|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| logowydzialu | Instytut Informatyki Politechniki Śląskiej  Zespół Mikroinformatyki i Teorii Automatów Cyfrowych  **Laboratorium SMiW** | | logoii | |
| **Rok akademicki** | **Rodzaj studiów\*: SSI/NSI/NSM** | **Numer ćwiczenia:** | **Grupa** | **Sekcja** |
| **2015/2016** | **SSI** | **9** | **4** | **2** |
| **Data i godzina planowana ćwiczenia:**  dd/mm/rrrr - gg:mm | **24/10/2015 8:30** | **Prowadzący**:  JP/KT/GD/GB | **JP** | |
| **Data i godzina wykonania ćwiczenia:**  dd/mm/rrrr - gg:mm | **24/10/2015 8:30** |
| ***Sprawozdanie*** | | | | |
| **Temat ćwiczenia:**  Mikrokontrolery z serii AVR | | | | |
| **Skład sekcji:** | 1. Dominika Błasiak 2. Aleksandra Chronowska | | | |

# Treść zadania

Do portu A mikrokontrolera AVR Atmega256 o zegarze taktowanym z częstotliwością 4 MHz podłączono 8 przycisków podpiętych do masy.

Do portu B podłączono 8 diod świecących przez tranzystor do zasilania.

W dowolnym miejscu w pamięci programu umieszczono tablice - "tab\_ROM" o nieznanej długości i zakończonej słowem 0 x FF oraz tablice o znanej długości "tab\_RAM".

Poprawny zakres działania programu to przedział od 0 -1024 bajtów.

Program ma za zadanie wyświetlić zawartość tablicy” tab\_ROM” za pomocą diod po naciśnięciu pojedynczego przycisku. Każdorazowe wyświetlanie zawartości tablicy występuje z opóźnieniem 1 ms wyliczonej za pomocą częstotliwości zegara wykorzystując przy tym obsługę przerwań.

# Kod programu - język wysokiego poziomu C

|  |
| --- |
| #include <avr/io.h>  #include <avr/pgmspace.h>  #include <avr/interrupt.h>  //rozmiar tablicy ROM  #define ROM\_SIZE 5  #define *F\_CPU* 4000000UL //clock = 4MHz  //tablica umieszczona gdzieś w pamięci ROM  const char ROMTAB[ROM\_SIZE] PROGMEM= {1,2,3,4,0xFF};  //flaga potrzebna do obsłużenia przerwania int0  volatile int int0Flag = 0 ;  //flaga potrzebna dla opóźnienia  volatile int delayFlag = 0 ;  //przerwanie INT0  ISR(INT0\_vect){  int0Flag = 1;  EIMSK |= 0<<INT0;//wylacza przerwania zewnętrzne INT0, znajdujące się one na porcie D  }  //przygotowanie przerwania  void InitINT0()  {  //podłączenie portu D do obsługi przerwań  DDRD = 1<<PD0;  PORTD = 1<<PD0;    EIMSK |= 1<<INT0; //wlaczenie przerwania zewnętrzneINT0    // Falling-Edge Triggered INT0 - This will depend on if you  // are using a pullup resistor or a pulldown resistor on your  // button and port  EICRA |= 1<<ISC01; //trigger (wywoływacz) przerwania INT0    sei();// wlaczenie obsługi przerwań  }  //inicjalizacja portów  void initPorts()  {  DDRA = 0x00; //port a jako wejscie przez rezystor podciagajacy(podłączamy do niego klawisze)  PORTA = 0xFF; //PORTA podpinamy do masy  DDRB = 0xFF; //port b jako wyjscie (podłączamy do niego diody)  PORTB = 0x00;//zapewniamy napiecie niskie, aby diody nie swiecily  }  //timer  ISR(TIMER1\_OVF\_vect){  TIMSK0 = 0<<TOIE0;//wyłączenie przepełnionego przerwania (po upłynięciu 1ms)  delayFlag = 1; // ustawienie flagi na 1  }  void printROMTAB()  {  int i=0;  //wczytaj pierwszy element z tablicy  unsigned char dataBit = pgm\_read\_byte(&(ROMTAB));  while(1)  {  //ustawienie flagi na 0  delayFlag = 0;  // opoznienie  TCNT1 = 65536 - 4000; //ustawienie timera (16 bitow): (2^16 -1)-4000  TCCR1B = 1<<CS10; //ustawienie prescalera  TIMSK1 = 1<<TOIE1; //ustawinie mozliwosci przepelnienia przerwania  while(delayFlag == 0);  //zapalenie wybranej diody odbywa sie przez wystawienie 0 na odpowiednim pinie portu B  PORTB = (dataBit) ^ 0xFF; //0 xor 1 = 1 ; 1 xor 1 = 0;  // jeśli wczytany element jest równy 0xFF wtedy kończy się wykonywanie pętli  if(dataBit == 0xFF)  break;  i++;  //wczytanie kolejnego elementu  dataBit = pgm\_read\_byte(&(ROMTAB[i]));  //czekanie na puszczenie klawisza    }  }  int main(void)  {  InitINT0();  while(int0Flag == 0);  initPorts();  printROMTAB();    } |

# Kod w programu - język niskiego poziomu ASM

|  |
| --- |
|  |

# Wnioski

Laboratorium miało zapoznawało z programowaniem mikrokontrolerów z rodziny AVR przy użyciu obsługi przerwań. Naszym zadaniem było przerobienie dotychczasowo stworzonego przez nas działającego kodu programu z poprzednich laboratorium na kod wykorzystujący obsługę przerwań, działający zarówno w języku wysokiego poziomu – C, jak i w języku niskiego poziomu - w języku asemblerowym. Do tego celu wykorzystałyśmy wygodne środowisko Atmel Studio.

Kod napisany w języku asemblerowym ponownie wypadł pamięciowo i czasowo lepiej od kodu napisanego w języku C. Jednak język wysokiego poziomu okazał się być bardziej przyjemny i łatwiejszy do zaimplementowania.