Akademia
Górniczo-Hutnicza
w Krakowie
Instytut Elektroniki

Laboratorium mikrokontrolerów

Ćwiczenie 1

Wprowadzenie

Autor: Paweł Russek & Sebastian Koryciak

wer. 9.03.21

1. Wprowadzenie

1.1. Cel ćwiczenia

Głównym celem ćwiczenia jest zapoznanie studenta z elementami składowymi zestawu do ćwiczeń z "Laboratorium mikrokontrolerów".

Części, które tworzą laboratorium stanowią środowisko rozwoju oprogramowania dla mikrokontrolera ATmega32 firmy Microchip (dawniej Atmel). Pojedyncze stanowisko składa się z:

- Komputera PC z systemem operacyjnym Microsoft Windows,
- zestawu uruchomieniowego ATmega328PB_Xplained_mini,
- oprogramowania Microchip Studio 7.0.

Wstęp instrukcji zawiera podstawowe informacje na temat rodziny mikrokontrolerów megaAVR firmy Microchip. Następnie przedstawiono oprogramowanie Microchip Studio w wersji 7.0 oraz płytkę uruchomieniową ATmega328PB Xplained mini.

W celu umożliwienia studentowi rozpoczęcia samodzielnego programowania mikrokontrolera w języku assembler podano podstawowe informacje na temat składni tego języka. Następnie, aby student mógł wykonać pierwsze ćwiczenia praktyczne tj. napisać i uruchomić prosty program, instrukcja przedstawia elementarne instrukcje umożliwiające manipulację portami we/wy mikrokontrolera.

Na zakończenie ćwiczenia student wykona proste zadania weryfikujące nabytą wiedzę i umiejętności.

1.2. Konieczne wiadomości wstępne

- Znajomość postaw elektroniki cyfrowej: bramki, rejestry, maszyny stanów skończonych.
- Znajomość podstaw informatyki: reprezentacja liczb w zapisie binarnym, podstawy programowania (instrukcje przypisania, warunkowe, pętle)

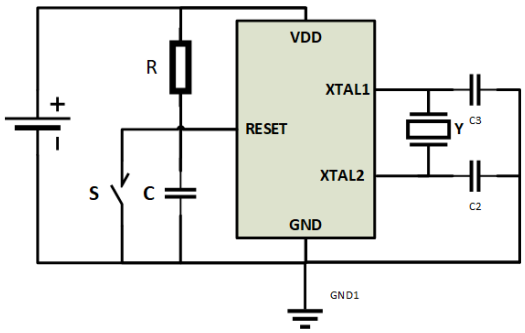
2. Architektura mikrokontrolerów megaAVR

W instrukcji zostaną podane tylko najbardziej podstawowe informacje na temat mikrokontrolerów megaAVR. Instrukcja nie podaje pełnego opisu architektury tych mikrokontrolerów. Student powinien samodzielnie poszerzyć znajomość tematu korzystając z dodatkowych materiałów. Na przykład:

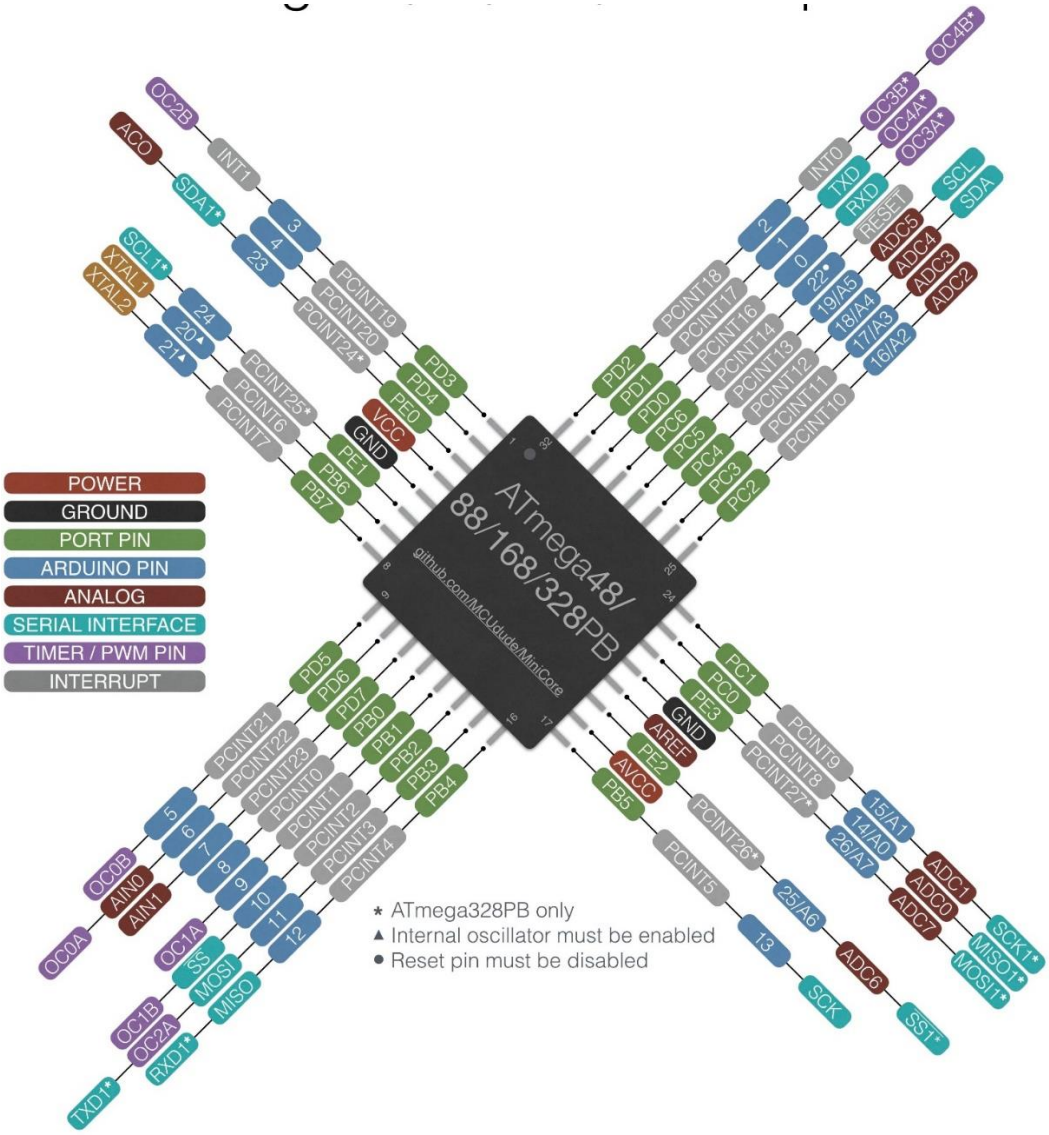
- **Wykład i materiały wykładowe,**
- **Datasheet: „ATmega328PB complete datasheet"** (dostępne na stronie laboratorium),
- **John Morton, „AVR Introductory course",**
- **Rafał Baranowski, „Mikrokontrolery AVR ATmega w praktyce",**
- **http://www.avrbeginners.net/**
- i wiele, wiele innych źródeł internetowych i książek. Rodzina megaAVR jest bardzo bogato opisana z racji swojej wieloletniej już popularności.

Mikrokontrolery AVR firmy Atmel to procesory 8 bitowe o architekturze typu RISC. Mikrokontrolery te integrują w jednej strukturze krzemu podsystem CPU, pamięć programu

i danych oraz elementy peryferyjne (np. porty we/wy, liczniki, przetworniki AC i CA). W celu uruchomienia, mikrokontrolery AVR wymagają minimalnej liczby komponentów zewnętrznych. W praktyce wystarczy tylko zasilanie i układ generujący sygnał RESET. Typowo na obwód RESET" składają się rezystor podciągający do zasilania („pull-up resistor") oraz przełącznik. W układach AVR sygnał RESET jest aktywny niskim poziomem napięcia. Mikrokontroler AVR może wykorzystywać wewnętrzny, wbudowany oscylator zegarowy, ale układy mogą korzystać również z zegara/oscylatora zewnętrznego.



Poniżej na rysunku przedstawiono konfigurację połączeń mikrokontrolera ATmega328PB w laboratoryjnym zestawie XPlained.



Widoczne na rysunku końcówki układu („pins”) realizują funkcje równoległego we/wy („parallel ports”):

- Port B (bity B0 do B7): końcówki 7, 8, 12 - 17.
- Port C (bity C0 do C6): końcówki 23 do 29.
- Port D (bity D0 do D7): końcówki 30 do 32, 1, 2, 9 - 11.
- Port E (bity E0 do E3): końcówki 3, 6, 19 i 22.

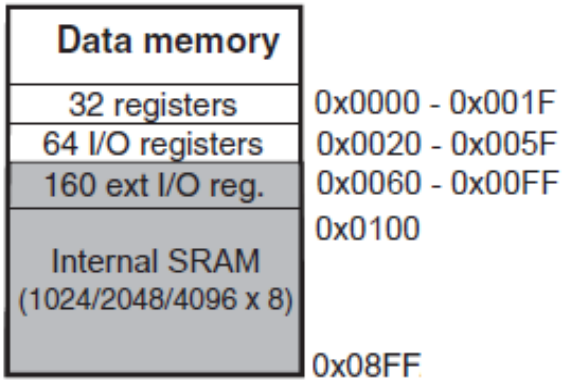
Porty we/wy są zapisywane i odczytywane poprzez zapis i odczyt odpowiednich rejestrów mikrokontrolera. Stany bitów tych dedykowanych rejestrów odpowiadają stanom logicznym końcówek mikrokontrolera, które odpowiadają za bity danego portu. Bity portów mogą być wykorzystane do zapalenia podłączonej do nich diody LED lub do odczytania stanu podłączonego przełącznika napięcia (albo przycisku). Dodatkowo, niektóre końcówki mikrokontrolera oprócz funkcji bitów portów mogą służyć do komunikacji z wbudowanymi w układ urządzeniami peryferyjnymi (takimi jak na przykład: interfejs komunikacji szeregowej UART, przetwornik analogowo-cyfrowy, itp). Większość oferowanych mikrokontrolerów megaAVR ma wbudowane liczniki, układy ADC, układy UART, I2C, SPI. Istnieją również wersje z wbudowanymi portami USB, CAN i innymi modułami.

2.1. Architektura mikrokontrolerów megaAVR

Wewnętrzna architektura AVR, to tak zwana architektura harwardzka. Oznacza to, że pamięć programu i pamięć danych są oddzielnymi blokami pamięci. Dzięki takiemu rozwiązaniu mikrokontroler szybciej wykonuje instrukcje, ponieważ odczyt programu może następować równocześnie z odczytem i zapisem danych.

Pamięć programu mikrokontrolera AVR to pamięć nieulotna typu FLASH. Pamięć ta może być zapisywana przez zewnętrzny programator. Z punktu widzenia CPU mikrokontrolera jest to pamięć o dostępie typu ROM (Read Only Memory). Rozmiar pamięci programu jest zmienna w zależności od typu układu AVR i waha się od 1kB do 256kB. Wykorzystywany na laboratorium układ może zmieścić 32 KByte (32×1024 bajty) programu. Wato wspomnieć, że instrukcje mikrokontrolera AVR są 16 bitowe i dlatego pamięć FLASH jest zorganizowana w słowa 16 bitowe. Mamy więc 16 K 16 bitowych słów.

Pamięć danych mikrokontrolera AVR jest pamięcią o dostępie typu RAM (Random Access Memory). Jest to pamięć ulotna typu SRAM. Mapa tej pamięci jest przestawiona na rysunku poniżej.



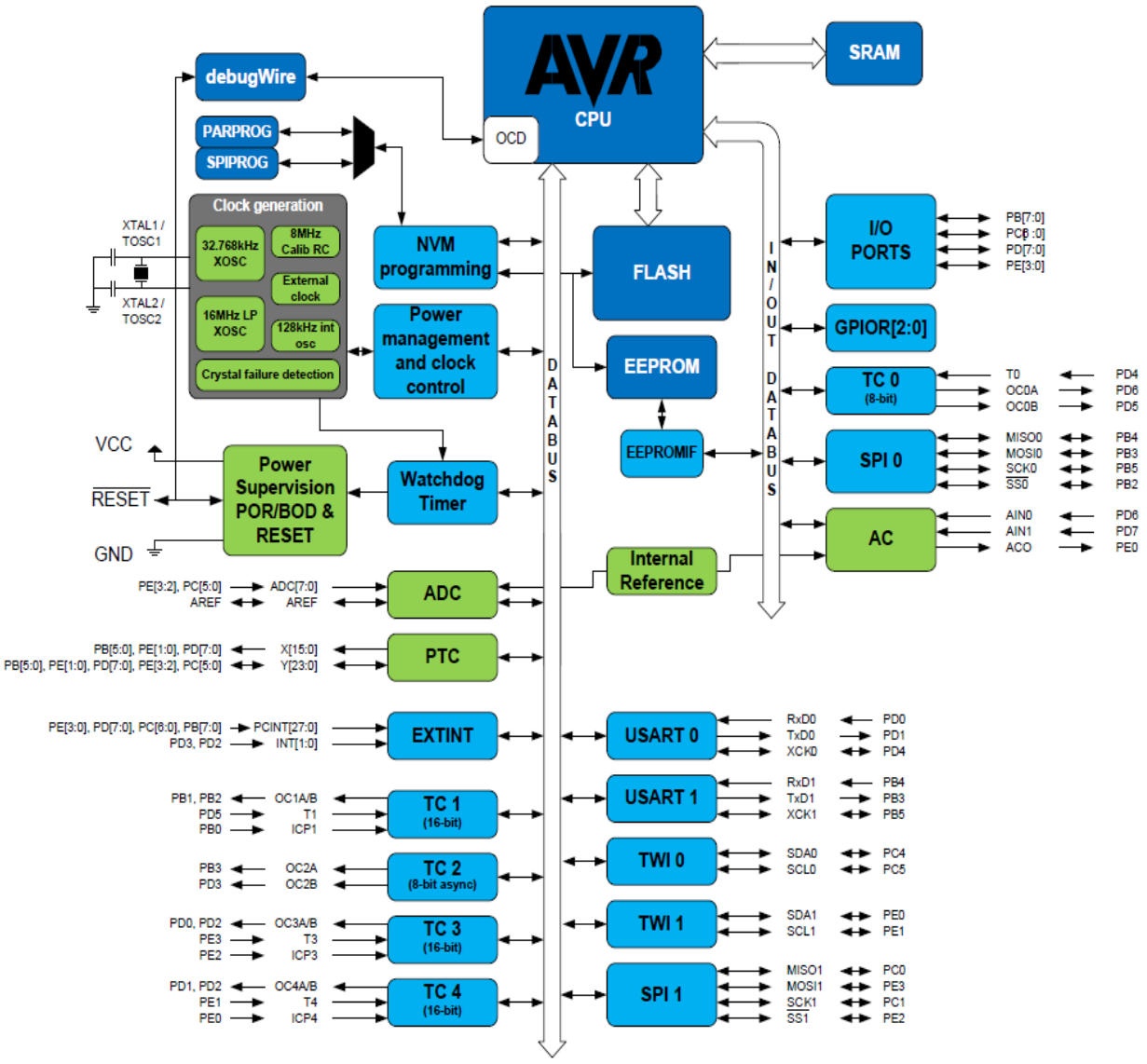
Pamięć danych jest to pamięć 8 bitowa. Maksymalnie AVR może zaadresować 64kB przestrzeni adresowej. ATmega328PB oferuje jednak tylko 2KB pamięci danych RAM.

We wszystkich mikrokontrolerach megaAVR, pierwsze 32 bajty w pamięci RAM są wykorzystywane przez CPU jako rejestry ogólnego przeznaczenia. Rejestry te są oznaczone kolejno jako R0, R1, … R31. Ogólnie R*n.*

Reszta pamięci RAM jest wykorzystywana jako rejestry układów peryferyjnych we/wy („I/O registers") oraz pamięć operacyjna. Rozmiar przestrzeni dla rejestrów we/wy zależy od rodzaju układu z rodziny megaAVR. Przestrzeń ta ma w architekturze AVR specjalne zastosowanie, ponieważ jest to przestrzeń adresowa zarezerwowana dla rejestrów przeznaczonych do komunikacji z urządzeniami peryferyjnymi. Obszar pamięci operacyjnej jest przeznaczony do przechowywania danych podczas wykonywania przez CPU programu.

Dodatkowo, układy AVR oferują mały obszar pamięci nieulotnej typu EEPROM. Jest to pamięć, która może być wykorzystywany do przechowywania danych które nie powinny zniknąć po wyłączeniu zasilania mikrokontrolera. CPU może zapisywać i odczytywać pamięć EEPROM.

Rysunek poniżej przedstawia schemat blokowy architektury mikrokontrolera AVR.



Ćwiczenie 1.1

Otwórz, dostępny na stronie kursu na UPEL, dokument"ATmega328PB complete datasheet".

1. Wskaż i nazwij przynajmniej trzy charakterystyczne cechy układów AVR wymienione w dokumencie.
2. Podaj rozmiar pamięci EEPROM dla układu ATmega328PB.
3. Odszukaj końcówki XTAL1 i XTAL2. Jaka jest funkcja tych końcówek?
4. Sprawdź czym różnią się układy ATmega328PB od układów ATmega328P.
5. Od czego pochodzi skrót PTC? Do czego służą te peryferia?

3. Oprogramowanie Microchip Studio

Microchip Studio (dawne Atmel Studio) to zintegrowane środowisko programistyczne do programowania i debugowania mikrokontrolerów firmy Microchip.

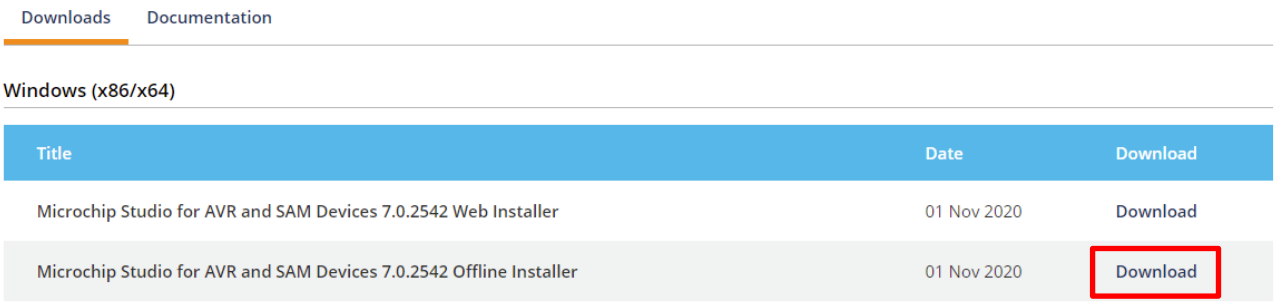
Ćwiczenia 1.2a

Sprawdź, czy oprogramowanie Microchip Studio 7.0 jest zainstalowane na twoim komputerze. Znajdź ikonkę z obrazkiem biedronki w menu startowym systemu Windows. Jeżeli oprogramowanie jest zainstalowane możesz przejść do ćwiczenia 1.2b.

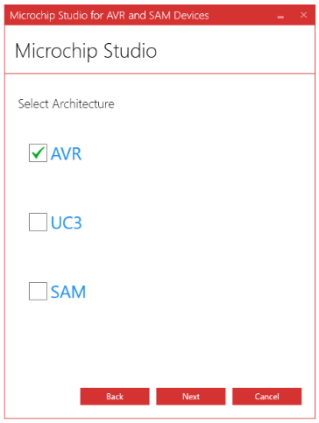
Ściągnij oprogramowanie Microchip Studio ze strony firmy Microchip:

https://www.microchip.com/en-us/development-tools-tools-and-software/microchip-studio-for-avr-and-sam-devices

Odszukaj na wskazanej stronie aktualny link do oprogramowania **Microchip Studio for AVR and SAM Devices 7.0.xxxx Offline Installer** (jak na obrazku poniżej).



Uruchom pakiet instalacyjny i wykonuj kolejne polecenia programu.

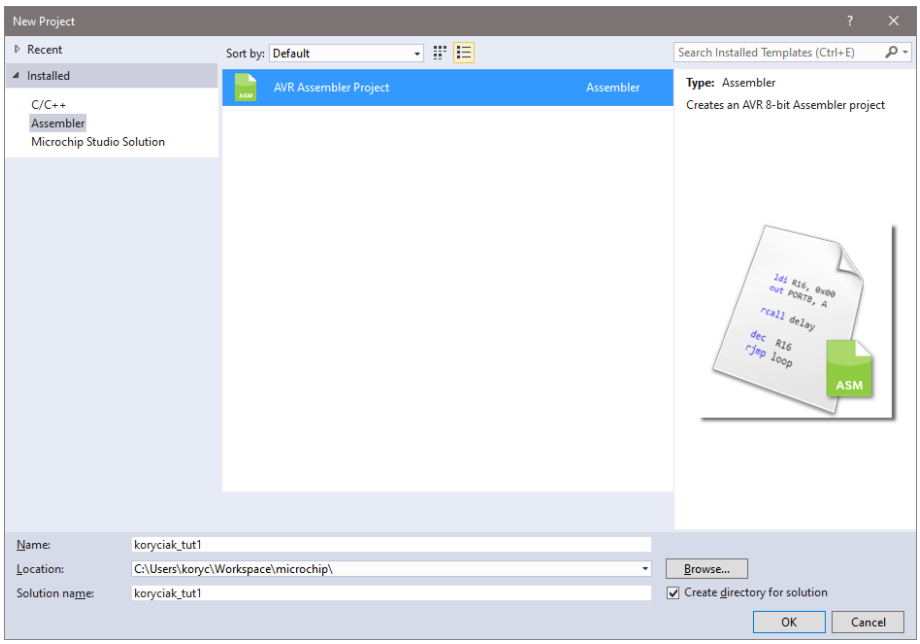


- Wybierz instalację pełnej wersji programu.
- Skorzystaj z domyślnej proponowanej ścieżki programu.
- Zainstaluj tylko pakiet dotyczący architektury AVR.
- Pozostałe ustawienia zostaw domyślne.
- Program jest darmowy i nie wymaga żadnej rejestracji.

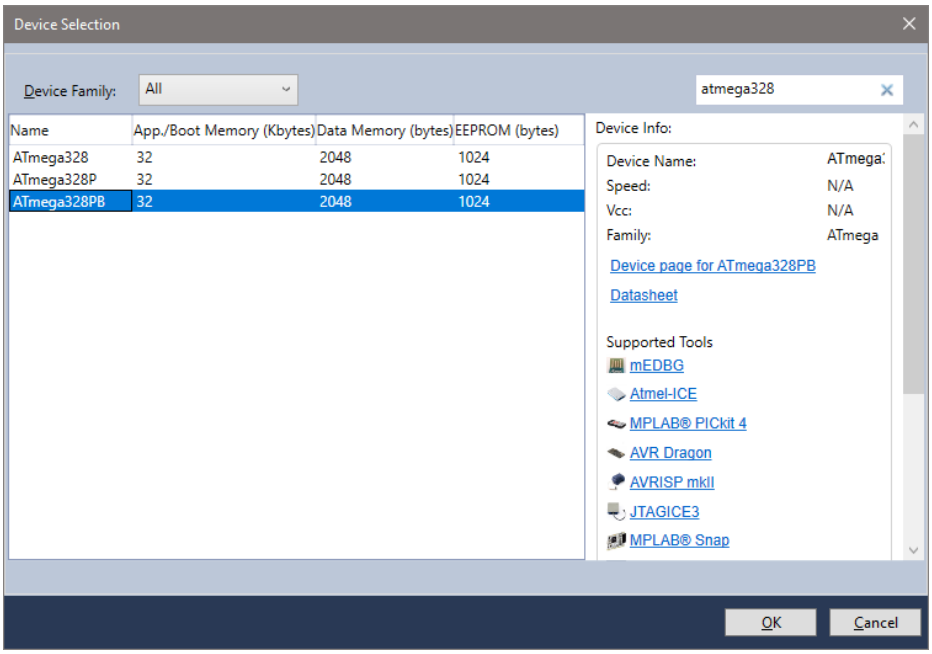
Ćwiczenie 1.2b

Uruchom Microchip Studio 7.0 i utwórz nowy projekt typu „Assembler project”

1. Kliknij “Microchip Studio 7.0” w menu start systemu Windows.
2. Z menu wybierz “File->New-> Project” lub w oknie “Start Page – Microchip Studio” wybierz link „New project”.
3. W oknie “New Project”, wybierz “Assembler” i “AVR Assembler Project”
4. Opcjonalnie, zmień nazwę swojego projektu w polu “Name” i ścieżkę dostępu w polu „Location”. Jeżeli z komputera, którego używasz korzysta wielu studentów warto w ścieżce uwzględnić swoje nazwisko: na przykład „c:/*shrek_tm_lab_czw_g900/cw1*”.

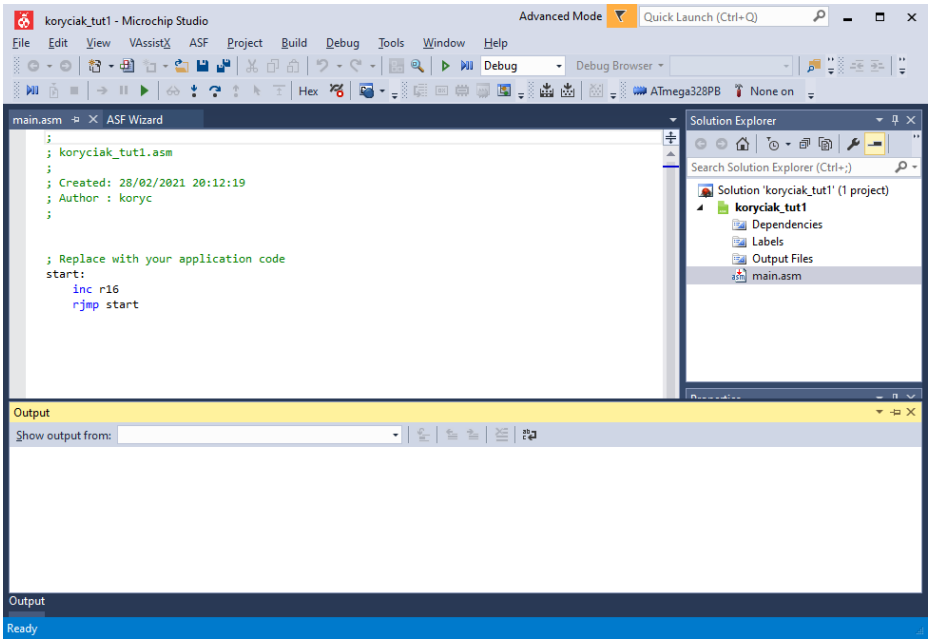


5. Kliknij 'OK'
6. Wybierz mikrokontroler ATmega328PB



7. Kliknij 'OK'

8. Właśnie utworzyłeś nowy projekt. Microchip Studio otworzył widok podstawowego okna roboczego.



4. Podstawy języka AVR assembler

Kompilator języka assembler (w sktócie Assembler) tłumaczy napisany przez programistę kod assemblera na kod maszynowy, który jest dostępny w plikach o różnym formacie. Np. obiektowy plik z kod maszynowym może być wykorzystywany do symulacji działania mikrokontrolera. Dodatkowo assembler tworzy plik binarny, który może być bezpośrednio (lub pośrednio po dodatkowej konwersji) wykorzystany do zaprogramowania pamięci FLASH mikrokontrolera.

4.1. Kod w języku assembler

Kompilator assemblera jako swoje wejście wykorzystuje plik z kodem, który zawiera instrukcje kodu maszynowego („instruction") zapisane w postaci mnemonicznej. Dodatkowo kod assemblera może zawierać etykiety („label") i dyrektywy („directives").

Pojedyncza linia kodu w pliku wejściowym jest ograniczona do 120 znaków. Każdą linię opcjonalnie rozpoczyna etykieta („label"), która jest ciągiem alfanumerycznym który kończy się znakiem dwukropka (':'). Etykiety są wykorzystywane przez instrukcje skoków i rozgałęzień do określenia adresu skoku w pamięci ROM oraz przez instrukcje odczytu i zapisu rejestrów do określenia adresu w pamięci RAM odpowiedniego dla ładowanej danej.

Linia assemblera może przybrać jedną z czterech form:

1. [label:] directive [operands] [Comment]

2. [label:] instruction [operands] [Comment]

3. Comment

4. Empty line

Komentarz rozpoczyna się od znaku średnika (';')

;[Text]

W składni przedstawionej powyżej elementy w nawiasach kwadratowych są opcjonalne. Każdy tekst umieszczony po znaku ';' i przed końcem linii (symbol EOL) jest ignorowany przez assembler.

Przykłady:

```
; Comment line
label:.EQU count=100    ;Set COUNT to 100 (Directive)
test: ldi r21, count    ; Load value 'count' to register r21 (Instruction)
      jmp test          ; Infinite loop (Instruction)
```

Instrukcje są przeznaczone do wykonywania przez CPU, natomiast dyrektywy są interpretowane jedynie przez assembler i nie pozostawiają śladu w kodzie maszynowym.

4.2. Dyrektywa EQU

Dyrektywa EQU określa symbol i przypisuje mu podaną przez programistę wartość. Dyrektywa ta jest używana w celu zadeklarowania i zdefiniowania stałych wartości wykorzystywanych przez programistę w kodzie programu. Typowo tą dyrektywę wykorzystuje się do określenia stałego adresu lub do ustalenia stałej wartości danej. W trakcie kompilacji assembler podmienia podany symbol na podaną wartość liczbową.

4.3. Instrukcja 'load immediate value' (ldi)

Instrukcja *ldi* kopiuje 8-bitową wartość podaną jako drugi parametr instrukcji do wybranego rejestru ogólnego stosowania.

```
ldi Rd,K    ;16 ≤ d ≤ 31, 0 ≤ K ≤ 255
```

Symbol K reprezentuje 8-bitową wartość i może przyjmować wartość z zakresu 0 do 255. Rejestr może być wybrany z zakresu rejestrów r16 do r31 (jest to górny zakres rejestrów ogólnego zastosowania). Określenie *„immediate”* w nazwie określa tryb adresowania natychmiastowego i oznacza, że wartość, która ma być wpisany do rejestru musi być podany bezpośrednio jako argument w instrukcji.

4.4. Instrukcja 'relative jump' (rjmp)

Wykonanie instrukcji:

```
rjmp address
```

powoduje, że następna instrukcja wykonana przez CPU będzie pobrana spod adresu '*address*' w pamięci programu mikrokontrolera. W kodzie asemblera argument '*address*' jest odpowiednią etykietą („label”) umieszczoną w innej linii programu.

Na przykład:

```
stop:  rjmp stop
```

4.5. Reprezentacja danych

Istnieją cztery sposoby na to, aby w programie assembler przedstawić liczbę 8-bitową. Są to odpowiednio reprezentacje: heksadecymalna, binarna, dziesiętna i format ASCII.

Reprezentacja heksadecymalna

Liczbę heksadecymalną można przedstawić w assemblerze na dwa sposoby:

1. Umieścić 0x jako prefiks liczby heksadecymalnej. Na przykład: 0x99
2. Umieścić znak $ z przodu liczby. Na przykład: $99

Reprezentacja binarna

Umieścić 0b jako prefiks liczby binarnej. Na przykład: 0b10011001

Format ASCII

Aby jako argument podać kod ASCII znaku, należy użyć wybranego znaku i średników, na przykład: 'A'

Liczby dziesiętne

Liczby dziesiętne podajemy bezpośrednio, bez żadnych modyfikatorów. Można również używać liczb ze znakiem. Na przykład: 2 i −2. Liczby ze znakiem kodowane przez assembler są w systemie uzupełnień do dwóch U2.

Ćwiczenie 1.3

Używając edytora w środowisku Microchip Studio napisz program, który:

1. Definiuje różne, wybrane wartości liczbowe w formatach heksadecymalnym, binarnym, dziesiętnym i ASCII. Stałe proszę nazwać odpowiednio 'decimal', 'hex', 'binary', i 'ascii'
2. Ładuje stałą 'decimal' do rejestru *r16*, stałą 'hex' do rejestru *r17*, stałą binary do rejestru *r18,* a stałą 'ascii' do rejestru *r19*.
3. Zakończenie kodu to instrukcja skoku do samej siebie. Oznacz etykietą 'stop' linię z instrukcją 'rjmp'.

Dokonaj assemblacji (kompilacji) napisanego kodu.

4.6. Assemblacja kodu

Wybierz 'Build->Build all' z menu programu Microchip Studio. Jeżeli kompilacja przebiegła poprawnie w konsoli, na końcu logu wykonanych operacji powinna się pojawić informacja:

```
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

5. Symulator AVR

5.1. Debugowanie kodu programu

Zanim program zostanie uruchomiony w kontrolerze możliwe jest dokonanie jego weryfikacji przy pomocy symulatora programowego AVR Simulator, który jest wbudowany w Microchip Studio.

Uruchom symulator:

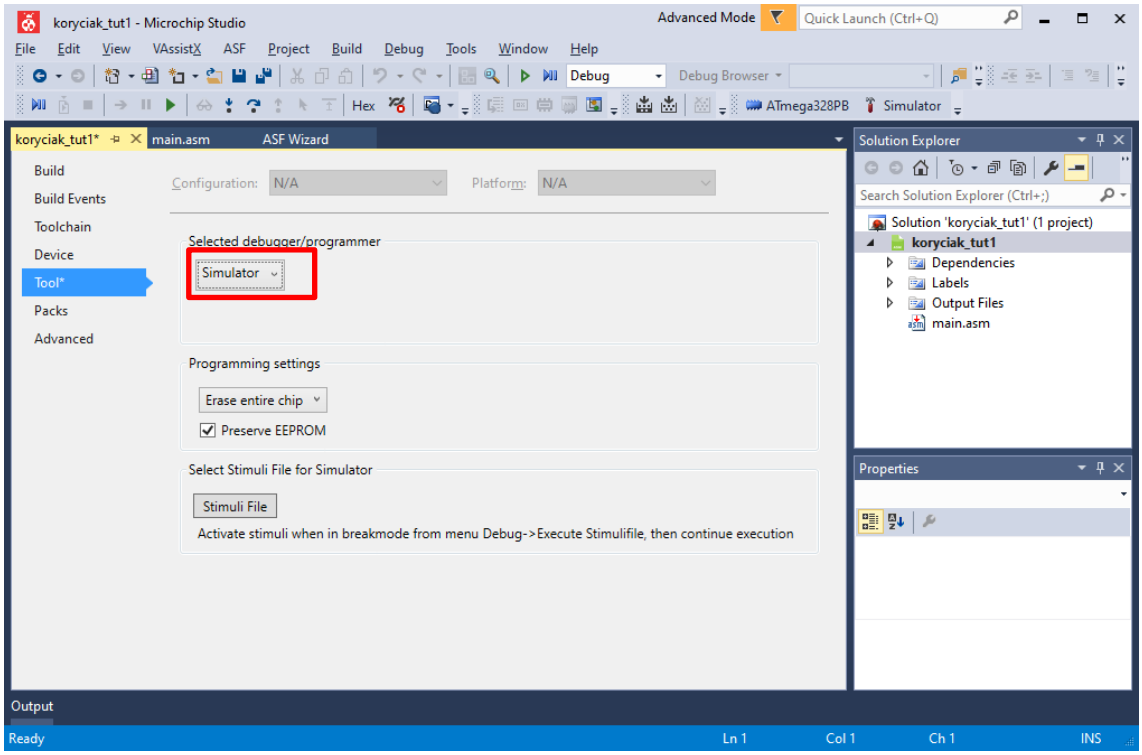
Wybierz *'Debug -> Start Debugging and Break'*

Pojawi się informacja (przy pierwszym uruchomieniu w nowym projekcie), że należy wybrać urządzenie, przy pomocy którego będziemy debugować kod.

Wybierz *'Selected debugger/programmer -> Simulator'*

Zatwierdź zmiany zapisując ustawienia (Ctrl+S lub *File -> Save Selected Items*)

Podpowiedź: Gwiazdka przy nazwie zakładki oznacza, że dany plik nie jest zapisany.



Powtórz próbę włączenia debugowania ('Debug -> Start Debugging and Break').

Widok w Microchip Studio zmieni się na widok typu “Debugger View”. Pojawiły się nowe okna: ”Watch” i „Memory”. Jednocześnie, pojawiła się nowa linijka podświetlająca linię kodu w oknie programu. Linijka ta wskazuje na miejsce, gdzie zatrzymał się symulator programu. W naszym przypadku jest to początek programu. Teraz można „krokować” program instrukcja po instrukcji, używając komendy Step symulatora.

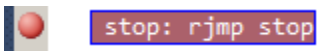
a) Podstawowe komendy symulatora (Menu 'Debug')

Start debugging and break (Alt-F5)		Uruchamia symulator i zatrzymuje się na pierwszej instrukcji programu.
Stop debugging (Ctrl-Shift-F5)		Zatrzymuje symulację
Continue (F5)		Wykonuje symulację do następnego punktu przerwy („break point”)
Step into (F11)		Przejdź do następnej linii instrukcji.

b) Punkty przerwy („Breakpoints”)

Ustawienie puntu przerwy w kodzie programu powoduje, że podczas debugowania lub symulacji, program zatrzyma się w danej linii – działanie procesora zostanie przerwane i kontrola nad konsolą zostanie przekazana programiście.

Aby ustawić punkt przerwy w wybranej linii kodu należy kliknąć margines okna edytora obok wybranej linii. Kropka obok linii z instrukcją oznacza, że w danej linii ustawiono punkt przerwy. Powtórne kliknięcie usuwa punkt przerwy.



Uruchomiona symulacja (F5) zatrzyma się po osiągnięciu instrukcji oznaczonego punktem przerwy.

Ćwiczenie 1.4

1. Dodaj 'hex', 'decimal', 'binary', 'ascii', 'r16', 'r17', 'r18', 'r19' i 'stop' do okna 'Watch'. Wyjaśnij wartości, które wyświetliły się obok dodanych zmiennych.
2. W oknie „Memory window” zmień obszar pamięci („Memory region”) na „Data REGISTERS”.
3. Ustaw punkt przerwy w wybranej, dowolnej instrukcji programu.
4. Uruchom symulator (F5).
5. Doprowadź symulację do końcu programu.
6. Porównaj wartości rejestrów w oknie „Memory window” z wartościami zdefiniowanymi w kodzie programu.

6. Programowanie portów we/wy

Mikrokontrolery megaAVR wykorzystywane na laboratorium oferują projektantowi cztery porty we/wy. Porty te są oznaczona jako: port B, port C, port D i port E.

CPU mikrokontrolera kontroluje pracę portów przy pomocy zestawu rejestrów. Każdy port ma swój niezależny taki zestaw. Każdy bit rejestru kontrolnego jest przypisany do jednej końcówki określonego portu we/wy (na przykład bit 0 kontroluje pierwszy pin portu).

Wyróżniamy dwa rejestry kontrolne dla każdego portu: „output value register" (*portX*), „direction register" (*ddrX).*

Poniższa tablica pokazuje rejestry i ich adresy dla portu B.

Nazwa rejestru	Adres	Funkcja
portb	$25	output
ddrb	$24	direction

6.1. Funkcja rejestru ddrX

Każda końcówka portu we/wy może pełnić rolę cyfrowego wejścia albo wyjścia. Stan bitów w rejestrze *ddrX* decyduje o tym czy odpowiadające bitom końcówki będą pełniły rolę wejścia czy wyjścia. Aby wszystkie końcówki portu B pełniły rolę wyjścia, należy do rejestru *ddrb* wpisać wartość 0b11111111 (0xff, 255). Aby port B był wejściem należy do *ddrb* wpisać same zera. Po włączeniu zasilania mikrokontrolera (i po sygnale reset) wszystkie porty mają wartości zero w rejestrach *ddr*.

Podpowiedź

Łatwo można zapamiętać, że rejestr ddrx musi zawierać jedynki, aby port był wyjściem, jeżeli założymy, że funkcja wyjścia polega na dawaniu czegoś co się posiada. Tak więc musimy coś mieć (jedynki) w ddrx, aby port mógł coś dać – być wyjściem.

6.2. Instrukcja 'Store to SRAM' (sts)

Instrukcja *sts* powoduje, że CPU zapisuje zawartość rejestru ogólnego zastosowania (r0-r31) pod podany adres w pamięci danych. Adres może wskazywać na adres, pod którym znajduje się rejestr we/wy, rejestr ogólnego stosowania lub komórka pamięci SRAM.

```
Sts K, Rr   ; store register into location k
            ; k is an address between $0000 to $FFFF
```

Przykład:

```
ldi r16, 0x55 ; r16=55 (in hex)
sts 0x25, r16 ; copy r16 to port B
```

6.3. Instrukcja 'Output' (out)

Instrukcja *out* powoduje zapisanie zawartości rejestru ogólnego zastosowania (r0-r31) w rejestrze we/wy.

```
out P, Rr   ;store register to I/O location (r=0..31, P=0..63)
```

Ważne

Należy zwrócić uwagę, że instrukcja out adresuje tylko rejestry we/wy, a nie całą przestrzeń SRAM. Z związku z tym adres tego samego rejestru we/wy będzie inny w instrukcji out, a inny w instrukcji sts. Konkretnie adres dla sts będzie większy o wartość 32 (0x20). Na przykład 'out 0x16, r20' kopiuje zawartość rejestru r20 pod adres 0x16 w przestrzeni adresowej we/wy, ale w rzeczywistości jest to adres $36 w przestrzeni SRAM !!!

6.4. Out vs. sts

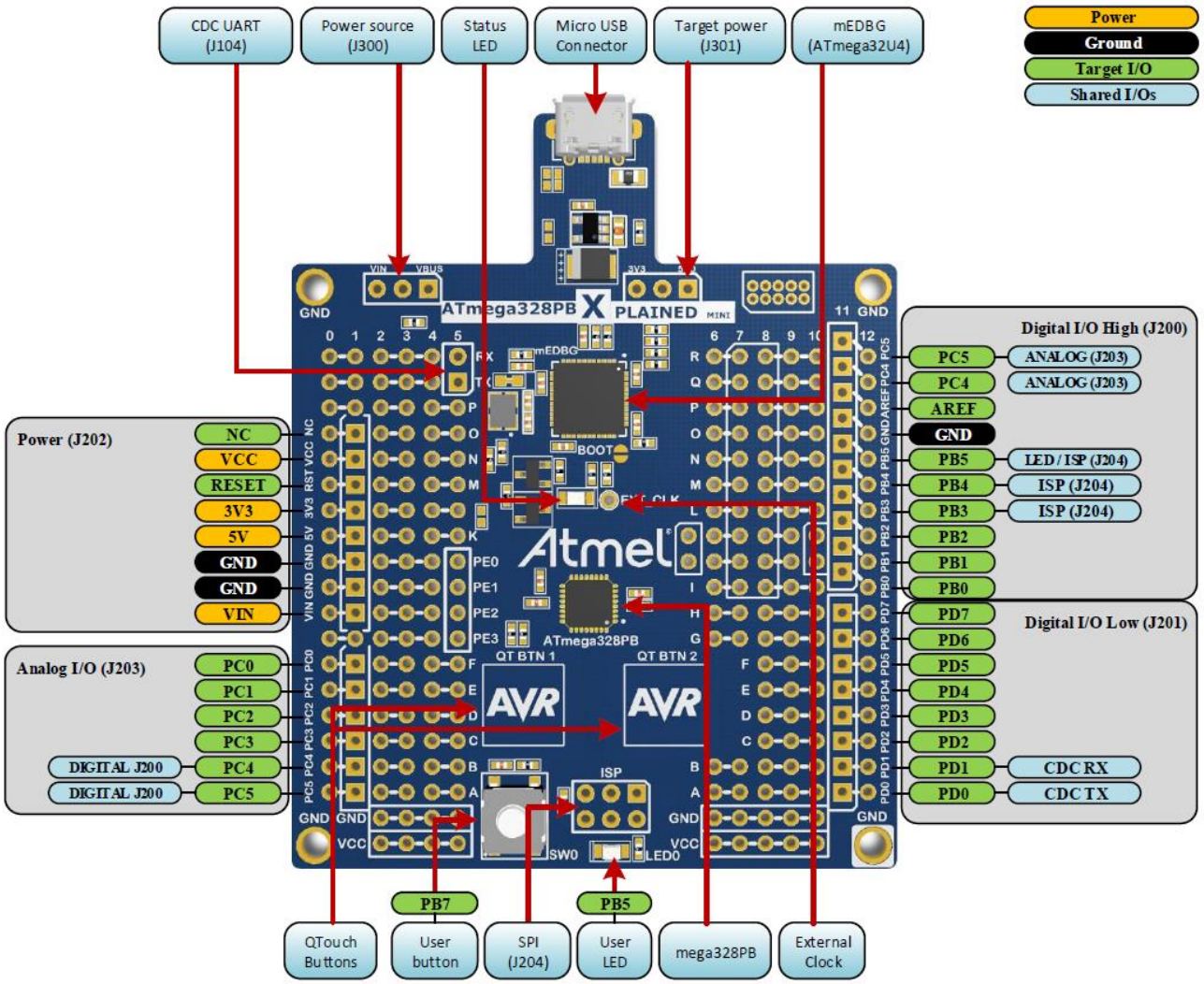
Jak już wspomniano, instrukcja *sts* kopiuje zawartość rejestru do dowolnej lokacji w SRAM w tym do lokacji zajmowanych przez rejestru we/wy. Oznacza to, że *sts* może wykonywać tą samą operację co *out.* Dlaczego więc mikrokontrolery AVR w liście instrukcji zawierają dodatkowo instrukcję *out*? Instrukcja *out* ma następując zalety:

1. CPU wykonuje instrukcje *out* szybciej niż *sts. Sts* jest wykonywane przez dwa takty zegara, a *out* przez jeden takt zegara.
2. Instrukcja *out* zajmuje w pamięci programu dwa bajty, a instrukcja *sts* cztery bajty.
3. Używając instrukcji *out*, programista może używać predefiniowanych nazw rejestrów zamiast ich adresów.
4. Instrukcja *out* jest dostępna na wszystkich mikrokontrolerach AVR, a *sts* nie występuje w niektórych układach AVR.

7. Zestaw uruchomieniowy ATmega328PB Xplained mini

7.1. Wprowadzenie

Wygląd platformy ewaluacyjnej wraz z opisem portów jest przedstawiony na rysunku poniżej.



Na płytce drukowanej znajdziemy dwa mikrokontrolery. ATmega328PB to mikrokontroler docelowy, którego działanie będziemy obserwować. Natomiast mikrokontroler ATmega32U4 spełnia zadania sprzętowego debuggera/programatora typu mEDBG, który może działać jako: programator, debugger i wirtualny port COM.

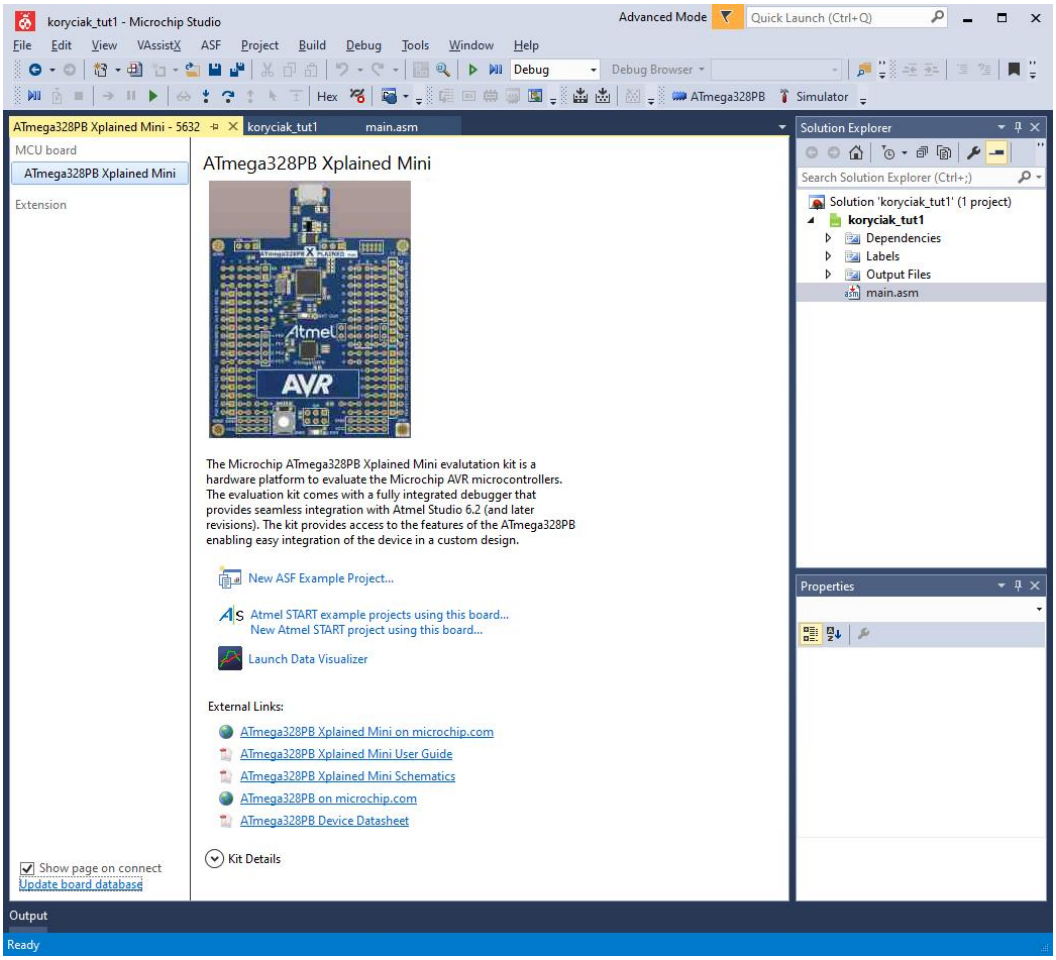
Na płytce każdy punkt lutowniczy jest oznaczony literą wiersza i numerem kolumny, jak na szachownicy (np. punkt R6).

Ponadto na płytce znajduje się fizyczny przycisk (PB7), dioda LED (PB5), dwa pola służące jako przyciski pojemnościowe. Mikrokontroler ATmega328PB korzysta z źródła zegarowego generowanego przez mEDBG o częstotliwości 16MHz.

Proszę zwrócić uwagę na przylutowane do płytki złącza. Prócz standardowych wyprowadzeń dla formatu zgodnego z Arduino (gniazda J200, J201, J202 i J203) przylutowane zostały złącza do portu E (kolumna 5, rzędy G, H, I i J), oraz do rzędu masy (GND) i rzędu zasilania (VCC).

7.2. Programowanie

Płytka ATmega328PB Xplained mini po podłączeniu do komputera poprzez port USB i uruchomieniu programu Microchip Studio sygnalizuje swoją obecność przez otworzenie zakładki jak na obrazku poniżej.

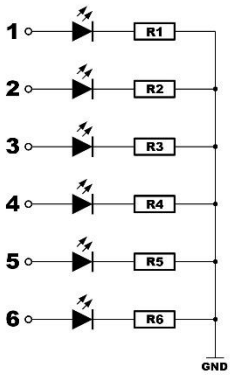


Wbudowany programator mEDBG pozwala na programowanie procesora docelowego na dwa sposoby: przez protokół ISP lub debugWIRE. Na debugowanie procesora ATmega328PB pozwala tylko interfejs debugWIRE, dlatego będzie głównie on wykorzystywany do pracy podczas laboratorium.

7.3. Diody LED

Diody LED stanowią najprostsze rozwiązanie pozwalające na obrazowanie stanu portów we/wy mikrokontrolera. Wysoki poziom bitu na porcie jest sygnalizowany przez świecącą diodę LED.

Sześć diod LED znajdziemy na dołączonym do zestawu module. Anody diod LED oraz masa (GND) są wyprowadzone na złącze.



8. Przygotowanie zestawu do ćwiczeń

8.1. Podłączenie ATmega328PB Xplained mini do PC

1. Podłącz płytkę przy pomocy dołączonego kabla USB do komputera PC. Jeżeli płytka jest podłączana po raz pierwszy, to rozpocznie się proces instalacji urządzenia w Windows. Będziesz potrzebował uprawnień administratora, aby poprawnie zakończyć instalację. W razie problemów proszę skonsultować się z prowadzącym zajęcia.

Uwaga

Proszę zwrócić uwagę na kształt płytki drukowanej. Przy podłączaniu i odłączaniu kabla USB proszę zachować szczególną ostrożność, żeby nie uszkodzić portu i płytki.

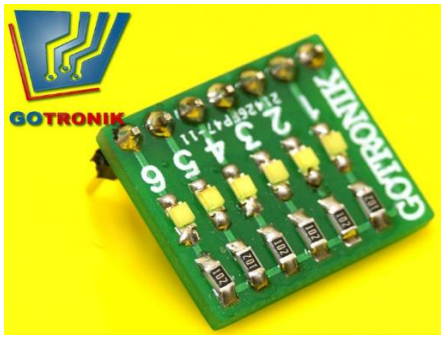
2. W programie Microchip Studio zawsze powinna się otworzyć zakładka „ATmega328PB Xplained Mini - XXXX" gdzie XXXX to ostatnie cyfry numeru seryjnego układu (jak na obrazku w punkcie 7.2).

3. Jeżeli urządzenie nie działa poprawnie zacznij do sprawdzenia w *menadżerze urządzeń* Windows czy programator jest zainstalowany poprawnie.

8.2. Sposób realizacji połączeń z diodami LED

Widok płytki zaprezentowano w punkcie 7.1. Przy pomocy dołączonych kabelków proszę wykonać następujące połączenia:

1. Proszę połączyć sześć pinów portu B (B0-B6, gniazdo J200) do sześciu wyprowadzeń diod LED (oznaczenia 1 – 6 na rysunku poniżej).
2. Proszę podłączyć masę z płytki (np. GND, gniazdo J200) do modułu z diodami LED (wolny pin na module z diodami LED, oznaczony nazwą „GOTRONIK").



9. Sterowanie diodami LED

Ćwiczenie 1.5

1. Proszę zapisać poprzednio stworzony program pod nazwą '*first_AVR_asm.asm*'

Uwaga

Dobrze jest zapisywać już napisane i więcej nieużywane programy pod nazwami innymi niż ich oryginalna nazwa. W procesie assemblacji Microchip Studio kompiluje tylko pliki, które są dołączone do projektu. Dołączone pliki można kontrolować w okienku „Solutions Explorer" Dla swojej własnej wygody przy pisaniu nowego programu zawsze używaj pliku o nazwie main.asm. Inne nieużywane już programy archiwizuj pod inną nazwą. Nie kasuj starych programów, bo na pewno jeszcze Ci się przydadzą.

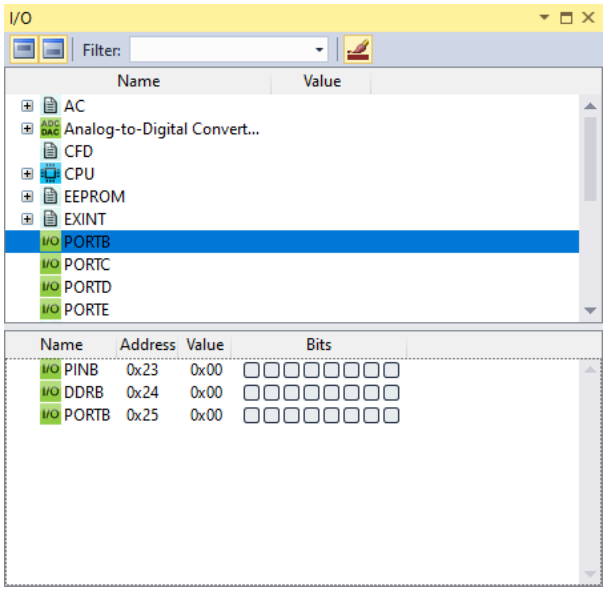
2. W pliku „main.asm", napisz program, który zapali parzyste diody LED.

Podpowiedź

- Stan logiczny jeden powoduje zapalenie się diody.
- W programie zapisz do rejestru *ddrb* wartość, która ustawi port B jako wyjście.
- Zapisz odpowiednią wartość do rejestru *portb.*
- Zakończ program instrukcją skoku do samego siebie.

Nazwa rejestru	Adres	Funkcja
portb	$25	output
ddrb	$24	direction

3. Proszę przekrokować program używając AVR Simulator. W symulatorze można zweryfikować stan diod LED sprawdzając wizualizację stanu rejestrów portu B w oknie „IO View." W razie jego braku włącza się je przez opcje: *Debug -> Windows -> I/O*



4. Jeżeli program działa poprawnie, to proszę zapisać plik jako '*LEDs_even.asm*'. (File->Save as)

9.1. Programowanie mikrokontrolera

Teraz, przystąpimy do programowania pamięci FLASH mikrokontrolera kodem binarnym który powstał w wyniku assemblacji programu z ćwiczenia 1.5

1. Proszę uruchomić narzędzie do programowania mikrokontrolerów AVR:

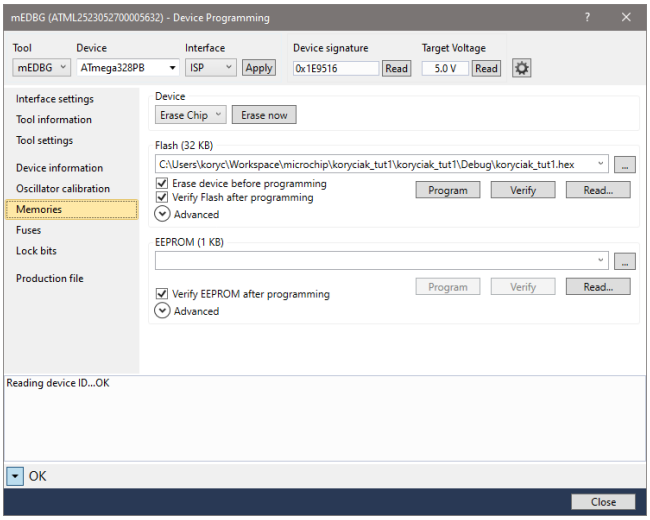
Wybierz: 'Tools->Device programming'. Pojawi się okno 'Device Programming'.

2. Wybierz typ programatora („Programming tool”), typ mikrokontrolera i typ interfejsu programującego.

'Tool' → ***'mEDBG'*** *'Device'* → ***'ATmega328PB'*** *'Interface'* → ***ISP***

Kliknij przycisk 'Apply', a następnie przycisk 'Read' w polu „Device signature”.

W dolnym pasku okna powinien się pojawić napis: ”Reading device ID … OK.”



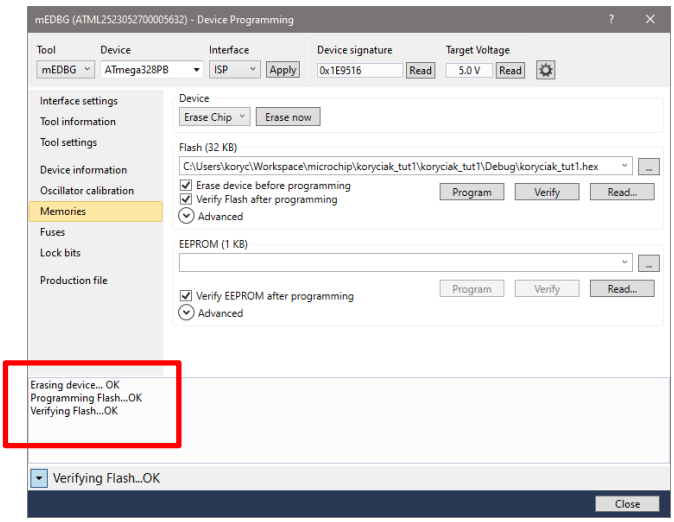
Uwaga

Jeżeli pojawi się informacja o błędzie i sygnatura urządzenia nie zostanie odczytana przejdź do punktu 9.3 poniżej. Punkt 9.3.5 opisuje jak aktywować ISP (dezaktywować debugWIRE).

3. Wybierz „Memories” z bocznego menu okna „Device Programming”.

4. W polu „Flash” wybierz plik 'hex' znajdujący się w folderze projektowym (na przykład *shrek_upt_lab.hex*)

5. Aby zaprogramować mikrokontroler kliknij przycisk 'Program'. Etap zostanie potwierdzony.



9.2. Weryfikacja programu w mikrokontrolerze

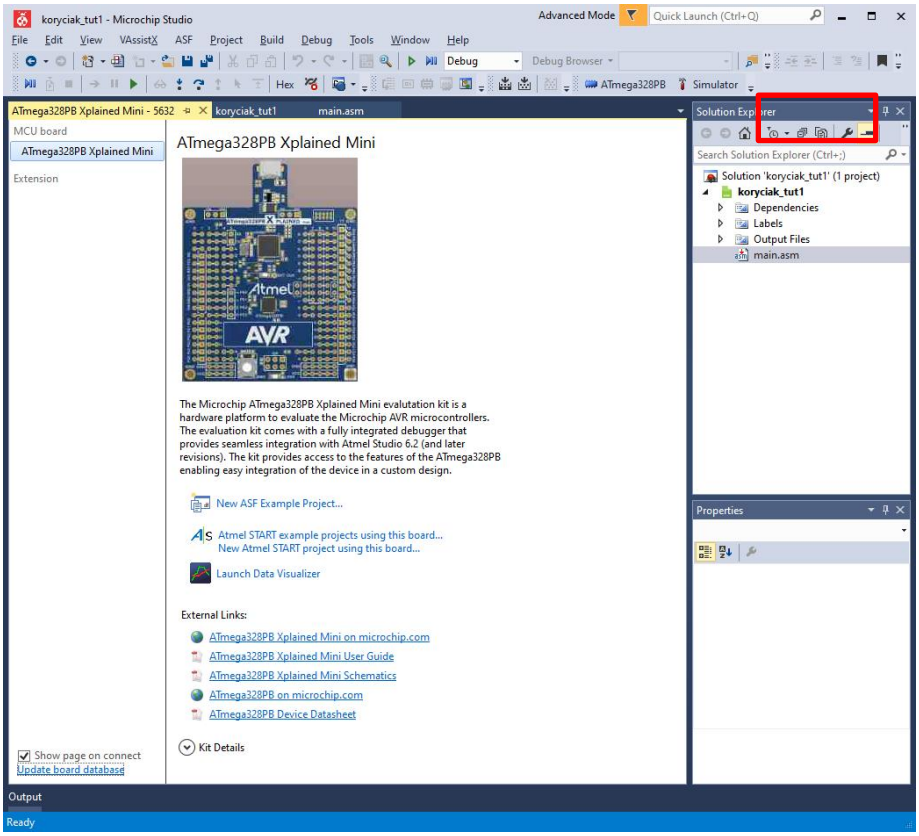
ATmega jest zaprogramowana i powinna sterować diodami zgodnie z programem napisanym w ćwiczeniu 1.5.

Proszę sprawdzić, czy na module LED świecą się diody o parzystych numerach.

9.3. Debugowanie na mikrokontrolerze

Środowisko Microchip Studio pozwala na wgranie do pamięci mikrokontrolera naszego programu, a następnie jego przekrokowanie. Obsługa tej funkcji wygląda dokładnie tak samo jak w punkcie 5 tej instrukcji, z tą różnicą, że zamiast symulatora wybieramy odpowiedni debugger.

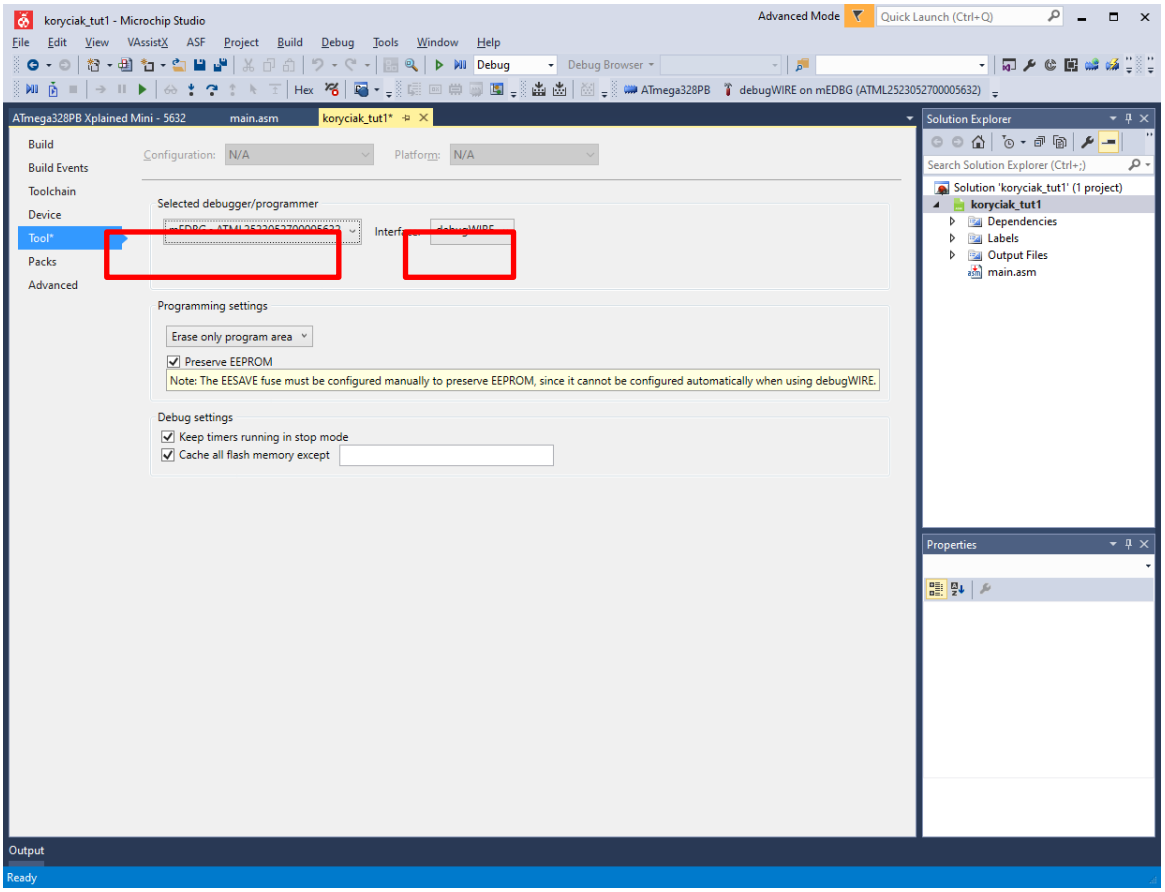
1. Naciśnij na ikonę ‘Simulator’ w celu jego zmiany (zaznaczony poniżej).



2. W zakładce, która się otworzyła (aktywny w lewym panelu ‘Tool’) zmień:

Selected debugger/programmer -> ‘mEDBG’

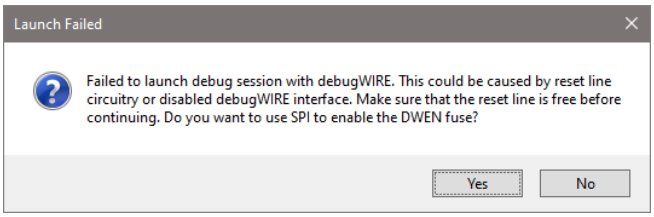
Interface -> ‘debugWIRE’



Zatwierdź zmiany zapisując ustawienia (Ctrl+S lub *File -> Save Selected Items*)

3. Włącz debugowanie (*'Debug -> Start Debugging and Break'*)

Jeżeli wcześniej ta funkcja nie była używana, jednorazowo pojawi się informacja jak poniżej. Należy zatwierdzić opcję przez przycisk ‘Yes’



4. Wykonaj krokowanie programu, tym razem z wykorzystaniem mikrokontrolera.

5. Aby zakończyć proces debugowania z możliwością późniejszego korzystania z interfejsu ISP (tylko do programowania) należy wybrać: *‘Debug -> Disable debugWIRE and Close’*

10. Sterowanie pojedynczymi bitami rejestrów.

Zestaw instrukcji mikrokontrolerów megaAVR zawiera instrukcje pozwalające na sterowanie pojedynczych bitów 8-bitowych rejestrów. Instrukcje tego typu są przydatne, kiedy istnieje konieczność zmiany ustawienia jedynie pojedynczego bitu w rejestrze, a stan pozostałych bitów nie jest znany.

10.1. Instrukcja 'set Bit in I/O Register' (sbi)

Instrukcja *sbi* ustawia wskazany bit w rejestrze we/wy. Instrukcja interpretuje adres jako adres w przestrzeni we/wy. Zakres adresowania to 0 do 31.

```
sbi   A,    b ; Set bit b in A register  0 <=A<= 31, 0 <=b<= 7
```

Przykład:

```
sbi $0B, 7 ; Set bit 7 in Port D
```

10.2. Instrukcja 'clear Bit in I/O Register' (cbi)

Instrukcja c*bi* kasuje wskazany bit w rejestrze we/wy. Instrukcja interpretuje adres jako adres w przestrzeni we/wy. Zakres adresowania to 0 do 31.

```
cbi   A,    b ; Clear bit b in A register      0 <=A<=31, 0 <=b<= 7
```

Przykład:

```
cbi $0B, 7 ; Clear bit 7 in Port D
```

Ćwiczenia 1.6

1. *Napisz program, który zaświeci parzyste diody LED na podłączonym module. Użyj instrukcji sbi i cbi.*
2. *Przekrokuj program obserwując stan rejestrów portu B w symulatorze.*
3. *Zaprogramuj mikrokontroler i sprawdź poprawność działania programu.*
4. *Zapisz program jako 'LEDs_even_set.asm'.*

Uwaga

Plik heksadecymalny używany do programowania mikrokontrolera ma taką nazwę jak nazwa twojego projektu.