Akademia
Górniczo-Hutnicza
w Krakowie
Katedra Elektroniki

Laboratorium mikrokontrolerów

Ćwiczenie 1

Wprowadzenie

Autor: Paweł Russek
Tłumaczenie: Paweł Russek
http://www.fpga.agh.edu.pl/tm
wer. 4.03.15

1. Wprowadzenie

1.1. Cel ćwiczenia

Głównym celem ćwiczenia jest zapoznanie studenta z elementami składowymi stanowiska laboratoryjnego na "Laboratorium mikrokontrolerów".

Części, które tworzą laboratorium stanowią środowisko programistyczne dla mikrokontrolera ATMega32 firmy Atmel. Pojedyncze stanowisko składa się z:

- Komputera klasy PC z systemem operacyjnym Microsoft Windows,
- zestawu uruchomieniowego ZL3AVR,
- programatora AVR Dragon firmy Atmel,
- oprogramowanie Atmel Studio 6.0.

Na początku instrukcji przedstawiono podstawowe informacje na temat rodziny mikrokontrolerów megaAVR firmy Atmel. Następnie przedstawiono oprogramowanie Atmel Studio w wersji 6.0 oraz płytkę uruchomieniową ZL3AVR.

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

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

1.2. Konieczne wiadomości wstępne

- Postawy elektroniki cyfrowej.
- Znajomość zapisu i konwersji liczb w systemie binarnym i heksadecymalnym.

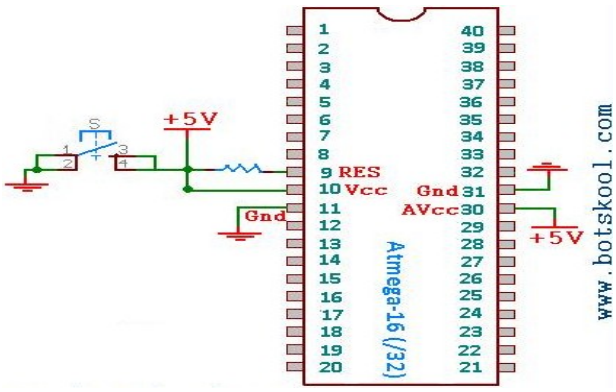
2. Architektura mikrokontrolerów megaAVR

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

- **Data sheet: ATMega16/ATMega32/ATMega64** (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 oraz elementy peryferyjne. W celu uruchomienia, mikrokontrolery AVR wymagają minimalnej

liczby komponentów zewnętrznych. Poniżej na rysunku przedstawiono minimalną konfigurację układu AVR ATMega, która umożliwia jego poprawną pracę w układzie elektronicznym. Układ z rysunku poniżej wykorzystuje wewnętrzny, wbudowany oscylator zegarowy, ale układy ATMega mogą korzystać również z zegara/oscylatora zewnętrznego.



W zasadzie AVR potrzebuje tylko źródła zasilania i obwodu dla sygnału „Reset" w celu rozpoczęcia pracy. Typowo na obwód „Reset" składają się rezystor podciągający do zasilania („pull-up resistor") oraz przełącznik („reset button"). W układach AVR sygnał reset jest aktywny niskim poziomem napięcia.

Pozostałe widoczne na rysunku końcówki układu („pins") realizują funkcje równoległego we/wy („parallel ports"):

- Port A (bity A0 do A7): końcówki 40 w dół do 33 („40 downto 33").
- Port B (bity B0 do B7): końcówki 1 do 8.
- Port C (bity C0 do C7): końcówki 22 do 29.
- Port D (bity D0 do D7): końcówki 14 do 21.

W podstawowym trybie pracy porty we/wy są zapisywane i odczytywane jako rejestry 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.

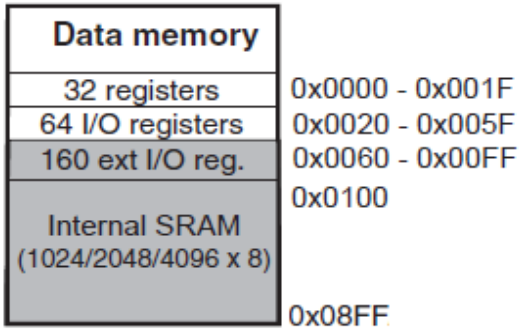
2.1. Architektura mikrokontrolerów megaAVR

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. Rozmiar pamięci programu jest zmienna w zależności od typu układu AVR i waha się od 1kB do 256kB. Wykorzystywany na laboratorium układ może zmieścić 32kB (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 kilo 16 bitowych słów.

Pamięć danych mikrokontrolera AVR jest pamięcią o dostępie typu RAM. Mapa tej pamięci jest przestawiona na rysunku poniżej.



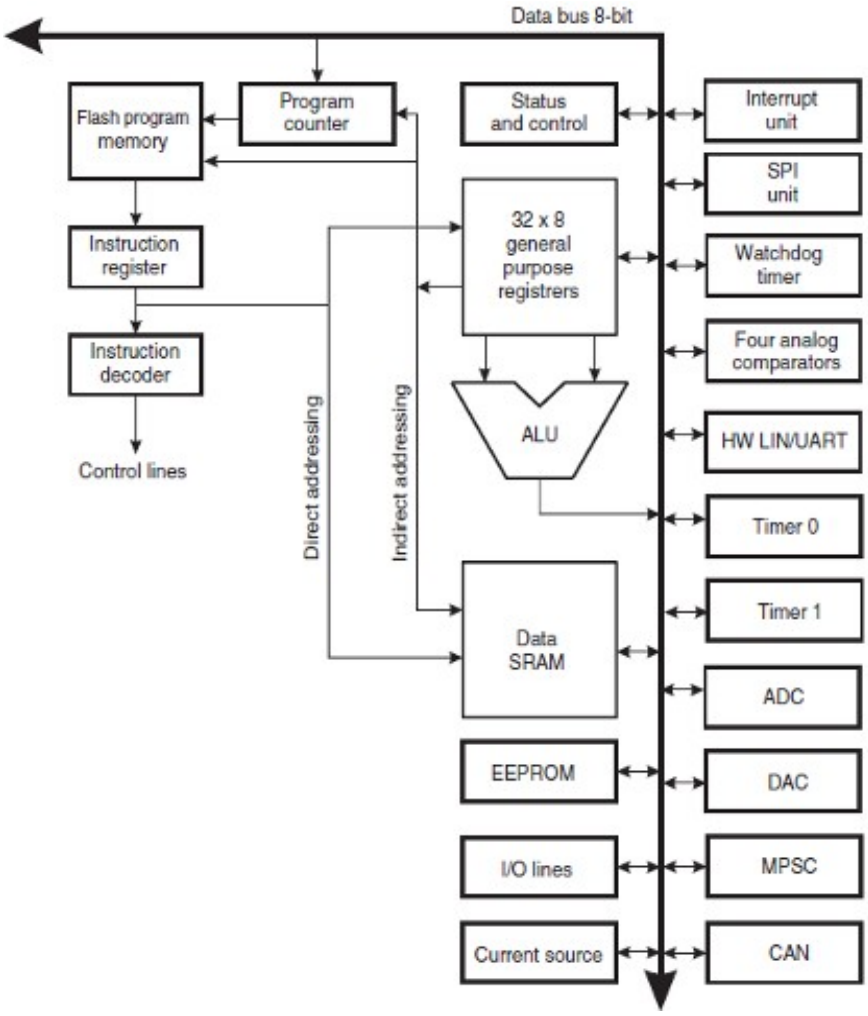
Pamięć danych jest to pamięć 8 bitowa. Maksymalnie AVR może zaadresować 64kB przestrzeni adresowej. ATMega32 oferuje jednak tylko 4KB pamięci danych RAM.

We wszystkich mikrokontrolerach megaAVR, pierwszy 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 we/wy oraz pamięć SRAM. Rozmiar przestrzeni we/wy („I/O registers") 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 SRAM jest przeznaczony do przechowywania danych podczas wykonywania przez CPU programu.

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

Rysunek poniżej przedstawia schemat blokowy architektury mikrokontrolera AVR.



Ćwiczenie 1.1

Otwórz dokument PDF pt.:"ATMega16/ ATMega32/ ATMega64 datasheet".

1. Wskaż i nazwij przynajmniej trzy charakterystyczne cechy układów AVR wymienione w dokumencie.
2. Podaj rozmiar pamięci SRAM dla układów ATMega16, ATMega32 i ATMega64
3. Odszukaj końcówki XTAL1 i XTAL2. Jaka jest funkcja tych końcówek.

3. Oprogramowanie Atmel Studio

Atmel Studio to zintegrowane środowisko programistyczne do programowania i debugowania mikrokontrolerów firmy Atmel.

Ćwiczenia 1.2a

Sprawdź czy oprogramowanie Atmel Studio 6.0 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.

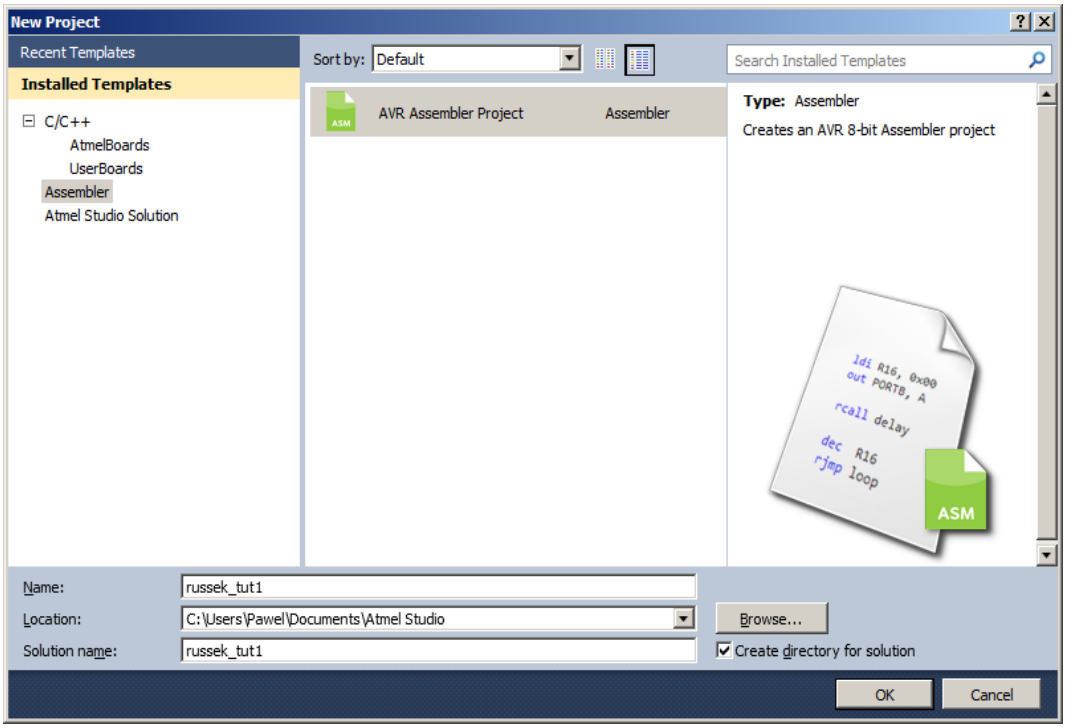
Ściągnij oprogramowanie Atmel Studio ze strony wskazanej przez prowadzącego. Możesz także skorzystać z wersji instalacyjnej która znajduje się na stronie firmy Atmel: www.atmel.com. Uruchom pakiet instalacyjny i wykonuj kolejne polecenia programu.

- Wybierz instalację pełnej wersji programu.
- Skorzystaj z domyślnej proponowanej ścieżki programu.
- Zainstaluj wszystkie proponowane komponenty pakietu.

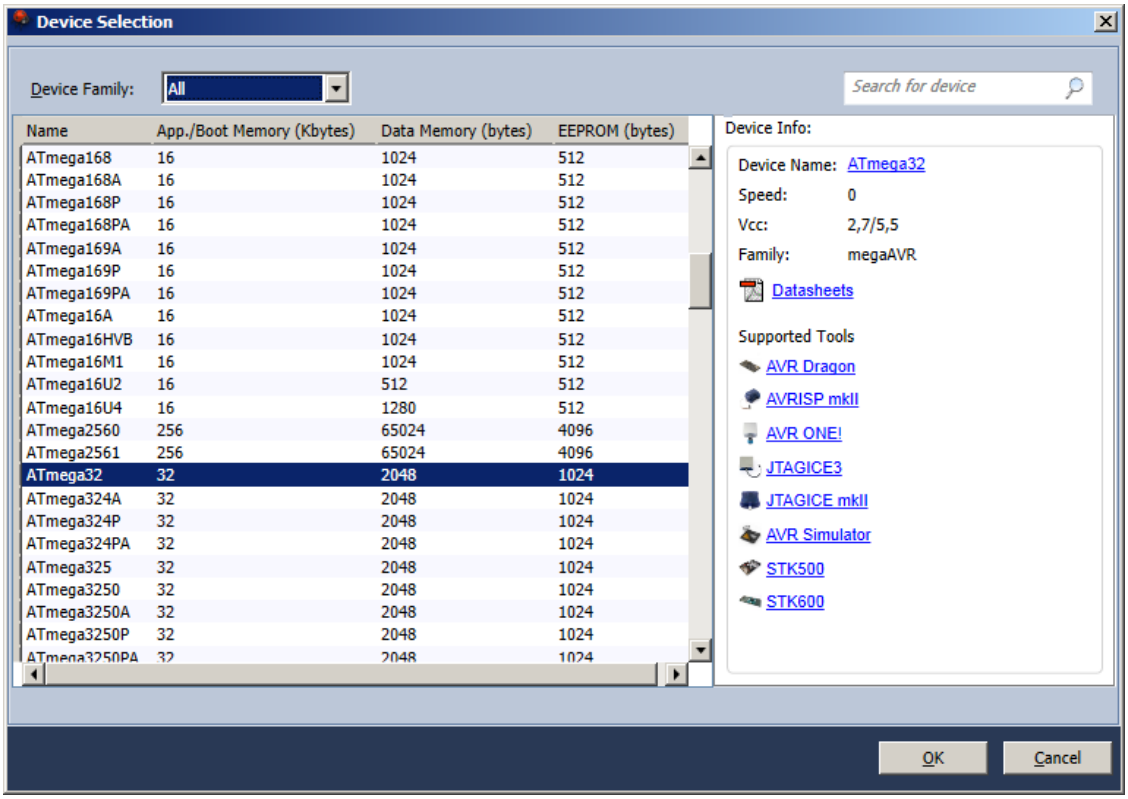
Ćwiczenie 1.2b

Uruchom Atmel Studio 6.0 i utwórz nowy projekt typu „Assembler project"

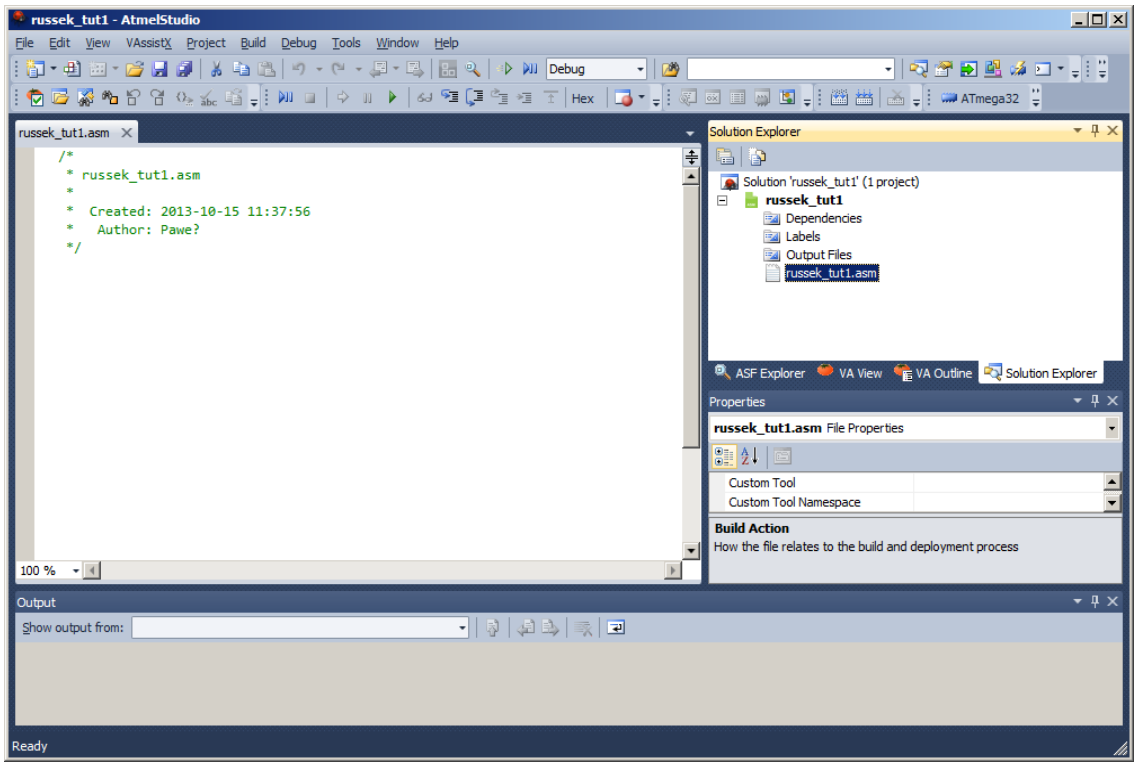
1. Kliknij "Atmel Studio 6.0" w menu start systemu Windows.
2. Wybierz "New project" w oknie "Start Page – Atmel Studio".
3. Wybierz "Assembler" i "AVR assembler project"
4. Zmień nazwę swojego projektu w polu "Name". Z komputera którego używasz korzysta wielu studentów, dlatego najlepiej jeżeli w nazwie projektu uwzględnisz swoje nazwisko: na przykład '*shrek_tm_lab_czw_g900*`. Możesz także zmienić domyślny folder dyskowy w którym zostanie utworzony projekt (pole "Location").



5. Kliknij 'OK'
6. Wybierz mikrokontroler ATMega32



7. Kliknij 'OK'
8. Właśnie utworzyłeś nowy projekt. Atmel Studio otwarł widok podstawowego okna roboczego.



4. Podstawy AVR assembler

Program assembler (kompilator assemblera) tłumaczy napisany przez programistę kod assemblera na kod obiektowy. Kod obiektowy może być wykorzystywany na przykład jako plik wejściowy do programu symulatora albo emulatora (na przykład AVR In-Circuit Emulator). Dodatkowo assembler tworzy plik, który może być bezpośrednio (lub pośrednio po dodatkowej konwersji) wykorzystany do zaprogramowania pamięci FLASH mikrokontrolera.

4.1. Kod w języku assembler

Program assembler jako swoje wejście wykorzystuje plik z kodem który zawiera instrukcje assemblera zapisane w postaci mnemonicznej. Dodatkowo kod assemblera może zawierać etykiety („label") i dyrektywy assemblera („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:

```
label:  .EQU count=100 ; Set COUNT to 100 (Directive)


test:   ldi r21, count ; Load immediate value 'count' to register r21 (Instruction)
        jmp test ; Infinite loop (Instruction)
; Pure comment line
; Another comment line
```

Instrukcje są wykonywane przez CPU, natomiast dyrektywy są interpretowane jedynie przez assembler.

4.2. Dyrektywa EQU

Drektywa 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:

r*jmp 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 modyfikacji. Można również używać liczb ze znakiem. Na przykład: 2 i -2. Liczby ze znakiem kodowane są w systemie U2.

Ćwiczenie 1.3

Używając edytora w środowisku Atmel 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 Atmel 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

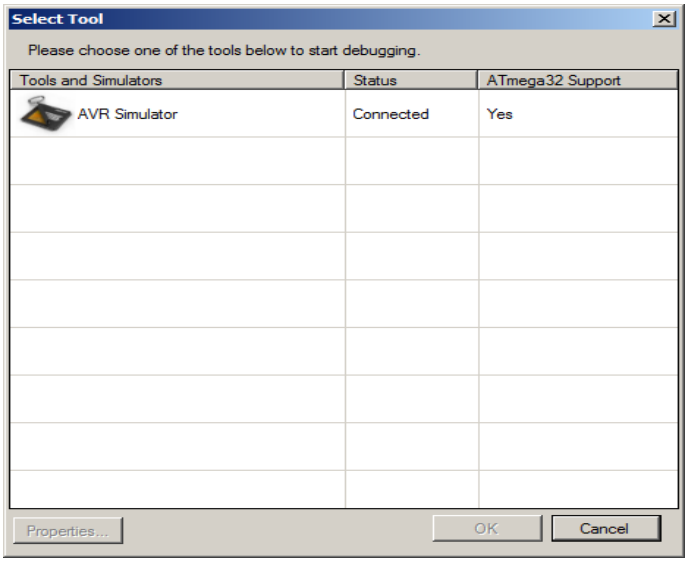
5.1. Debuggowanie kodu programu

Zanim program zostanie uruchomiony w kontrolerze zalecane jest dokonanie jego weryfikacji przy pomocy symulatora programowego AVR Simulator.

Uruchom symulator:

Wybierz *'Debug->Start debug and break'*

Pojawi się okno



Wybierz *'AVR Simulator'* i 'kliknij *'OK'*

Widok w Atmel 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

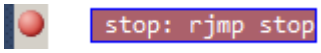
naszym przypadku jest to początek programu. Teraz można „krokować" program używając komendy *step* symulatora.

a) Podstawowe komendy symulatora (Menu 'Debug')

Start debugging and break (Alt-F5)		Uruchamia symulator i zatrzymuje się na pierwszej instrukcji programu.
Stop debugging (Cntr-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")

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, które mają obudowy z 40 wyprowadzeniami oferują projektantowi cztery porty we/wy. Porty te są oznaczona jako: port A, port B, port C i port D.

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 A.

Nazwa rejestru	Adres	Funkcja
porta	$3b	output
ddra	$3a	direction

6.1. Funkcje 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 A pełniły rolę wyjścia, należy do rejestru *ddra* wpisać wartość 0b11111111 (0xff, 255). Aby port A był wejściem należy do *ddra* 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 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 ;r10=55 (in hex)
sts 0x3b, r10 copy r10 to port A
```

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. 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. 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.

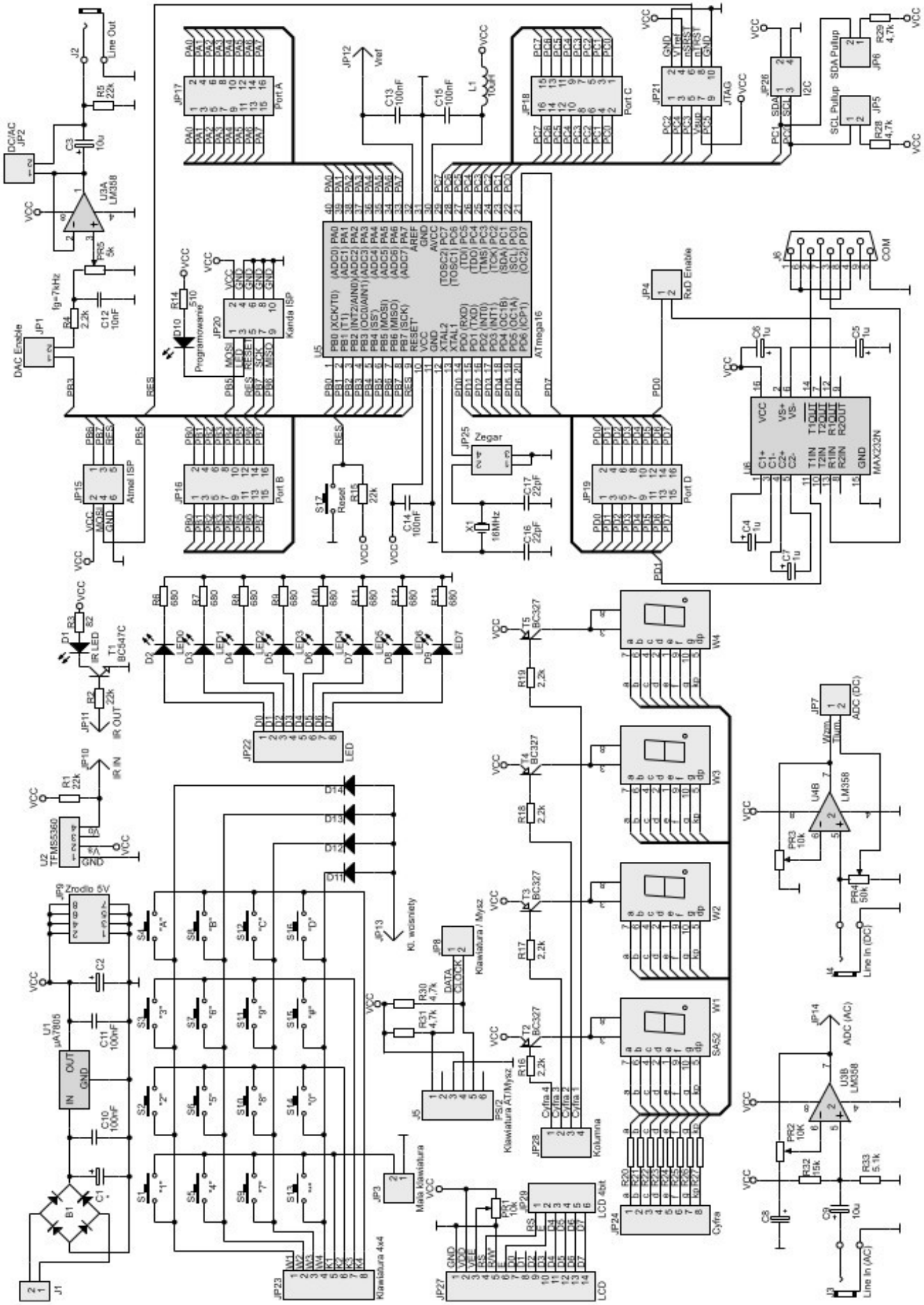
Tablica poniżej zawiera rejestry, ich adresy w pamięci SRAM i adresy we/wy.

Nazwa rejestru	Adres w przestrzeni SRAM (instrukcja *sts*)	Adres we/wy (instrukcja *out*)	Funkcja
porta	$3b	$1b	Wyjście portu
ddra	$3a	$1a	Kierunek

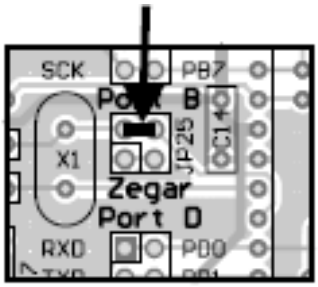
7. Zestaw uruchomieniowy ZL2AVR

7.1. Wprowadzenie

Schemat ZL3AVR jest przedstawiony na rysunku poniżej.



Mikrokontroler ATMega32 w zestawie ZL3AVR korzysta z wewnętrznego źródła zegarowego o częstotliwości 1MHz. Istnieje jednak możliwość podłączenia zewnętrznego źródła zegara lub kwarcu, ale to wymaga zmiany ustawienia zworki JP25 jak na rysunku poniżej.



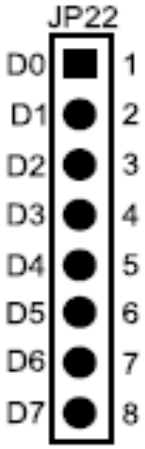
Ustawienie zworki JP25 dla wewnętrznego źródła zegarowego

7.2. Programowanie

Zestaw ZL3AVR oferuje trzy rożne porty, które mogą służyć do zaprogramowania mikrokontrolera. Są to: Atmel ISP, KANDA ISP i JTAG. Każde złącze wymaga użycia innego typu programatora. W tym ćwiczeniu będzie wykorzystywane złącze JTAG.

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. Anody diod LED są wyprowadzone na złącze JP22.

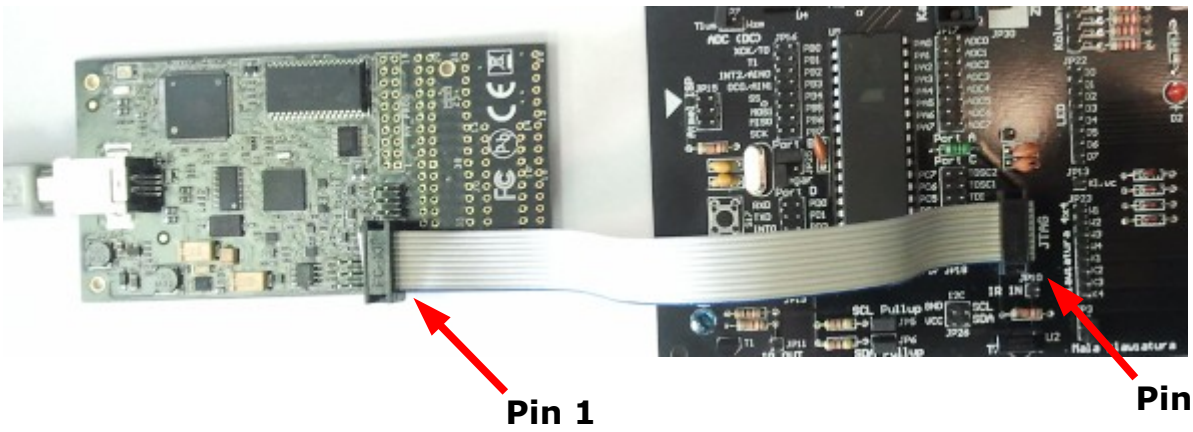


8. Przygotowanie zestawu ZL3AVR do ćwiczeń

8.1. Podłączenie programatora AVR Dragon

Programator AVR Dragon oferuje złącze JTAG albo ISP do programowania mikrokontrolerów AVR. W naszym przypadku będziemy korzystać ze złącza JTAG.

1. Podłącz płytkę AVR Dragon do ZL3AVR poprzez port JTAG. Złącze JTAG znajduje się na wyprowadzeniu JP21 płyty ZL3AVR. Zdjęcie poniżej przedstawia sposób podłączenia. Proszę zwrócić uwagę na lokalizację pierwszego pinu złącza JTAG na module Dragon i płycie ZL3AVR.

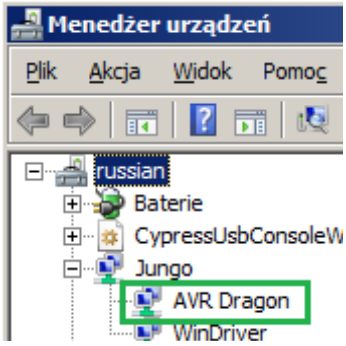


Uwaga

Do połączenia proszę używać krótkich kabli o maksymalnej długości 20 cm. Dłuższe kable wprowadzają tłumienie sygnału, które może uniemożliwić poprawne programowanie.

2. Proszę podłączyć AVR Dragon do portu USB komputera PC. Jeżeli moduł Dragon jest podłączany 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.

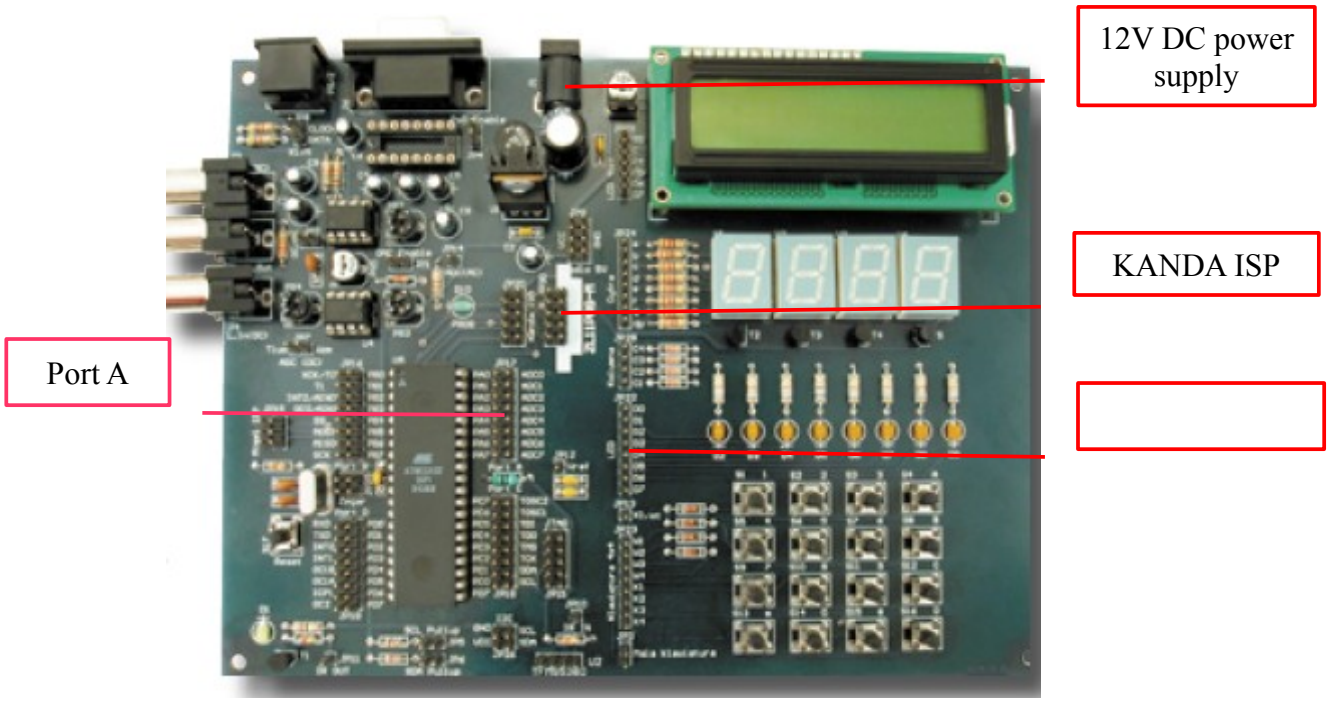
3. W menadżerze urządzeń Windows proszę sprawdzić czy Dragon jest zainstalowany.



8.2. Sposób realizacji połączeń na płycie ZL3AVR

Widok płyty ZL3AVR zaprezentowano poniżej.

1. Proszę połączyć osiem pinów portu A (JP17) do ośmiu wyprowadzeń diod LED (JP22).
2. Proszę podłączyć zasilacz 12V do złącza zasilania ZL3AVR. *Proszę zwrócić uwagę na brak diody sygnalizującej włączenie zasilania na płytce ZL3AVR.*



9. Sterowanie diodami LEDs

Ćwiczenie 1.5

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

Uwaga

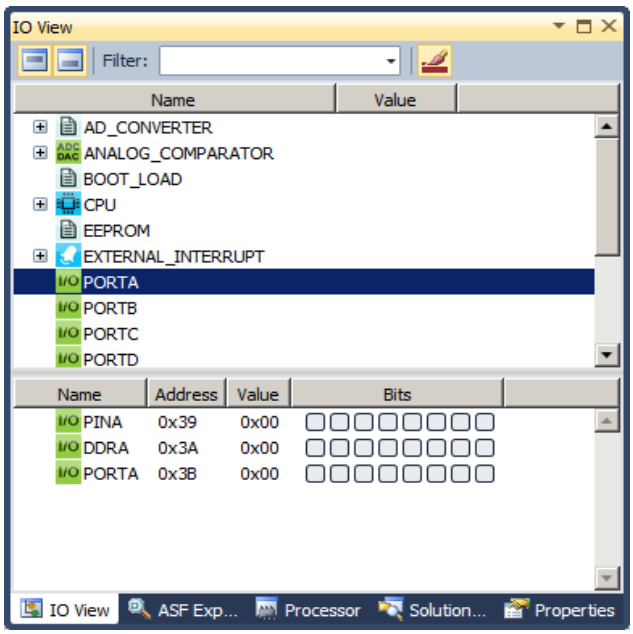
Dobrze jest zapisywać już napisane i więcej nieużywane programy pod nazwami innymi niż ich oryginalna nazwa. W procesie assemblacji Atmel Studio kompiluje tylko plik o nazwie która jest równocześnie nazwą całego projektu. Inne pliki są ignorowane. Dla swojej własnej wygody przy tworzeniu nowego programu zawsze używaj pliku o nazwie projektu. Inne nieużywane już programy archiwizuj pod inną nazwą. Nie kasuj starych programów, bo na pewno jeszcze Ci się przydadzą.

2. Napisz program który zapali parzyste diody LED na ZL3AVR.

Podpowiedź

- Sprawdź na schemacie jaki stan logiczny powoduje zapalenie diody.
- W programie zapisz do rejestru *ddra* wartość która ustawi port A jako wyjście.
- Zapisz odpowiednią wartość do rejestru *porta.*
- Zakończ program instrukcją skoku do samego siebie.

3. Proszę przekrokować program program używając programu AVR Simulator. W symulatorze można zweryfikować stan diod LED sprawdzając wizualizację stanu rejestrów portu A w oknie „IO View.”



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

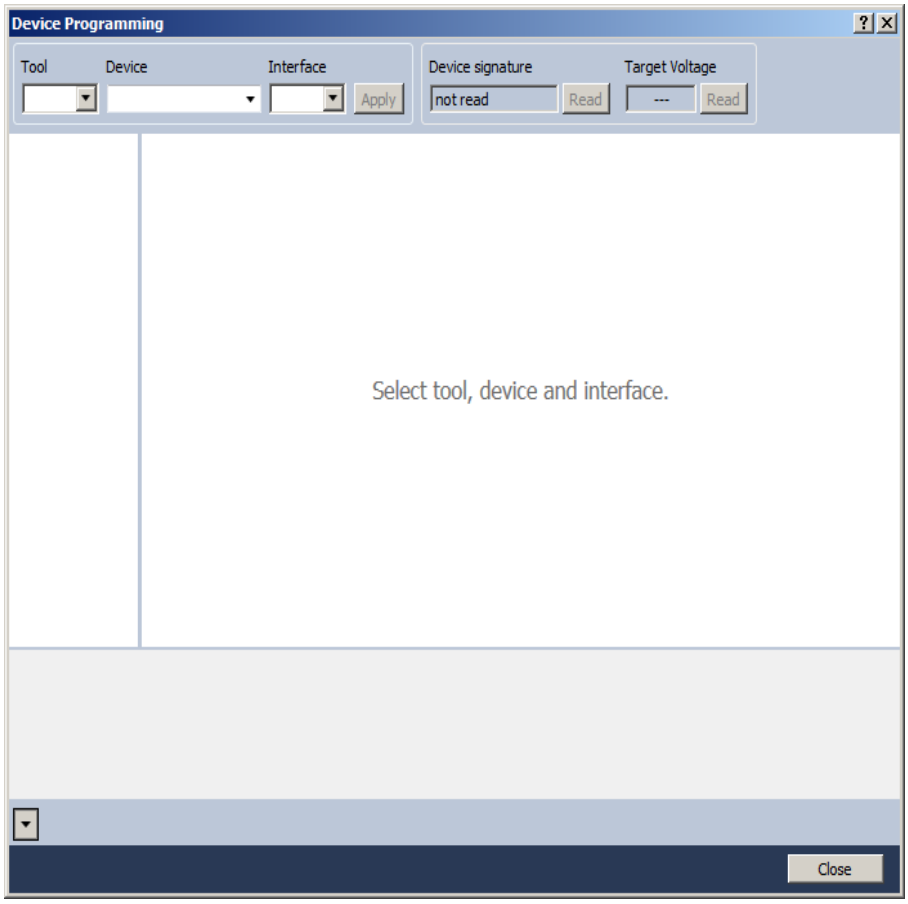
9.1. Programowanie mikrokontrolera

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.

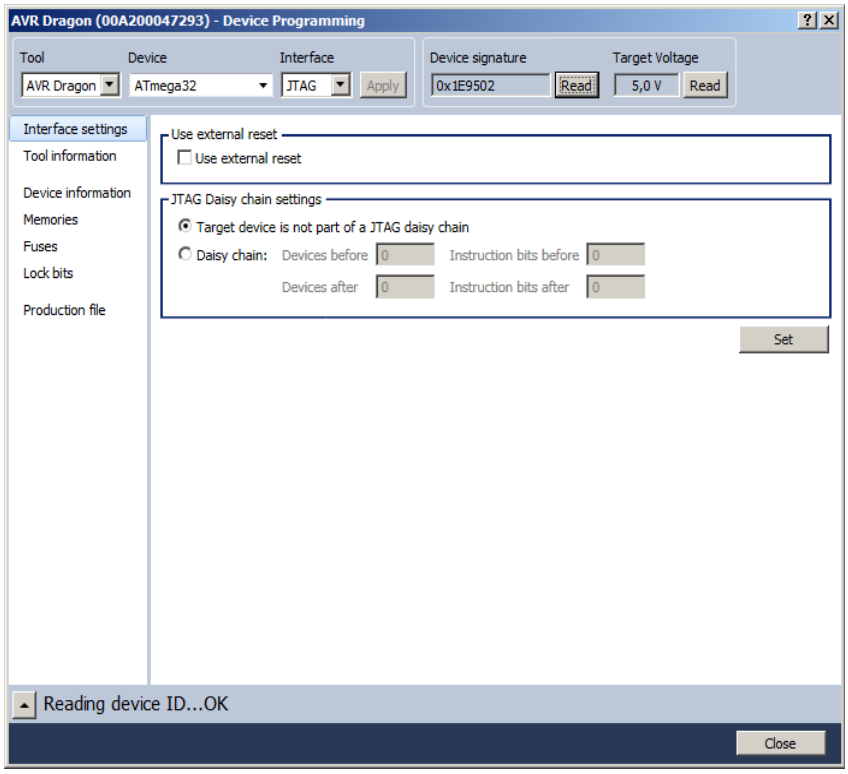
'Tool' → 'AVR Dragon'

'Device' → 'ATMega32'

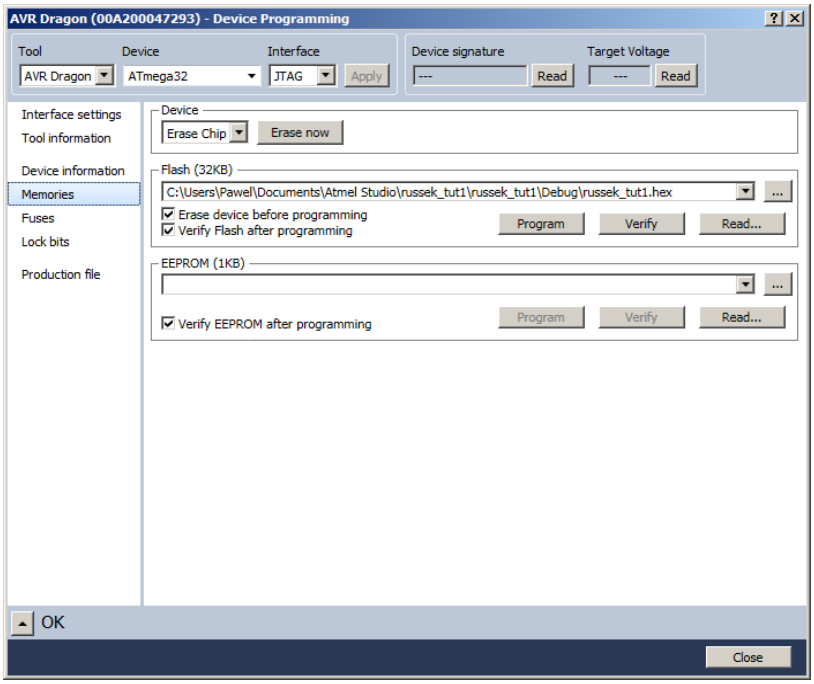
'Interface' → JTAG

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

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

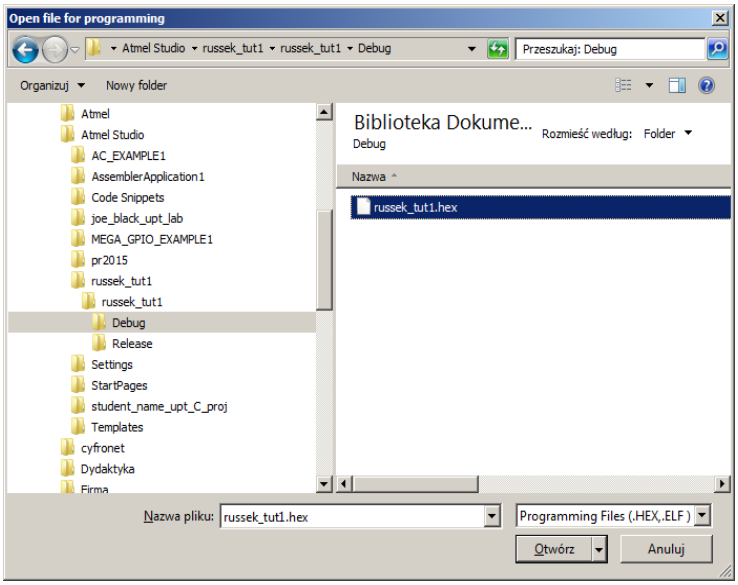


3. Wybierz „Memories” z bocznego menu okna „Device Programming”.

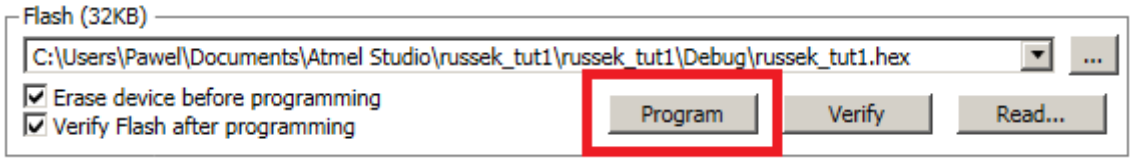


4. W polu „Flash” wybierz plik 'hex' znajdujący się w folderze projektowym (na przykład *shrek_upt_lab.hex*)

Kliknij przycisk '... 'aby otworzyć okno 'Open file for programming', wybierz plik i wciśnij przycisk 'Otwórz'.



5. Aby zaprogramować mikrokontroler kliknij przycisk 'Program'



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 płycie ZL3AVR świecą się diody o parzystych numerach.

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 $0 \leq A \leq 31, 0 \leq b \leq 7$

Przykład:

```
sbi $12,7 ; 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 $0 \leq A \leq 31, 0 \leq b \leq 7$

Przykład:

cbi $12,7 ; Clear bit 7 in Port D

Ćwiczenia 1.6

1. *Napisz program który zaświeci parzyste diody LED na płycie ZL3AVR. Użyj instrukcji sbi i cbi.*
2. *Przekrokuj program obserwując stan rejestrów portu A 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.