Warsztaty AVR

Czyli jak zacząć przygodę z embedded

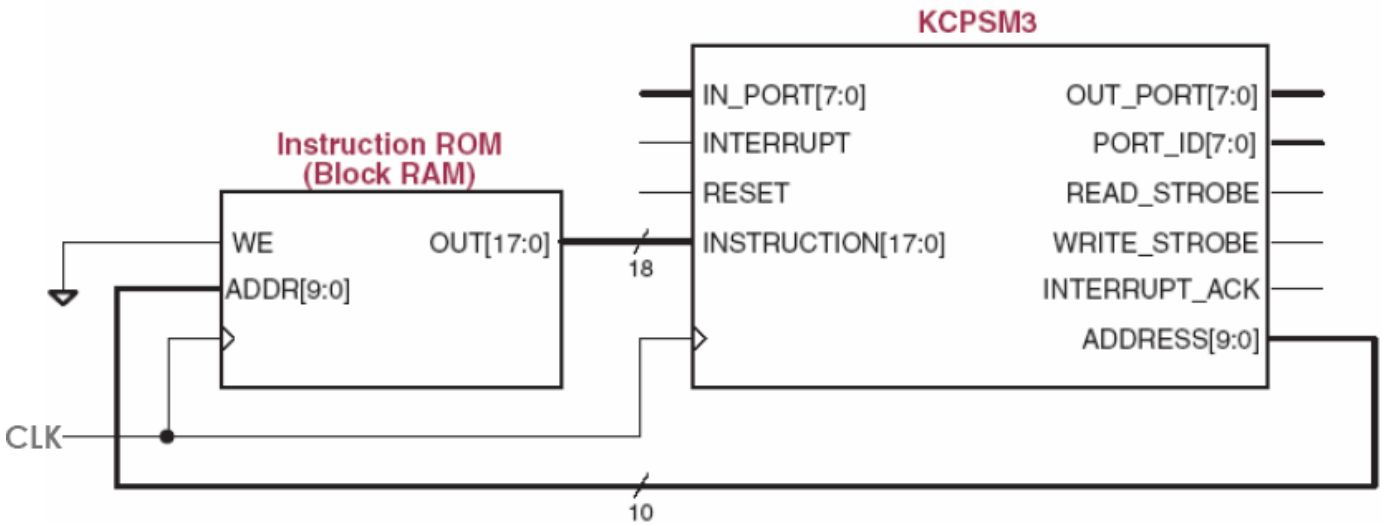
Łukasz Hawryłko

SKN CHIP

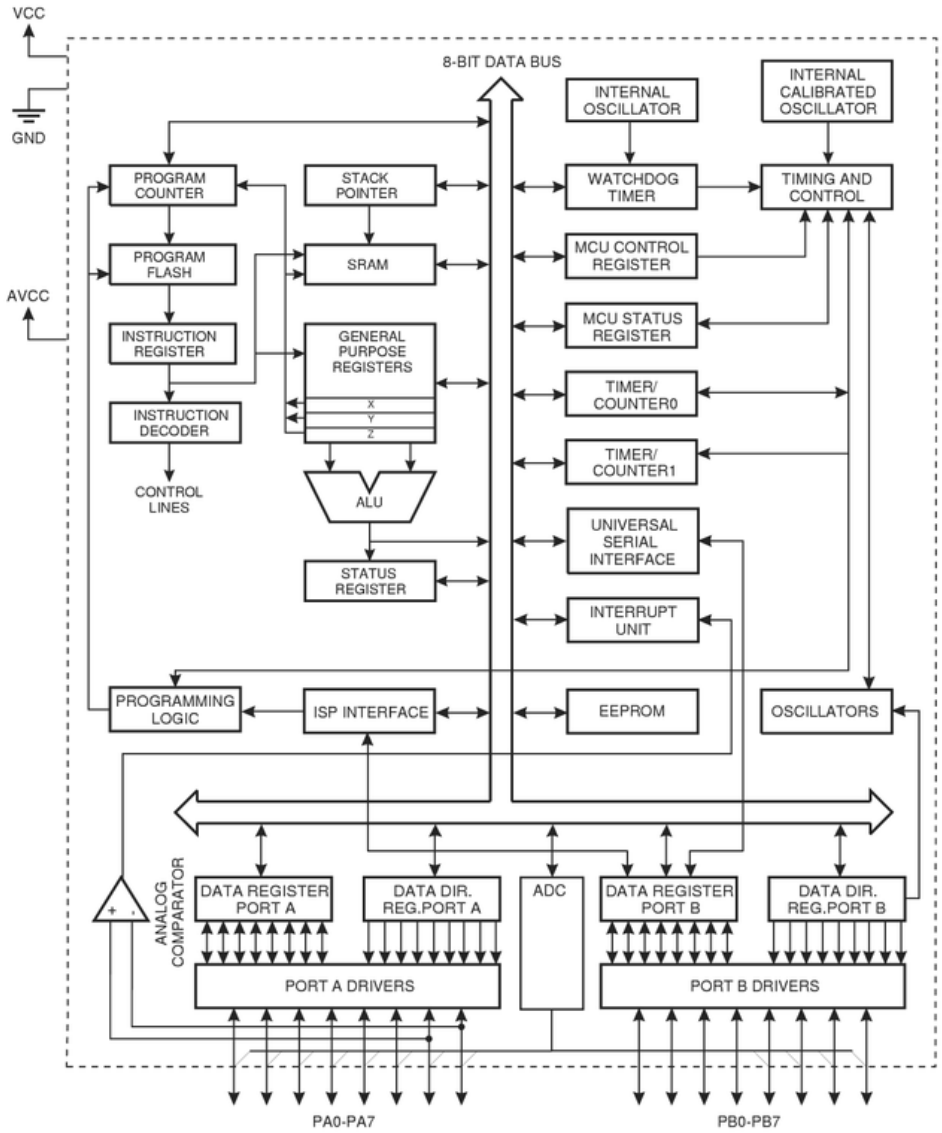
28 stycznia 2013

Budowa mikrokontrolera

Ogólna architektura mikroprocesora



Schemat blokowy uC z rodziny AVR



- Zmodyfikowana architektura harwardzka
- 8-bitowa szyna danych
- 16-bitowe instrukcje
- 8-bitowe słowa w pamięci SRAM

Pamięć uC ATmega32

- Pamięć programu typu (Flash) - 32kB
- Pamięć danych (SRAM) - 2kB
- Pamięć non volatile (EEPROM) - 1kB

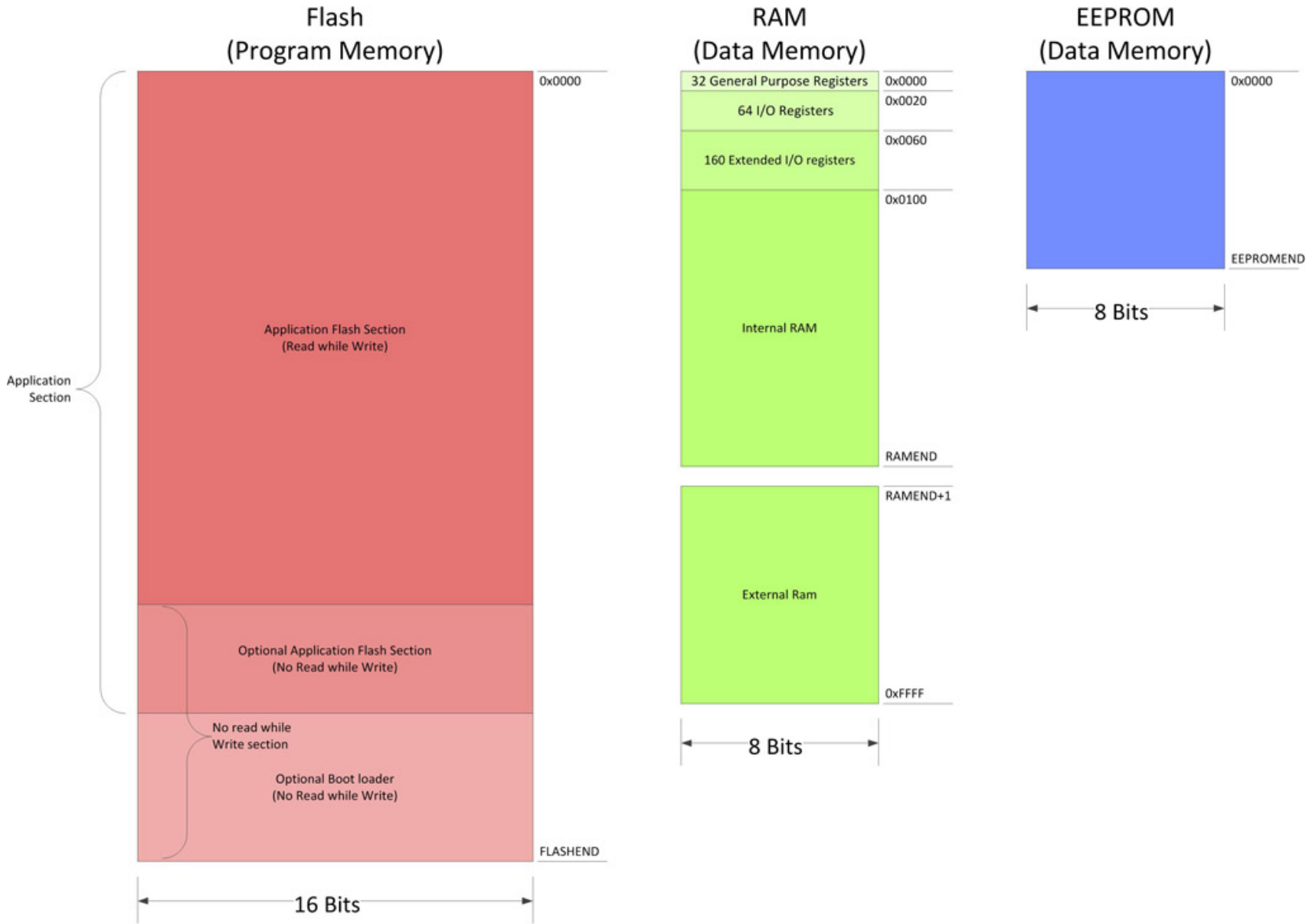
Pytanie

Po co dodatkowy blok pamięci EEPROM skoro Flash też jest non volatile?

Uwaga!

Przekroczenie zakresu pamięci programu jest zazwyczaj sygnalizowane przez kompilator, natomiast w przypadku pamięci danych niekoniecznie.

Mapa pamięci uC z rodziny AVR



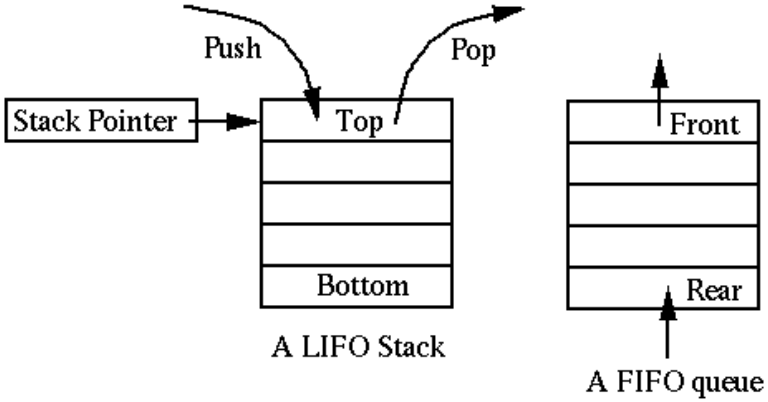
Rejestry

Rdzeń AVR zawiera trzy podstawowe typy rejestrów:

- 32 8-bitowe rejestry ogólnego przeznaczenia
- 16-bitowy wskaźnik stosu (stack pointer)
- 22-bitowy licznik programu (program counter)

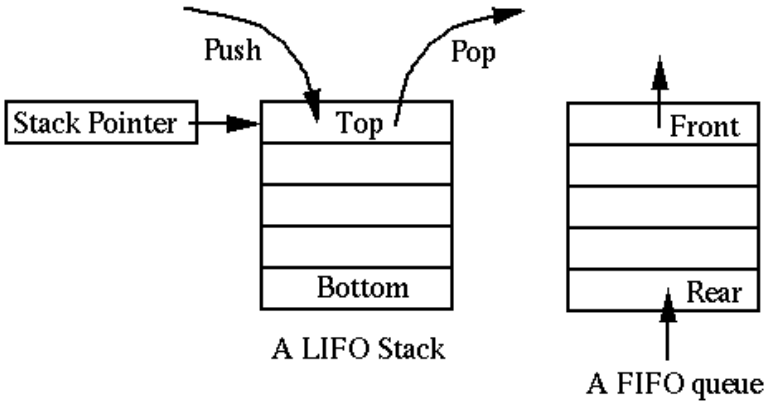
Pisząc w C szanse na potrzebę bezpośredniego skorzystania z rejestrów są niewielkie, ale przy nietypowych programach może zajść potrzeba np. modyfikowania wskaźnika stosu.

Stos i kolejka



- Stos (stack) jest wykorzystywany przy wywoływaniu podprogramów (funkcji)
- Zapisuje się w nim m.in. wartość licznika programu przed wywołaniem funkcji
- Jest tworzony oraz obsługiwany przez kompilator
- Kolejka (queue) to struktura wykorzystywana do budowy buforów
- Programista tworzy ją sam i sam zajmuje się obsługą

Stos i kolejka



- Stos (stack) jest wykorzystywany przy wywoływaniu podprogramów (funkcji)
- Zapisuje się w nim m.in. wartość licznika programu przed wywołaniem funkcji
- Jest tworzony oraz obsługiwany przez kompilator
- Kolejka (queue) to struktura wykorzystywana do budowy buforów
- Programista tworzy ją sam i sam zajmuje się obsługą

Stos i kolejka

Uwaga!

Nic nie stoi na przeszkodzie, żeby zaimplementować własny stos w programie. Aczkolwiek przyjęło się, że mówiąc „stos” ma się na myśli miejsce w pamięci wykorzystywane przez kompilator przy wywoływaniu funkcji.

Zegar

Mikrokontroler jak każdy układ synchroniczny potrzebuje zegara od którego zależy szybkość jego działania oraz pobór mocy. Najczęściej wykorzystywane źródła zegara to:

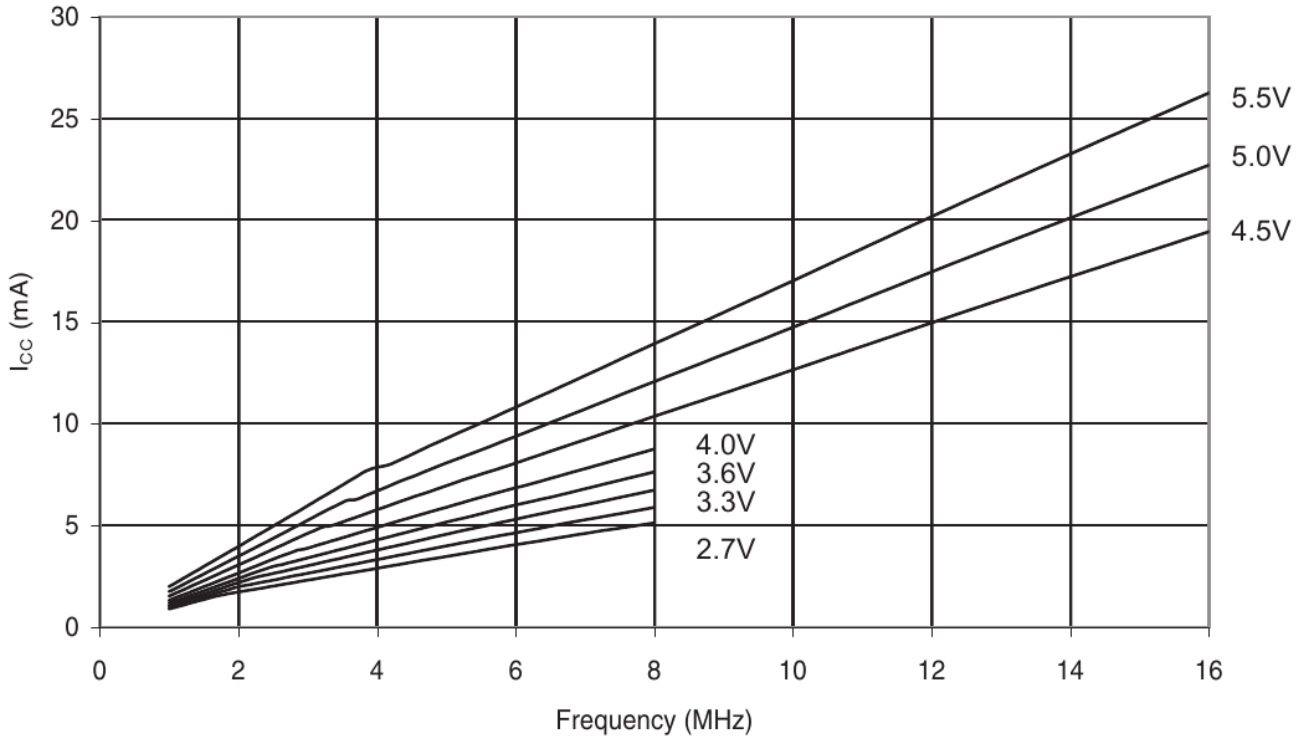
- Wewnętrzny oscylator RC
- Zewnętrzny rezonator kwarcowy
- Zewnętrzny generator zegara

ATmega32 może pracować z maksymalnym zegarem 16MHz, w nowszych mikrokontrolerach w celu uzyskania wysokich częstotliwości stosuje się układ PLL.

Pytanie

Po co inne źródła zegara, skoro można uruchomić uC na wbudowanych oscylatorze RC i wszystko działa?

Zegar a pobór mocy



Przy projektowaniu układów zasilanych z baterii warto zastanowić się jaka moc obliczeniowa jest nam tak na prawdę potrzeba i odpowiednio dobrać zegar.

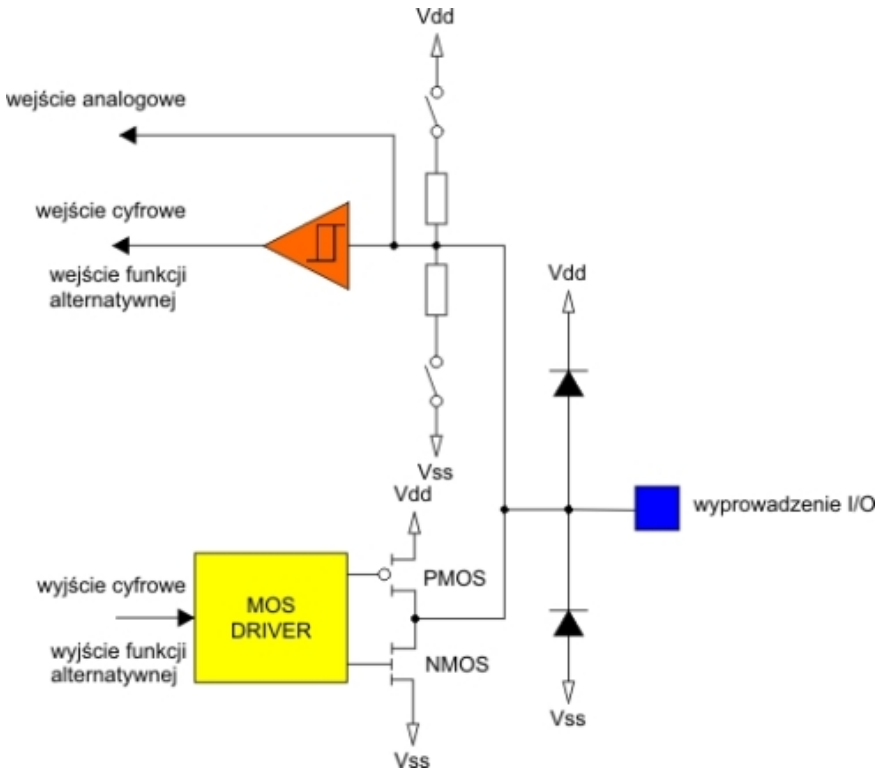
Inne sposoby ograniczenia poboru mocy

Praktycznie każdy mikrokontroler ma możliwość wejścia w tryb niskiego poboru mocy (sleep mode). W ATmega32 mamy do wyboru aż 6 różnych trybów. Uśpiony uC zatrzymuje wykonywanie kodu tuż za komendą uśpienia i czeka na wybudzenie. To, w jaki sposób można go wybudzić zależy od tego w jakim trybie go uśpiono, szczegóły można znaleźć w nocie katalogowej układu.

Porada

Wejście w tryb uśpienia może zmniejszyć pobór prądu nawet 1000x, przy budowie np. czujnika temperatury wysyłającego co sekundę dane do stacji bazowej usypianie uC jest czynnością obowiązkową.

GPIO

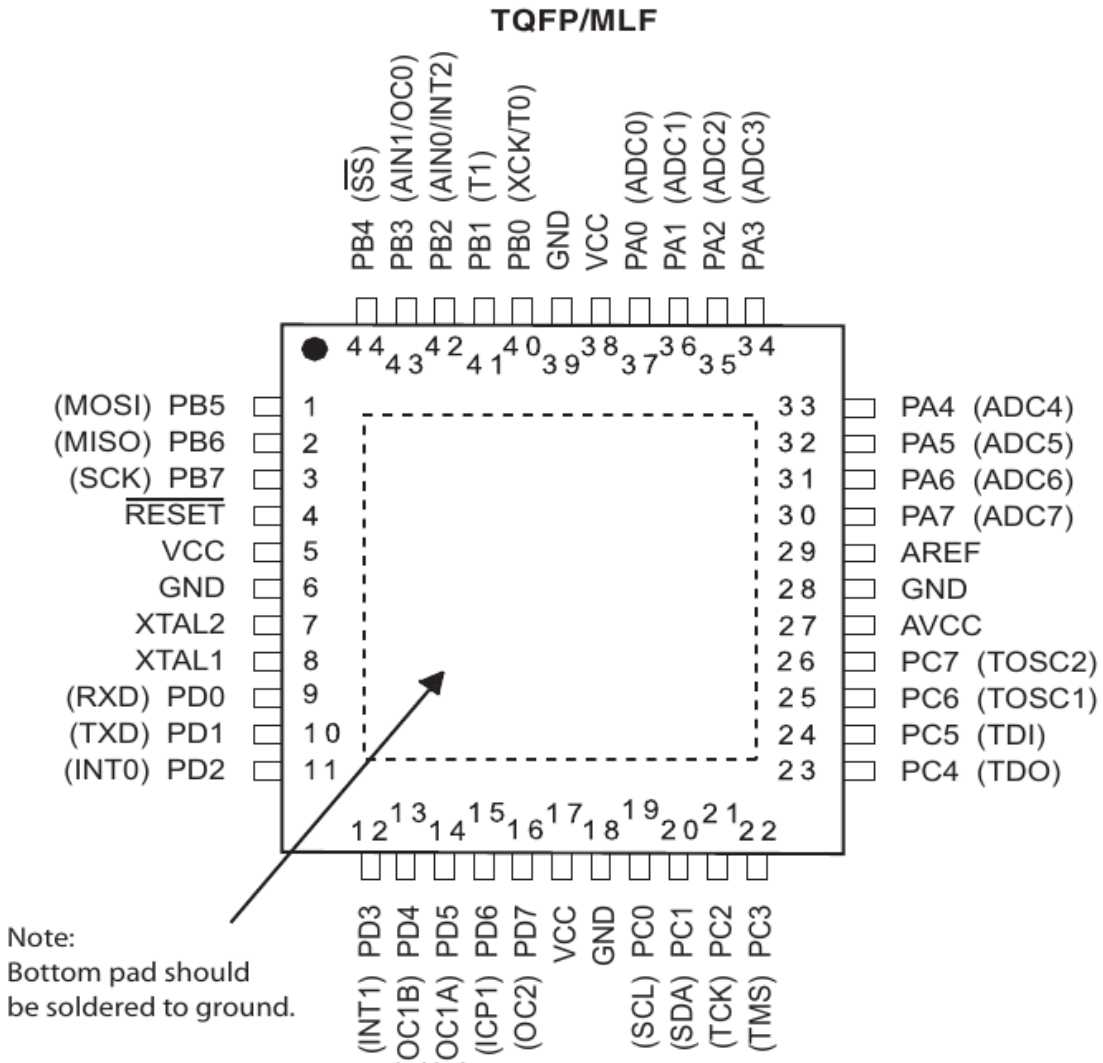


- Mogą pracować jako wyjście lub jako wejście
- W AVR jeżeli ustawimy jako wyjście mogą przyjąć dwa stany - 1 i 0
- Jeżeli ustawimy jako wejście zawsze mają stan Z, dodatkowo można włączyć rezystor pull-up
- W innych uC można ustawić np.: prąd na wyjściu, histerezę, tryb open-drain, czy włączyć rezystor pull-down

Pytanie

Jak są ustawione piny GPIO po resecie uC i dlaczego?

GPIO


TQFP/MLF
PB4 (SS)
PB3 (AIN1/OC0)
PB2 (AIN0/INT2)
PB1 (T1)
PB0 (XCK/T0)
GND
VCC
PA0 (ADC0)
PA1 (ADC1)
PA2 (ADC2)
PA3 (ADC3)
44 43 42 41 40 39 38 37 36 35 34
(MOSI) PB5 1
(MISO) PB6 2
(SCK) PB7 3
RESET 4
VCC 5
GND 6
XTAL2 7
XTAL1 8
(RXD) PD0 9
(TXD) PD1 10
(INT0) PD2 11
33 PA4 (ADC4)
32 PA5 (ADC5)
31 PA6 (ADC6)
30 PA7 (ADC7)
29 AREF
28 GND
27 AVCC
26 PC7 (TOSC2)
25 PC6 (TOSC1)
24 PC5 (TDI)
23 PC4 (TDO)
12 13 14 15 16 17 18 19 20 21 22
PD3 (INT1)
PD4 (OC1B)
PD5 (OC1A)
PD6 (ICP1)
PD7 (OC2)
VCC
GND
PC0 (SCL)
PC1 (SDA)
PC2 (TCK)
PC3 (TMS)
Note:
Bottom pad should
be soldered to ground.

Przerwania

Przerwanie jest takie zdarzenie, które natychmiastowo przerywa wykonywanie programu i wymusza wejście do funkcji je obsługującej. Rodzina AVR posiada system przerwań wektorowych, przez co nie ma potrzeby sprawdzania źródła przerwania, procesor sam wywołuję odpowiednią funkcję. Przerwania można podzielić na dwie grupy:

- Wewnętrzne, od peryferii uC
- Zewnętrzne, zgłaszane poprzez specjalnie piny uC

Więcej szczegółów będzie można znaleźć w części warsztatów poświęconych pisaniu programów w języku C.

Przetwornik A/D

Atmega32 posiada jeden przetwornik analogowo-cyfrowy z multipleksacją wejścia, tak że dostępne jest osiem kanałów. Ma on następujące parametry:

- Rozdzielczość - 10 bit
- Maksymalna szybkość próbkowania - 15kSPS
- Zakres przetwarzania od 0V do V_{REF}
- Wewnętrzne źródło napięcia referencyjnego 2.56V
- Osobne zasilanie i możliwość podania zewnętrznego napięcia referencyjnego

Warto wiedzieć

Większość uC ma wbudowany przetwornik typu SAR (z sukcesywną aproksymacją), jest on tani i szybki, ale ma ograniczoną rozdzielczość. Jeżeli zależy nam na dużej rozdzielczość to warto kupić zewnętrzny przetwornik $\Sigma - \Delta$.

Liczniki

Do dyspozycji mamy dwa liczniki 8-bitowe oraz jeden 16-bitowy, mogą być one wykorzystane do:

- Wyzwalania wykonywania jakiś cyklicznych czynności
- Generowania przebiegów, w tym przebiegów PWM
- Odliczania czasu
- Mierzenia ilości impulsów
- Mierzenia czasu trwania impulsu, wypełnienia sygnału

Układy interfejsowe

Aby skomunikować uC z innymi urządzeniami potrzeba skorzystać z jakiegoś ustandaryzowanego interfejsu. Można skorzystać z dwóch możliwości:

- Napisać emulację programową
- Skorzystać z interfejsów, które są wbudowane w uC

ATmega32 posiada kilka wbudowanych interfejsów, które umożliwiają komunikację zarówno z innymi uC, z zewnętrznymi układami peryferyjnymi, a także z komputerem, czy z telefonem. Są to:

- USART
- SPI
- I^2C, w procesorach Atmela występujący pod nazwą TWI

Pytanie

Czy poświęcić część warsztatów na omówienie tych interfejsów?

Miejsce miktrokontrolera w układzie

Zasilanie części cyfrowej

Pierwszą czynnością jaką należy wykonać przy projektowaniu układu z uC jest dobór napięcia zasilającego. W tym celu należy sprawdzić:

- Jakimi możliwymi źródłami napięć dysponujemy
- Zakres napięć pracy uC
- Zakres napięć pracy układów peryferyjnych

Najbardziej standardowymi napięciami zasilającymi są:

- 3.3V - obecnie najczęściej stosowane
- 5V - używane kiedyś, zostało wyparte przez 3.3V

W przypadku braku możliwości uzgodnienia jednego napięcia należy skorzystać z konwertera poziomów napięć.

Zasilanie części cyfrowej

Przy projektowaniu zasilania części cyfrowej musimy pamiętać o:

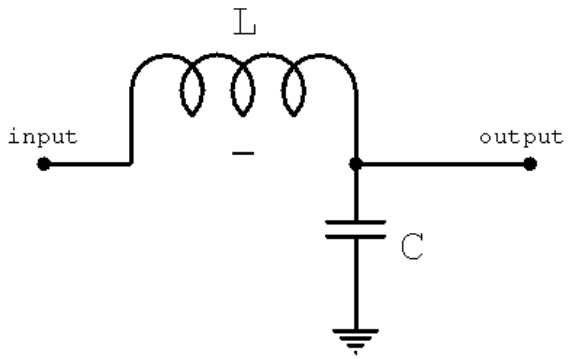
- Podłączeniu wszystkich pinów zasilających
- Umieszczeniu regulatora, wraz z kondensatorami zalecanymi w jego nocie aplikacyjnej
- Kondensatorach tłumiących składowe w.cz. (standardowo 100nF między zasilaniem, a masą) przy każdej parze pinów zasilających

Uwaga!

Bez spełnienia powyższych punktów bardzo możliwe, że układ będzie działał, ale może się okazać, że działa on niestabilnie, samoczynnie się resetuje, bądź po prostu przestanie działać na skutek elektromigracji.

Zasilanie części analogowej

W większości miktrokontrolerów, które mają wbudowany przetwornik A/D wyprowadzone jest osobne zasilanie dla części analogowej. Należy wtedy skorzystać z filtru LC, typowe wartości: C = 100nF, L = 10μH



Pytanie

Co to jest za filtr i jakie ma parametry?

Zasilanie części analogowej

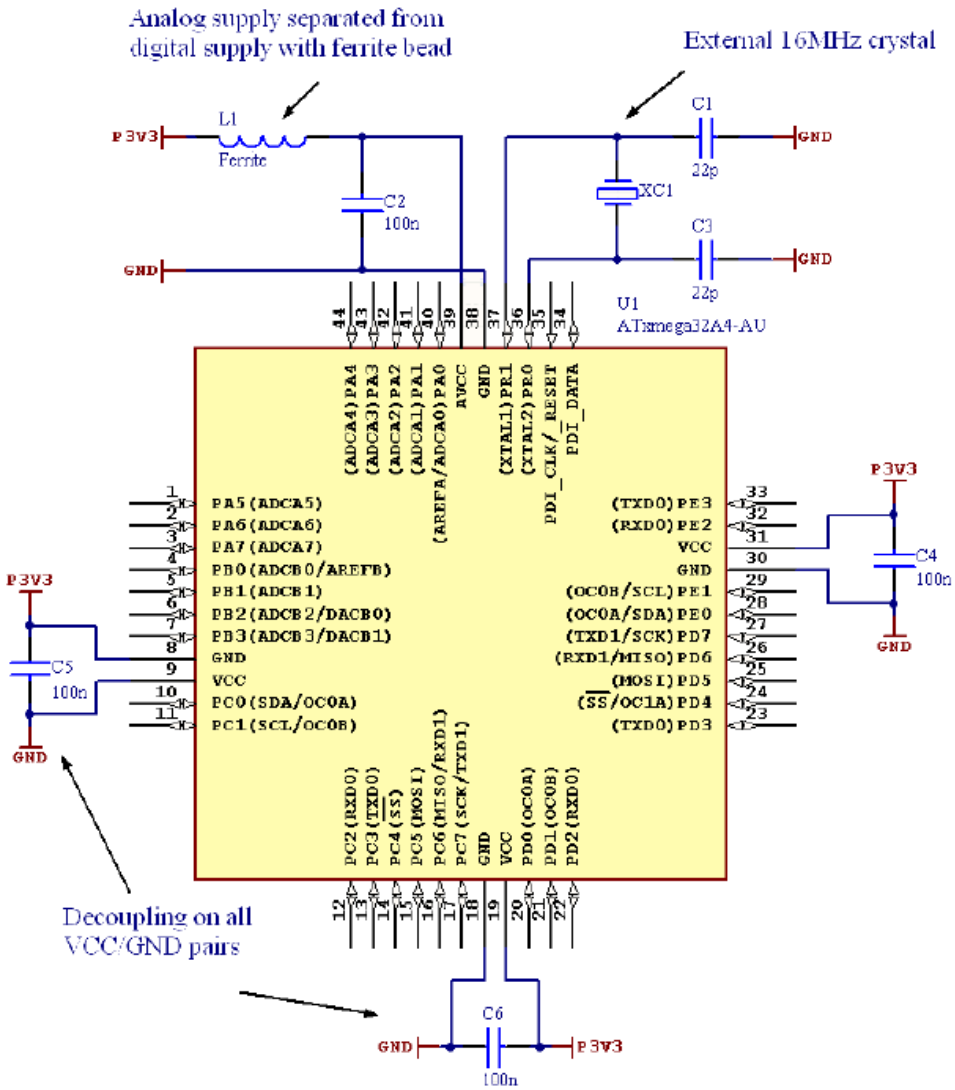
Można również skorzystać z zewnętrznego źródła napięcia odniesienia, należy przy tym pamiętać o następujących zależnościach.

$$0 < V_{ADC} < V_{REF}$$

$$0 < V_{REF} < V_{AVCC}$$

V_{ADC} - napięcie na wejściu przetwornika
V_{REF} - napięcie referencyjne
V_{AVCC} - napięcie zasilające przetwornik

Przykład dobrze podłączonego zasilania



Zewnętrzny kwarc

Najczęściej stosowanym źródłem zegara jest wbudowany oscylator, do którego podłączony jest zewnętrzny kwarc. Należy przy tym pamiętać:

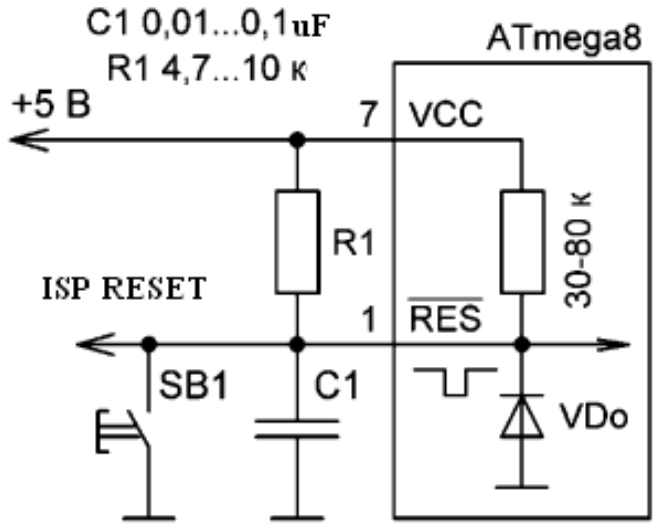
- Kwarc umieszczamy jak najbliżej procesora
- Do obydwóch nóżek podłączamy kondensatory do masy o wartości zależnej od kwarcu

Warto wiedzieć

Dostępne są w sprzedaży kwarce z wbudowanymi kondensatorami, mają one wtedy najczęściej cztery wyprowadzenia, do dwóch dodatkowych podłączamy masę.

Zewnętrzny reset

Warto dodać możliwość zresetowania uC przez wciśnięcie przycisku, można wtedy od razu dodać power-on reset wykorzystując układ RC (który jest zalecany jeśli mamy do czynienia z wolno narastającym napięciem zasilania).



Interfejsy komunikacyjne

Przyciski monostabilne

Najprostsze przyciski monostabilne, które się najczęściej wykorzystuje mogą przyjąć dwa stany - rozwarcie w stanie stabilnym, oraz zwarcie kiedy są wciśnięte. W celu uniknięta stanu nieustalonego przy rozwartych stykach należy stosować rezystor podciągający. Można skorzystać z wbudowanego rezystora pull-up i w takim przypadku przycisk musi po wciśnięciu zwierać do masy.

LEDy

Należy bezwzględnie pamiętać o stosowaniu rezystorów ograniczających prąd, ich wartość zależy od rodzaju diody, lecz nie można przekroczyć maksymalnej wartości prądu dla nóżki uC (w przypadku ATmega32 40mA). Bezpiecznie jest użyć rezystora o rezystancji 1kΩ.
Ze względu na różnice w ruchliwości dziur i elektronów zaleca się podłączać diody katodą do uC, tak żeby prąd płynął przez tranzystory typu n (do masy) w momencie ich włączenia.

Uwaga!

Maksymalna wartość prądu dla nóżki (40mA w przypadku ATmega32) jest wartością krytyczną. W praktyce nie powinno się do niej zbliżać. Warto także zwrócić uwagę na to, że wszystkie prądy płynące przez GPIO muszą także płynąć przez zasilanie.

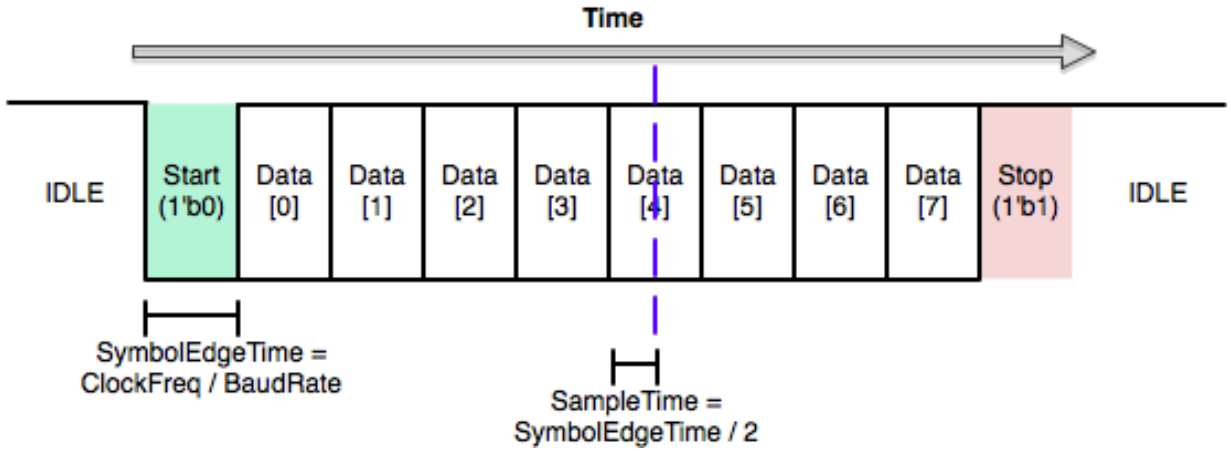
Interfejs UART

Jest to jeden z podstawowych interfejsów do komunikacji, wykorzystywany głównie do łączenia uC z komputerem. Jego cechy to:

- Interfejs szeregowy, full duplex, punt-punkt
- Dwie linie danych - RxD i TxD
- Możliwość sterowania przepływem poprzez linie RTS i CTS (w praktyce nieużywane)
- Standardowo 8-bitów danych
- Możliwość dodania bitu parzystości
- Prędkość transmisji od 2400bps do 2.5Mbps (standardowo 9600bps)

Ramka UART

Standardowa ramka UART składa się z: bitu startu, ośmiu bitów danych i bitu stopu, który trwa aż do następnej ramki. Opcjonalny bit parzystości dodawany jest za bitami danych i przed bitem stopu.

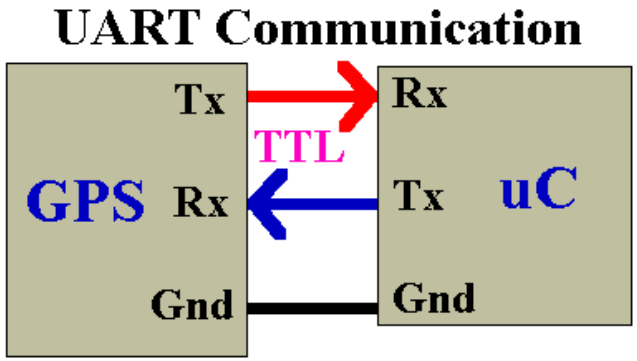


Warto wiedzieć

W celu uzyskania dokładnej prędkości transmisji można zastosować specjalny kwarc, którego wartość jest wielokrotnością biterate'u.

Podłączenie urządzenia przez UART

Dwa urządzenia z interfejsem UART łączymy krosując linie: TxD<->RxD i TxD<->RxD



Należy oczywiście pamiętać o podłączeniu masy. Standardowo UART jest interfejsem punkt-punkt, aczkolwiek można próbować podłączać więcej urządzeń stosując adresowanie oraz kontrolę przepływu danych.

Podłączenie komputera przez UART

W starych komputerach można znaleźć port RS-232, który od UART'a różni się tylko warstwą fizyczną, czyli poziomami napięć. Aby połączyć komputer z uC wykorzystując port RS-232 należy zastosować konwerter poziomów napięć, np. MAX232.

Drugą możliwością, bardziej uniwersalną, jest dołączenie konwertera USB<->UART np. FT232. Po podłączeniu jego pod USB oraz zainstalowaniu sterowników z poziomu komputera jest widoczny dodatkowy port szeregowy będący UART'em na wyjściu kostki FT232. Nie ma już wtedy potrzeby stosowania konwertera poziomów, można ten układ od razu podłączyć do uC.

Podłączenie komputera przez UART

W starych komputerach można znaleźć port RS-232, który od UART'a różni się tylko warstwą fizyczną, czyli poziomami napięć. Aby połączyć komputer z uC wykorzystując port RS-232 należy zastosować konwerter poziomów napięć, np. MAX232.
Drugą możliwością, bardziej uniwersalną, jest dołączenie konwertera USB<->UART np. FT232. Po podłączeniu jego pod USB oraz zainstalowaniu sterowników z poziomu komputera jest widoczny dodatkowy port szeregowy będący UART'em na wyjściu kostki FT232. Nie ma już wtedy potrzeby stosowania konwertera poziomów, można ten układ od razu podłączyć do uC.

Podsumowanie UART

- Bardzo popularny interfejs, można go spotkać w wielu urządzeniach
- Stosowany do łączenia dwóch urządzeń ze sobą, każde może rozpocząć transmisję w dowolnej chwili
- W prosty sposób można przejść na różnicowy RS-485 dodając transceivery (taka sama struktura ramki)
- Często w uC można znaleźć kilka niezależnych interfejsów UART, przez co można podłączyć więcej urządzeń
- W oparciu o UART można zbudować system do transmisji bezprzewodowej przez podczerwień (IRDA)

Interfejs SPI

Jest to szybki interfejs umożliwiający podłączenie wielu urządzeń peryferyjnych. Opiera się na topologii magistrali z osobną linią adresową podłączoną do każdego urządzenia. Jego główne cechy:

- Szeregowy, full duplex, topologia magistrali, jeden master i wiele slave'ów
- W jednej chwili transmisja tylko między masterem i jednym slave'em
- Trzy linie główne (MISO, MOSI, SCK) i jedna adresowa do każdego slave'a ($\overline{\mathrm{CS}}$)
- Transmisja słowo za słowo
- Długość słowa od 4 bitów do 16 bitów (standardowo 8 bitów)
- Liczba urządzeń podłączonych do magistrali ograniczona przez liczbę pinów u mastera
- Prędkość transmisji teoretycznie nieograniczona, w praktyce do 10Mb/s

Master i slave w SPI

Specyfikacja SPI wymaga występowania jednego mastera (nie może być więcej), reszta urządzeń pracuje jako slave. Cechy urządzenia master:

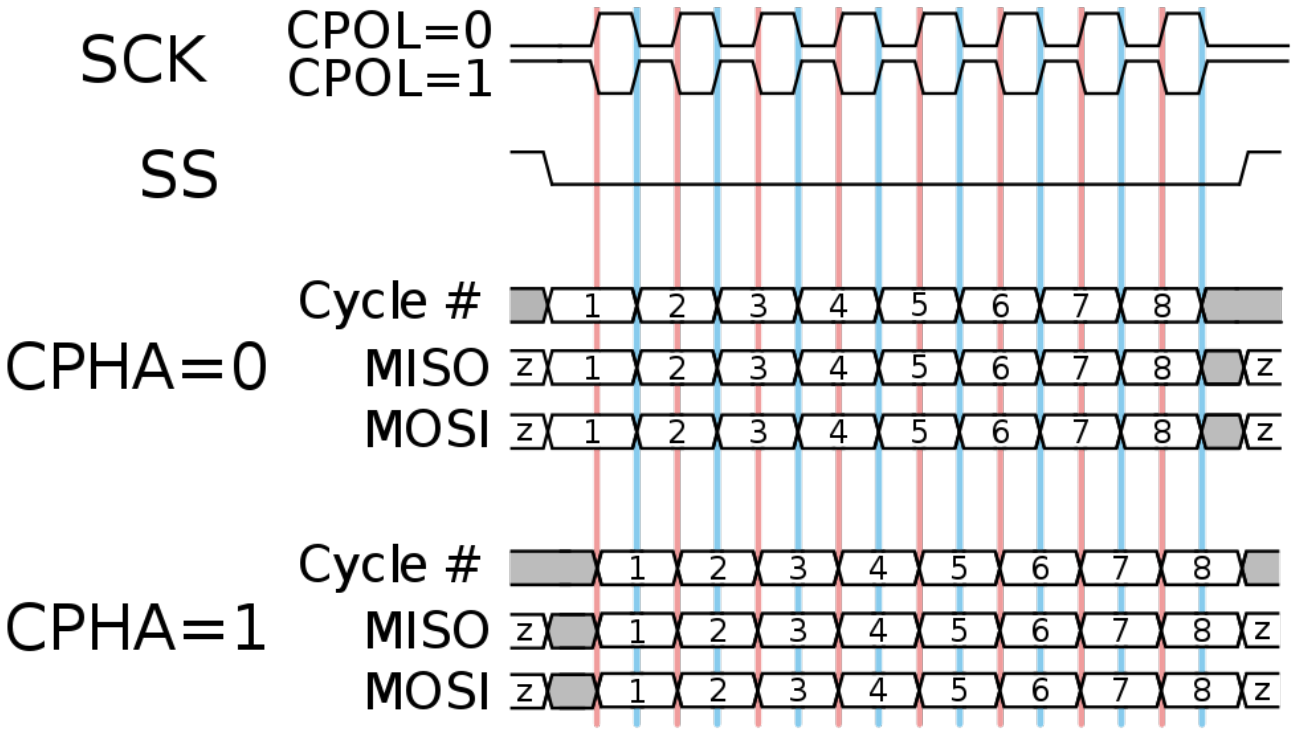
- Inicjuje transmisje
- Adresuje urządzenia typu slave
- Generuje zegar
- Może w dowolnej chwili przerwać transmisję

Slave jedyne co może, to w momencie aktywowania jego linii $\overline{\mathrm{CS}}$ wystawić porcję danych do rejestru wyjściowego.

Uwaga!

Linia $\overline{\mathrm{CS}}$ jest zanegowana, więc urządzenie jest wybrane jeśli na jego linii $\overline{\mathrm{CS}}$ jest stan niski.

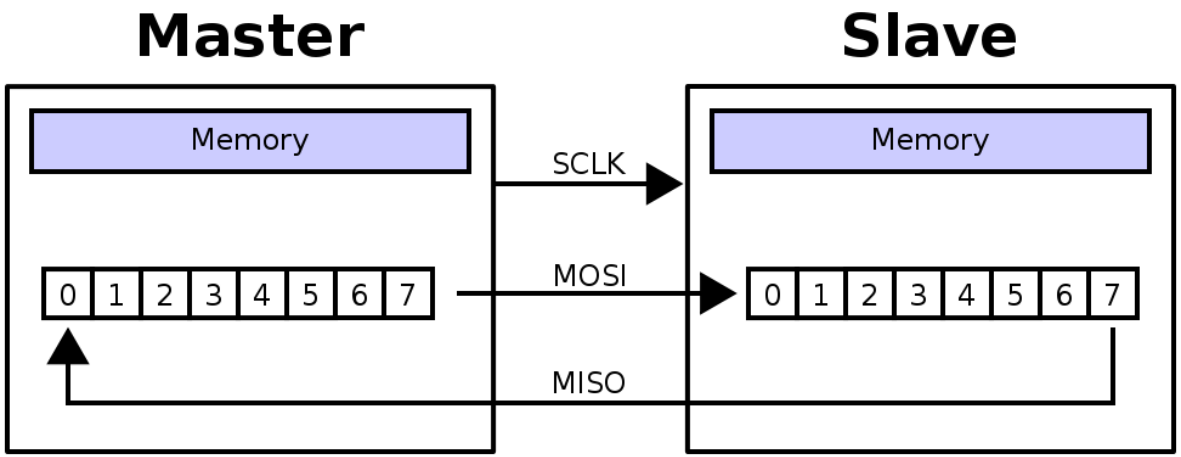
Ramka SPI



Warto pamiętać, że dostępne są aż cztery warianty ramki SPI, trzeba sprawdzić który z tych wariantów jest obsługiwany przez dane urządzenie peryferyjne i odpowiednio skonfigurować mastera.

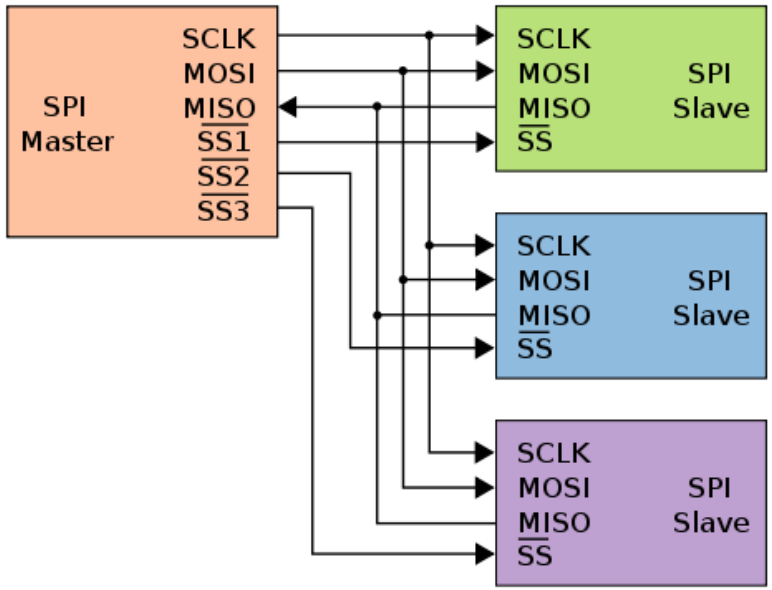
Zasada działania SPI

Aby zrozumieć działanie SPI i sposób w jaki przesyłane są dane warto zapamiętać poniższy rysunek.



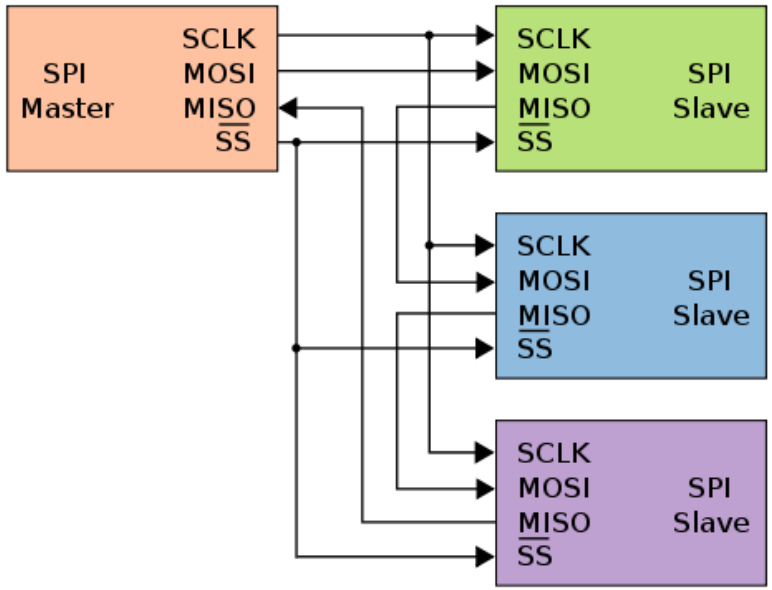
Głównym elementem interfejsu są dwa rejestry przesuwne, które przesuwają dane w takt zegara generowanego przez mastera. Oznacza to, że nawet jeśli chcemy coś odczytać z innego urządzenia to musimy równocześnie coś zapisać, żeby „wypchnąć" jego dane.

Łączenie urządzeń przez SPI (1)



Jest to najprostszy sposób na podłączenie wielu urządzeń, master dla każdego urządzenia ma przeznaczoną osobną linię adresową. Jednocześnie dane można wymieniać z jednym urządzeniem, poszczególne urządzenia nie muszą mieć takich samych parametrów SPI (prędkość, rodzaj ramki, długość słowa).

Łączenie urządzeń przez SPI (2)



Można też spotkać podłączenie w którym jednocześnie są wybrane wszystkie urządzenia i są one podłączone szeregowo do magistrali. Takie podłączenie wymaga zastosowania urządzeń o identycznych parametrach SPI. Jest to rzadziej spotykana konfiguracja.

Podsumowanie SPI

- Szybki i prosty w obsłudze interfejs szeregowy
- Specyfikacja nie ogranicza maksymalnej ilości urządzeń peryferyjnych, stosując odpowiednie bufory na liniach oraz dekodery adresowe można podłączyć znaczną ilość urządzeń
- Wymaga przynajmniej czterech linii, co może być problemem przy bardzo małych uC
- Nie wymaga dokładnego kwarcu, zegar jest generowany przez master dla wszystkich urządzeń
- Bardzo prosta budowa, bardzo prosty w implementacji np. w układzie FPGA (jeden rejestr przesuwny)
- Każda transmisja jest inicjowana przez urządzenie master
- Można spotkać inne nazewnictwa tego interfejsu - 3-wire, SSI

Interfejs I^2C

I^2C jest interfejsem typu magistrala, do którego można podłączyć wiele urządzeń peryferyjnych. Wykorzystuje porty typu open drain, wymaga rezystorów podciągających na wszystkich liniach, ma zaimplementowanie 7-bitowe adresowanie. W uC firmy Atmel pod nazwą TWI.

- Szeregowy, half-duplex, topologia magistrali, jeden master i wiele slave'ów, istnieje tryb multimaster
- W jednej chwili transmisja tylko między masterem i jednym slave'em
- Dwie linie - SCL i SDA, obydwie typu open drain, wymagają rezystorów podciągających
- 7-bitowy adres nadawany przy produkcji urządzenia
- Długość słowa 8 bitów
- Przy większej liczbie urządzeń problem z pojemnością linii
- Maksymalna prędkość 400kb/s, silnie zależy od konfiguracji magistrali (liczba urządzeń, długość)

Master w I^2C

Istnieje możliwość trybu multimaster, ale nie będzie on tutaj poruszony. Standardowa konfiguracja - jeden master, wiele slave'ów. Zadania mastera:

- Inicjuje transmisje
- Adresuje urządzenia typu slave
- Generuje zegar
- Generuje warunki startu i stopu
- Wybiera tryb transmisji
  - Master -> Slave
  - Slave -> Master

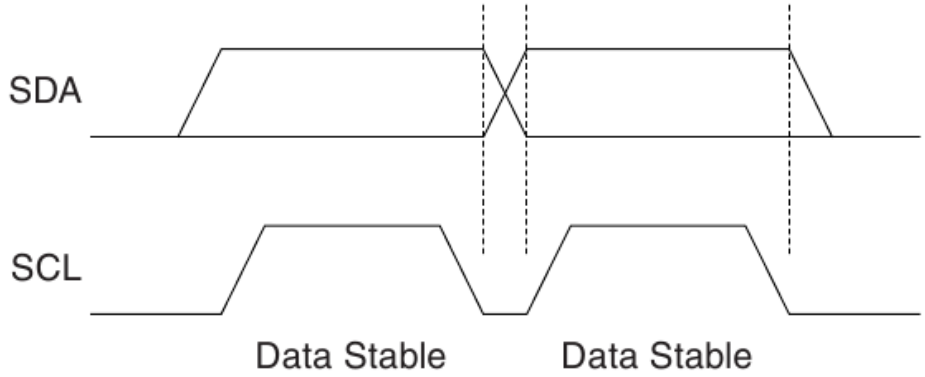
Slave w I^2C

Aktywacja urządzenia typu slave następuje w momencie wykrycia swojego adresu, następnie zależnie od kierunku komunikacji slave:

- Master -> Slave
  - Odpowiada ACK na swój adres
  - Odczytuje dane z magistrali
  - Odpowiada ACK na odczytane dane
- Slave -> Master
  - Odpowiada ACK na swój adres
  - Wysyła dane do mastera

Kierunek transmisji zależy od bitu $R/\overline{W}$

Przesyłanie bitów

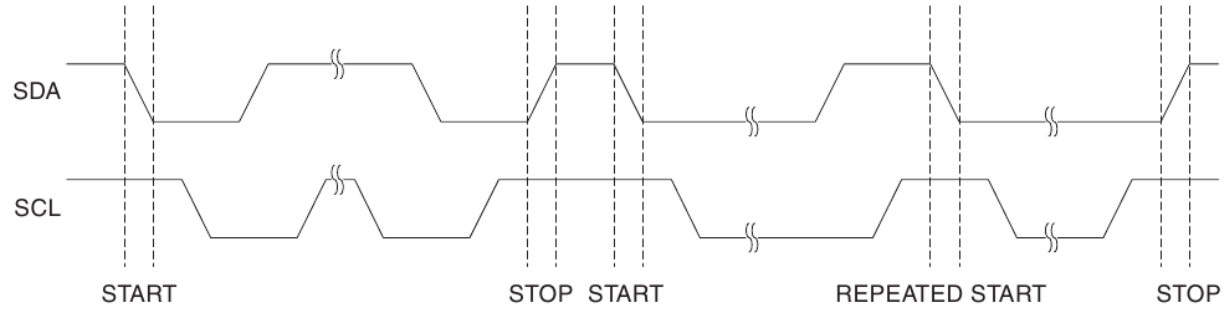


W magistrali I^2C bity na linii danych SDA zmieniają się tylko w chwili niskiego stanu na linii zegara SCL. Przez cały czas trwania stanu wysokiego na linii SCL sygnał na linii SDA musi pozostać taki sam.

Pytanie

Do czego można wykorzystać takie restrykcje?

Warunki START, REPEATED START, STOP



Każda transmisja musi zostać rozpoczęta warunkiem START, a zakończona warunkiem STOP. REPEATED START występuje w środku transmisji, kiedy chcemy zainicjować transmisje od początku bez warunku STOP.

Warto wiedzieć

REPEATED START wykorzystuje się na przykład kiedy chcemy najpierw coś zapisać do slave, a potem odczytać.

Bity ACK i R/$\overline{\mathrm{W}}$

Każda 8-bitowa paczka danych musi zostać potwierdzony przez urządzenie odbierające (slave lub master) bitem ACK. Przesyła się go jak normalny bit danych o wartości 0.

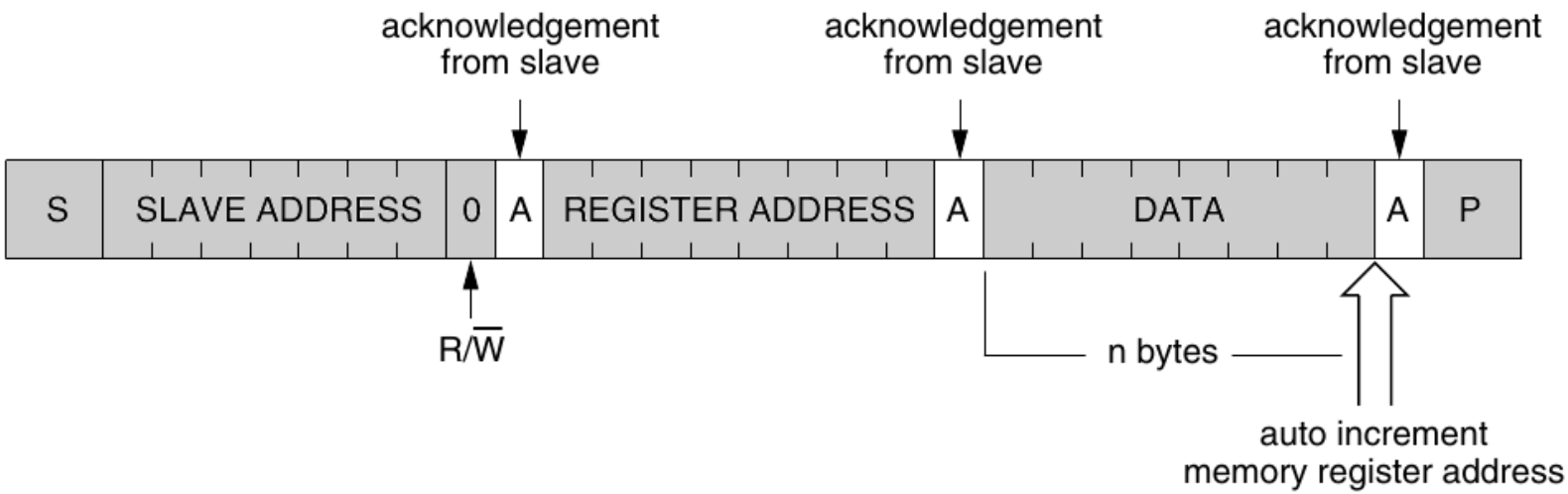
Bit R/$\overline{\mathrm{W}}$ wysyłany jest przez master do slave tuż po 7-bitowym adresie. Od niego zależy kierunek transmisji: 1 - slave -> master, 0 - master -> slave. Zgodnie z poprzednią regułą odebranie przez slave 8-bitowej paczki danych (adres + bit R/$\overline{\mathrm{W}}$) jest potwierdzane bitem ACK.

Bity ACK i R/$\overline{\mathrm{W}}$

Każda 8-bitowa paczka danych musi zostać potwierdzony przez urządzenie odbierające (slave lub master) bitem ACK. Przesyła się go jak normalny bit danych o wartości 0.

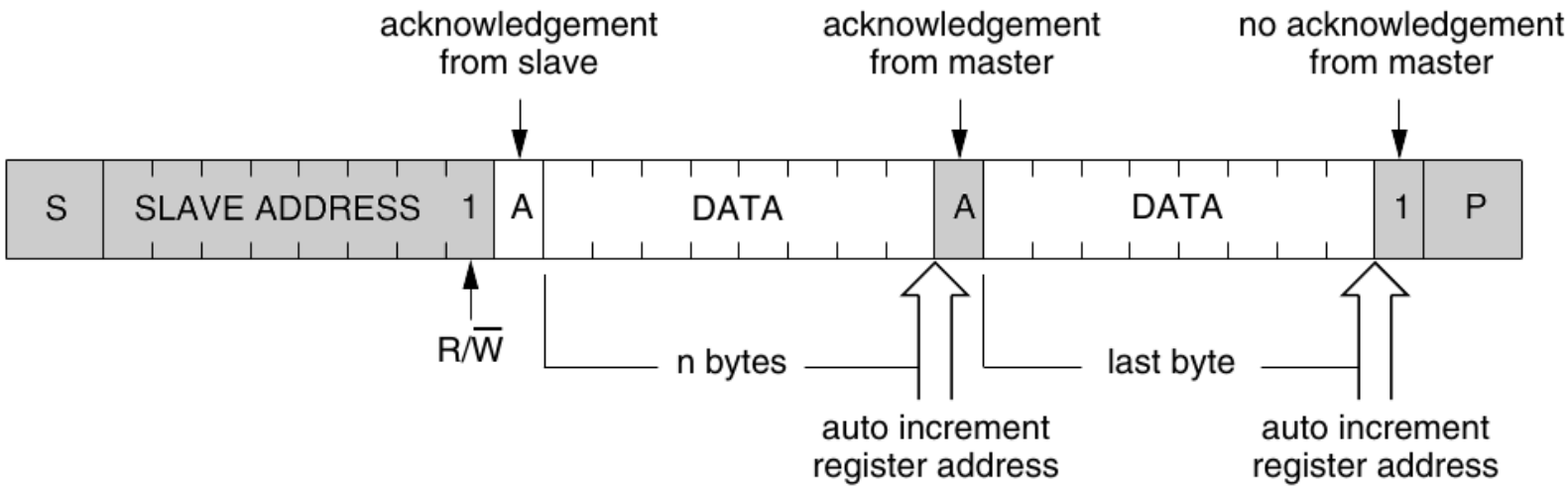
Bit R/$\overline{\mathrm{W}}$ wysyłany jest przez master do slave tuż po 7-bitowym adresie. Od niego zależy kierunek transmisji: 1 - slave -> master, 0 - master -> slave. Zgodnie z poprzednią regułą odebranie przez slave 8-bitowej paczki danych (adres + bit R/$\overline{\mathrm{W}}$) jest potwierdzane bitem ACK.

Ramka I^{2}C: Master -> Slave



Kolorem szarym zaznaczone bity wysyłane przez mastera. S - warunek START, A - ACK, P - warunek STOP.

Ramka I^{2}C: Slave -> Master

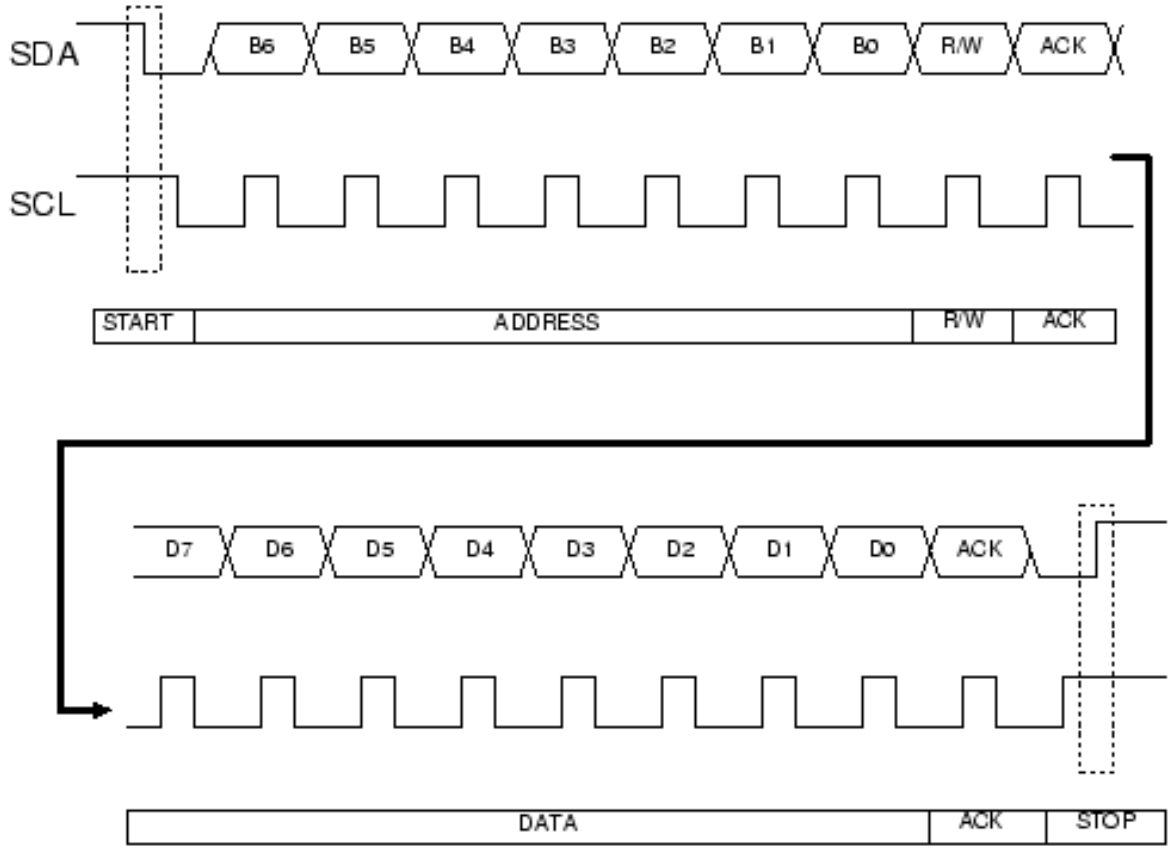


Oznaczenia jak na poprzednik slajdzie

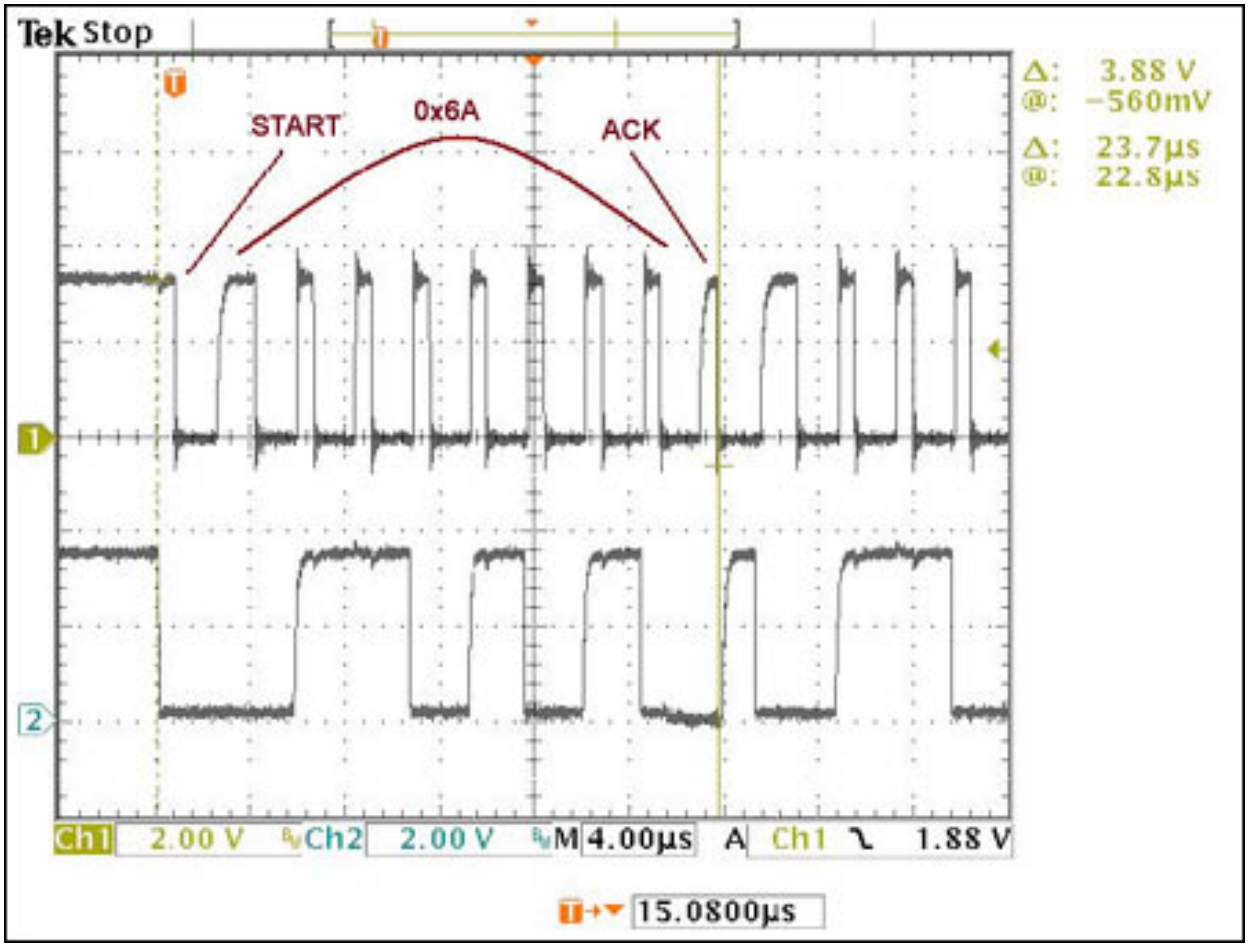
Zadanie

Narysować przebiegi I^{2}C dla którejś w powyższych przykładów ramki.

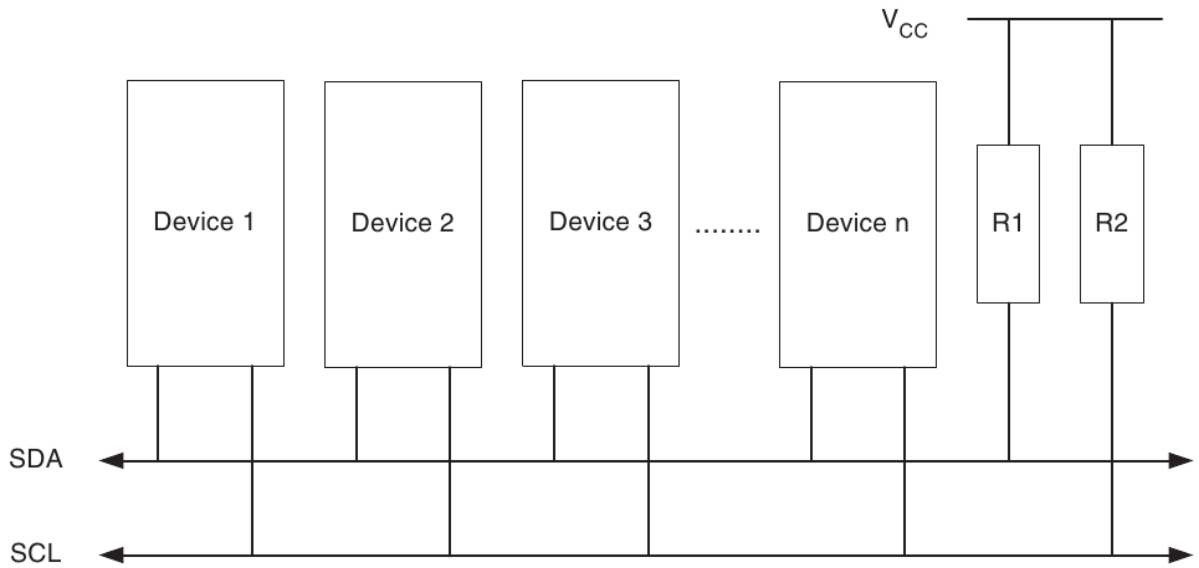
Ogólne przebiegi I^2C



Rzeczywiste przebiegi I²C



Łączenie urządzeń przez I^2C



Należy bezwzględnie pamiętać o rezystorach podciągających, typowe wartości to 10k, jeżeli magistrala jest mocno rozbudowana, albo zależy nam na dużej szybkości warto je zmniejszyć.

Niuanse w I^2C

I^2C jest interfejsem, który może sprawić problemy, dlatego warto nauczyć się jak wygląda ramka, jak wyglądają przebiegi, żeby mieć możliwość interpretacji pomiarów przy użyciu oscyloskopu. Największy problem występuje w momencie kiedy jakieś urządzenie ściągnie linie do masy i nie będzie chciało odpuścić. Jedno takie wadliwe urządzenie może zablokować całą magistralę.

Podsumowanie I^2C

- W miarę szybki interfejs szeregowy
- Teoretyczna maksymalna liczba urządzeń to 128, tyle można zaadresować na 7 bitach, w praktyce kilka
- Tylko dwie linie, niezależnie od liczby urządzeń
- Nie wymaga dokładnego kwarcu, zegar jest generowany przez master dla wszystkich urządzeń
- Skomplikowany protokół, łatwo się pomylić, trudno zrobić własną implementację
- Każda transmisja jest inicjowana przez urządzenie master
- W uC firmy Atmel nazwany został TWI (Two Wire Interface)

Porównanie interfejsów

Nazwa	Kierunki transmisji	Maksymalna liczba urządzeń	Szybkość	Prostota	Liczba linii
UART	full duplex	2	115,2kb/s	proste	2
SPI	full duplex	>2	10Mb/s	proste	min. 4
I^2C	half duplex	>2	400kb/s	trudne	2

Nazwa	master/slave	Gdzie można znaleźć
UART	nie	starsze komputery, GPS, telefony
SPI	tak	ADC, DAC, karty SD, wyświetlacze LCD
I^2C	tak	ADC, DAC, RTC, czujniki, EEPROM

Przygotowanie do programowania

Języki programowania

W 90% przypadków pisząc programy na uC korzysta się z języka C, gdyż zapewnia on dobry kompromis między czytelnością i prostotą kodu, a optymalnym wykorzystaniem zasobów i kontrolą nad wykonywaniem kodu. Czasami dodaje się wstawki asemblerowe, najczęściej w sytuacjach gdy:

- Jest potrzeba bardzo szybkiej obsługi przerwań
- Dostępne jest mało pamięci programu (np. w bootloaderze)
- Potrzeba dokładnie co do pojedynczego cyklu zegara wyznaczyć czas wykonywania kodu
- Program wymaga niekonwencjonalnego działania na stosie

Niektóre toolchain'y umożliwiają pisanie w C++, aczkolwiek nie jest to jeszcze język popularny w systemach embedded.

Środowisko dla AVR

W związku z tym, że AVR są bardzo prostymi uC, a jednocześnie drogi jest JTAG do nich najczęściej rezygnuje się z możliwości debuggowania. Najbardziej popularne zestawy programów działające pod Windows umożliwiające pisanie oprogramowania na AVR bez możliwości debuggowania to:

- WinAVR + notepad + Makefile + AVRDUDE
- WinAVR + Eclipse + AVR plugin + AVRDUDE

Dostępne jest także IDE od Atmela - AVRStudio, ale nie będzie one omawiane na tych warsztatach.

WinAVR

WinAVR jest to zestaw narzędzi potrzebnych do wygenerowania na podstawie kodu w języku C lub ASM pliku binarnego gotowego do wgrania na uC. Są to trzy podstawowe programy:

- Kompilator - zamienia kod napisany w języku C na kod w języku ASM
- Linker - łączy pliki wygenerowane przez kompilator w jeden plik
- Asembler - Zamienia kod wygenerowany przez linker w ASM na plik binarny, który można wgrać bezpośrednio do pamięci uC

W Linux'ie dostępne są dokładnie te same narzędzia, tylko pod nazwą avr-gcc.

Warto wiedzieć

Proces generowania kodu przez jedno urządzenie dla innego urządzenia, różniącego się architekturą nazywa się cross-kompilacją

Edytor tekstu

Jest to po prostu program umożliwiający tworzenie plików tekstowych. W najprostszej wersji może być to zwykły windowsowy notatnik, aczkolwiek warto skorzystać z edytora, który koloruje składnie. Do najbardziej popularnych edytorów można zaliczyć

- Crimson Editor - Windows
- Notepad++ - Windows
- Programmer's Notepad - Windows
- Kate - Linux
- Vim - Linux - dla prawdziwych twardzieli

Mimo, że zintegrowane IDE posiadają dużo ułatwień w pisaniu kodu, warto zacząć od takich edytorów, żeby dokładnie poznać proces tworzenia programów.

Makefile

Makefile to plik, który może sprawić na początku wiele problemów. Jest to zbiór reguł kompilacji, na które składa się:

- Jakimi narzędziami należy kompilować kod
- Jakie pliki są potrzebne
- W jakiej kolejności generować pliki (najpierw kompilacja, potem linkowanie)
- Jakie flagi, opcje kompilatora użyć
- Na jaki uC należy skompilować kod
- i wiele innych

Przykładowy Makefile, z którego warto skorzystać jest dostępny na stronie, razem z wykładami.

AVRDUDE

Ostatnim potrzebnym programem jest AVRDUDE. Umożliwia on wgranie gotowego kodu do procesora przy użyciu programatora. Wykorzystujący podany wcześniej Makefile nie trzeba się przejmować konfiguracją tego programu, lecz jest ona na tyle prosta, że warto ją poznać. Najczęściej wykorzystywane wywołanie tego programu to:

```
avrdude -p atmega32 -c usbasp -U flash:w:test.hex
```

Wykonując te polecenie zaprogramujemy pamięć flash (programu) uC ATmega32 przy użyciu programatora usbasp plikiem test.hex

Eclipse

Eclipse to zintegrowane środowisko programistyczne umożliwiające tworzenie kodu w praktycznie każdym języku na każdą platformę. Wiele profesjonalnych IDE jest oparta na Eclipse przez co środowisko te stało się pewnym standardem. Po zainstalowaniu pluginu do AVR mamy dostępne:

- Automatyczna konfiguracja toolchaina
- Automatyczne generowanie i edytowanie Makefile
- Zarządzanie plikami projektu
- Kolorowanie składni, przechodzenie do podfunkcji, zwijanie kodu
- Obsługa programu AVRDUDE

Linux jako OS dla programistów

Warto zainteresować się systemem operacyjnym Linux jako bazą dla programistów systemów wbudowanych. Obecnie jest to mój główny system w domu, oraz jedyny w pracy i jest to świadoma decyzja. Główne zalety tego systemu to:

- Ułatwiona obsługa konsoli
- Wbudowane wsparcie dla języków skryptowych
- Obsługa wirtualnych pulpitów

Oprogramowanie mikrokontrolerów z rodziny AVR

Fuse bity

Fuse bity to 16 bitów zapisywanych w pamięci flash służących do konfiguracji mikrokontrolera. Ustawiają one takie parametry jak:

- Źródło zegara i szybkość
- Włączenie lub wyłączenie ISP oraz JTAG
- Rozmiar bootloadera
- Poziom napięcia dla BOD

Uwaga!

Ze względu na możliwość wyłączenia możliwości programowania należy ustawiać fuse bity z dużą uwagą.

Fuse bity dla ATmega32

Table 104. Fuse High Byte

Fuse High Byte	Bit No.	Description	Default Value
OCDEN[4]	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN[5]	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN[1]	5	Enable SPI Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
CKOPT[2]	4	Oscillator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size (see Table 99 for details)	0 (programmed)[3]
BOOTSZ0	1	Select Boot Size (see Table 99 for details)	0 (programmed)[3]
BOOTRST	0	Select reset vector	1 (unprogrammed)

Fuse bity dla ATmega32

Table 105. Fuse Low Byte

Fuse Low Byte	Bit No.	Description	Default Value
BODLEVEL	7	Brown-out Detector trigger level	1 (unprogrammed)
BODEN	6	Brown-out Detector enable	1 (unprogrammed, BOD disabled)
SUT1	5	Select start-up time	1 (unprogrammed)(1)
SUT0	4	Select start-up time	0 (programmed)(1)
CKSEL3	3	Select Clock source	0 (programmed)(2)
CKSEL2	2	Select Clock source	0 (programmed)(2)
CKSEL1	1	Select Clock source	0 (programmed)(2)
CKSEL0	0	Select Clock source	1 (unprogrammed)(2)

Rejestry GPIO

W mikrokontrolerach z rodziny AVR do obsługi GPIO wykorzystuje się trzy rejestry:

- DDRx - rejestr kierunku (1 wyjście, 0 wejście)
- PORTx - rejestr danych (jeśli wyjście) lub rejestr włączający rezystory pull-up (jeśli wejście)
- PINx - rejestr do odczytywania stanu bezpośrednio z nóżki

W miejsce x należy wstawić literę portu.

Uwaga!

Można czytać dane z rejestru PORTx, jednakże odczytuje się wtedy ostatnią wartość wprowadzoną do niego, a nie faktyczną wartość na nóżce.

Przykład wykorzystania

Wszystkie piny GPIO są pogrupowane po osiem sztuk w jednym porcie oznaczonym literą. Oznacza to, że jeden bit każdego z rejestru odpowiada jednemu pinowi GPIO w danym porcie, w taki sposób, że MSB oznacza pin nr 7, a LSB pin nr 0. Aby ustawić piny PD2 i PC3 na wyjścia z wartościami 1, a PD1 i PA2 na wejścia i na pinie PA2 włączyć rezystor podciągający należy ustawić następujące wartości:

- bit 2 DDRD na 1 oraz bit 3 DDRC na 1
- bit 2 PORTD na 1 oraz bit 3 PORTC na 1
- bit 1 DDRD na 0 oraz bit 2 DDRA na 0
- bit 1 PORTD na 0 oraz bit 2 PORTA na 1

Realizacja w języku C

W celu zmiany tylko pojedynczych bitów w rejestrze należy zastosować iloczyn logiczny (AND) w przypadku ustawienia bitu na 0 lub sumę logiczną (OR) w przypadku ustawienia bitu na 1. Instrukcje z poprzedniego slajdu napisane w C:

- DDRD |= (1 « 2); DDRC |= (1 « 3)
- PORTD |= (1 « 2); PORTC |= (1 « 3)
- DDRD &= ~(1 « 1); DDRA &= ~(1 « 2)
- PORTD &= ~(1 « 1); PORTA |= (1 « 2)

Do sprawdzenia stanu pinu pinu warto skorzystać z makra bit_is_set lub bit_is_clear, przykład - sprawdzenie, czy PB1 ma stan wysoki:

- bit_is_set(PIN1, PB1)

Pytanie

Jakie są wartości tych rejestrów po resecie i dlaczego?

Rejestry UART-a

Interfejs UART posiada wiele rejestrów konfiguracyjnych, jednakże w 90% korzysta się tylko z następujących

- UBRRH - wybór prędkości, 8 starszych bitów
- UBRRL - wybór prędkości, 8 młodszych bitów
- UCSRB - rejestr konfiguracyjny, posiada bity TXEN i RXEN odpowiedzialne za włączenie transmittera i receivera
- UDR - rejestr danych

Można też konfigurować takie parametry jak bit parzystości, liczbę bitów danych, liczbę bitów stopu, włączać przerwania itp. Wartości domyślne tych parametrów są najczęściej właściwie, więc nie ma potrzeby ich zmieniać.

Konfiguracja UART-a

W celu skonfigurowania modułu UART w domyślnym trybie należy wykonać dwie czynności

- w rejestry UBRRH i UBRRL wpisać wartości dzielnika, tak aby uzyskać pożądaną prędkość transmisji
- ustawić na 1 bity TXEN i RXEN rejestru UCSRB

W celu uzyskania 16-bitowej wartości dzielnika warto skorzystać z zależności:

```
ubrr = F_CPU / 16 / baudrate - 1
```

Wymagane jest zdefiniowanie wcześniej F_CPU, a w miejsce baudrate należy wpisać pożądaną prędkość transmisji.

Wysyłanie danych przez UART-a

W celu wysłania danych, zakładając, że moduł jest już skonfigurowany, należy tylko wpisać wartość, którą chcemy wysłać do rejestru UDR. Czynność ta automatycznie zainicjuje całą transmisję.

W celu uniknięcia sytuacji w której poprzednie dane nie zostały jeszcze wysłane, a program chce wysłać nowe należy przez zapisem do rejestru UDR sprawdzać stan bitu UDRE w rejestrze UCSRA. Transmisja może być rozpoczęta tylko i wyłącznie kiedy jest on ustawiony na 1. Można skorzystać z warunku:

```
UCSRA & (1 << UDRE)
```

Wysyłanie danych przez UART-a

W celu wysłania danych, zakładając, że moduł jest już skonfigurowany, należy tylko wpisać wartość, którą chcemy wysłać do rejestru UDR. Czynność ta automatycznie zainicjuje całą transmisję.

W celu uniknięcia sytuacji w której poprzednie dane nie zostały jeszcze wysłane, a program chce wysłać nowe należy przez zapisem do rejestru UDR sprawdzać stan bitu UDRE w rejestrze UCSRA. Transmisja może być rozpoczęta tylko i wyłącznie kiedy jest on ustawiony na 1. Można skorzystać z warunku:

```
UCSRA & (1 << UDRE)
```

Konfiguracja UART-a - przykład w C

```
void UartInit(uint16_t baudrate)
{
    uint16_t ubrr = F_CPU / 16 / baudrate - 1;
    UBRRH = (uint8_t)(ubrr >> 8);
    UBRRL = (uint8_t)ubrr;
    UCSRB = (1 << TXEN)|(1 << RXEN);
}
```

Wysyłanie danych przez UART-a - przykład w C

```
void UartPutc(char c)
{
    while (!(UCSRA & (1 << UDRE)));
    UDR = c;
}

void UartSend(char* s)
{
    while (*s)
        UartPutc(*s++);
}
```