** to teraz zaczyna się najlepsze!

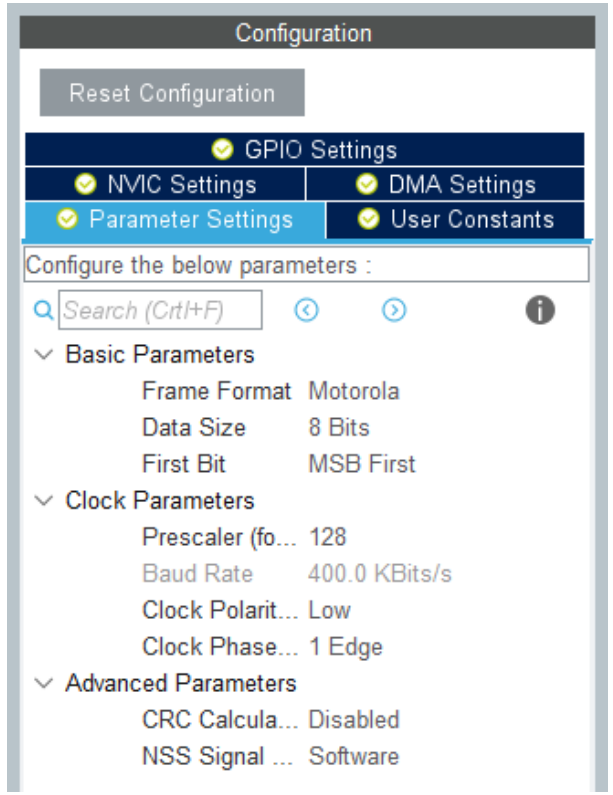
Xmega, PIC 18/24/32, ARM cortex based microcontroller (STM, NXP, ESP82xx)

Ilość wad = **0** (słownie: **ZERO**)

Tylko zalety takiego rozwiązania!

Jak zacząć? na STM32





- Możemy swobodnie użyć SPI z DMA

- Po skonfigurowaniu wybranego kanału SPIx z DMA
- Po każdej zmianie w Magic RAMbuffer, uruchamiamy transfer SPI-DMA
- Możemy użyć timera sprzętowego do wysyłania co 20 ms
- Możemy użyć wielu linii WS i obsłużyć nawet duże bufory dla wideo!

STM32 prosty przykład HAL SPI+DMA

dużo proście i dużo szybciej

```
char magic_buffer[4096];      // RAM buffer for 1024 RGBW Magic LEDs

/* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_SPI4_Init();

 /* somewhere in code */
  memcpy( magic_buf, fx_buf, 4096 );
  HAL_SPI_Transmit_DMA( &hspi4, magic_buf, 4096 );
```