Gra refleks

projekt realizowany w ramach przedmiotu systemy wbudowane
Lider grupy – Mariusz Gawrysiak

Grupa:
Mariusz Gawrysiak -185692
Piotr Kowalski - 189702
Małgorzata Świtała - 189727

Spis Treści:

Spis treści

Wymagania Funkcjonalne

Opis Projektu

Przygotowany przez naszą drużynę projekt jest grą, w której użytkownik musi w jak najkrótszym czasie dostrzec palącą się diodę i wcisnąć odpowiadający jej przycisk. Liczony jest czas od momentu zapalania się diody do chwili, w której gracz naciśnie odpowiedni przycisk. Po naciśnięciu prawidłowego przycisku dioda zaświeci się na zielono zaś jeśli gracz wybierze nieprawidłowy przycisk dioda zaświeci się na czerwono.

Gracz ma możliwość wybrania ilości tur gry tj. ile razy będzie losowana dioda. Na koniec gry liczony jest średni czas reakcji użytkownika. Wszystkie komunikaty wyświetlane są na wyświetlaczu LCD.

Do projektu wykorzystywany jest moduł eXtrino XL wyposażony w:
- mikro-kontroler ATxmega128A3U-AU
- 8 przycisków ogólnego przeznaczenia
- przycisk FLIP uruchamiający bootloader USB
- 8 diod LED
- złącze MINI-USB
- złącze do kart MicroSD
- złącze programowania zgodne ze standardem PDI oraz JTAG
- podstawki pod płytki zgodne z Arduino, wyświetlacz LCD ze sterownikiem HD44780, rezonator kwarcowy, pamięć I2C, pamięć SPI, przetwornik ADC, potencjometr cyfrowy, wzmacniacz programowalny, termometr LM35/DS18B20
- przycisk resetujący
- Układ stabilizujący napięcie zasilania oraz filtry napięcia zasilającego

Funkcjonalności

Numer	Funkcjonalność	Zastosowanie
1	Obsługa ekranu LCD	Wyświetlanie komunikatów dla użytkownika
2	Dioda RGB	Sygnalizowanie poprawności tury
3	Diody LED	Informacja o przycisku, który należy wcisnąć
4	Przyciski	Wprowadzanie danych przez użytkownika
5	Timer / Counter	Obsługa innych funkcjonalności
6	Przerwania	Pomiar czasu
7	System zdarzeń	Obsługa enkodera obrotowego

Podział Obowiązków

Osoba	Zadania	Udział w projekcie
Mariusz Gawrysiak	Timer/counter, przerwania	1/3
Piotr Kowalski	Enkoder, LCD, dioda RGB	1/3
Małgorzata Świtała	Przyciski, diody LED	1/3

Działanie Programu

Instrukcja użytkownika

Po uruchomieniu programu wyświetla się komunikat powitalny. Następnie użytkownik proszony jest o wybranie ilości tur gry za pomocą enkodera obrotowego. Gdy ilość tur zostanie wybrana należy zatwierdzić ją przyciskiem FLIP i poczekać do pojawienia się komunikatu startu gry a następnie zapalenia się pierwszej diody.

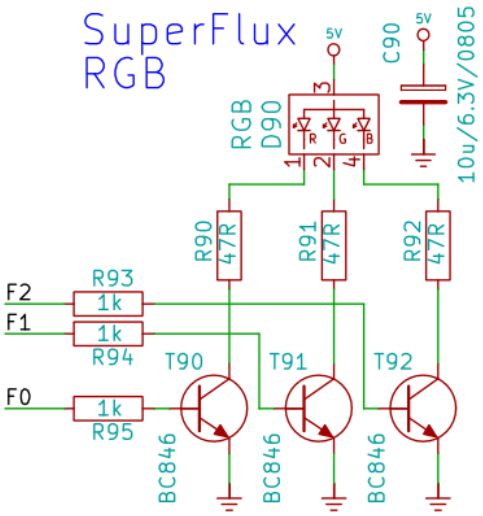
Gracz musi w jak najszybszym tempie przycisnąć przycisk przy mrugającej diodzie. Po przyciśnięciu poprawnego przycisku dioda RGB zaświeci się na czerwono oraz na wyświetlaczu LCD zostanie pokazany czas reakcji użytkownika. Dioda czerwona oznacza przyciśnięcie niepoprawnego przycisku. Po zakończeniu ostatniej tury zostaje pokazany średni czas reakcji.

Opis algorytmu

1. W pierwszym kroku inicjalizujemy wyświetlacz.
2. Następnie konfigurujemy kolejno diody, klawiaturę i timery.
3. Inicjalizujemy en koder obrotowy i konfigurujemy system zdarzeń.
4. Za pomocą en kodera obrotowego wybieramy ilość tur rozgrywki.
5. Rozpoczyna się główna część algorytmu gry – losujemy diodę LED.
6. Program oczekuje na wybór przycisku przez użytkownika. Czas reakcji gracza mierzony jest za pomocą timera.
7. Możliwe są tu dwie drogi:
    a. Użytkownik wybierze poprawny przycisk i czas zostanie uwzględniony w całkowitej średniej wyniku.
    b. Gracz popełni błąd i nic się nie stanie do momentu poprawnego wciśnięcia klawisza. Czas się nie zatrzymuje.
8. Po odegraniu wszystkich rund, gracz jest poinformowany o ostatecznym wyniku.

Opis funkcjonalności poszczególnych elementów projektu

Dioda RGB



Jak widać na schemacie nasza dioda podłączona jest do nóżek F0, F1 oraz F2, które odpowiednio oznaczają kolor: zielony, biały, czerwony. Na sam początek należy ustawić w rejestrze DIRSET portu F aby piny 0,1,2 były traktowane jako wyjścia. Zatem musimy ustawić na tych pinach wartość 1.

```
#define PIN0_bm 0x01
#define PIN1_bm 0x02
#define PIN2_bm 0x04
```

Możemy to zrobić tworząc odpowiednią maskę bitową za pomocą poniższego kodu:

```
PORTF.DIRSET    =    PIN2_bm | PIN1_bm | PIN0_bm;
```

Kiedy mamy już odpowiednio skonfigurowane piny wystarczy ustawić poziom wysoki na rejestr OUTSET odpowiedniego pinu. Dla przykładu, poniższy kod poskutkuje zaświeceniem się diody na zielono:

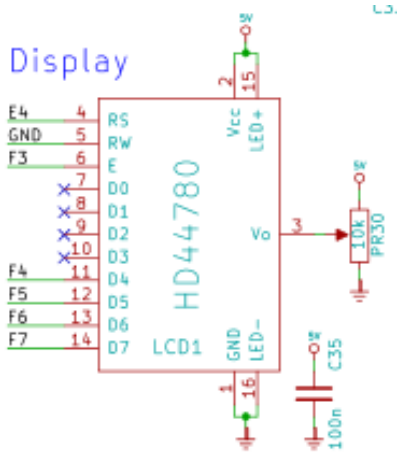
```
PORTF.OUTSET = PIN0_bm;
```

Aby dioda przestała się świecić należy wpisać do rejestru OUTCLR nasz pin F0:

```
PORTF.OUTCLR = PIN0_bm;
```

Oczywiście daje nam to możliwość mieszania kolorów ze sobą. Wystarczy wprowadzić odpowiednią maskę na rejestr OUTSET.

Wyświetlacz LCD



Wyświetlacz który został użyty to WC1602A-STBLWNC06. Sterownik wyświetlacza HD44780 komunikuje się z mikrokontrolerem za pomocą 8- lub 4-bitowej magistrali danych. Krótki opis wejść:

Numer	Wejście	Opis
1	RS	Wybór sygnału (0 – polecenia, 1 - tekst)
2	RW	Odczyt/Zapis danych
3	D0-D7	Dane
4	E	*Enabled* – określa czy można nawiązać połączenie z monitorem
5	VSS	Uziemienie
6	V0	Kontrast

Aby rozpocząć pracę z naszym wyświetlaczem LCD należy zainicjować połączenie. W naszym projekcie używamy biblioteki `HD44780.c`. Poniżej zostały opisane najważniejsze jej elementy.

Do inicjowania połączenia z monitorem używamy metody `LcdInit();` w której na samym początku należy skonfigurować piny potrzebne do przesyłania tekstu na ekran. Używamy do tego rejestru DIRSET następująco:

```
LCD_PORT.DIRSET =    LCD_D4_bm|
                     LCD_D5_bm|
                     LCD_D6_bm|
                     LCD_D7_bm|
                     LCD_E_bm;
LCD_PORT_RS.DIRSET= LCD_RS_bm;
```

Wraz z biblioteką dołączone są stałe #define do ułatwienia programowania oraz poprawienia wyglądu kodu.

```
#define HD44780_CLEAR                   0x01
#define HD44780_HOME                    0x02
#define HD44780_ENTRY_MODE              0x04
    #define HD44780_EM_SHIFT_CURSOR     0
    #define HD44780_EM_SHIFT_DISPLAY    1
    #define HD44780_EM_DECREMENT        0
    #define HD44780_EM_INCREMENT        2
#define HD44780_DISPLAY_ONOFF           0x08
    #define HD44780_DISPLAY_OFF         0
    #define HD44780_DISPLAY_ON          4
    #define HD44780_CURSOR_OFF          0
    #define HD44780_CURSOR_ON           2
    #define HD44780_CURSOR_NOBLINK      0
    #define HD44780_CURSOR_BLINK        1
#define HD44780_DISPLAY_CURSOR_SHIFT    0x10
    #define HD44780_SHIFT_CURSOR        0
    #define HD44780_SHIFT_DISPLAY       8
    #define HD44780_SHIFT_LEFT          0
    #define HD44780_SHIFT_RIGHT         4
#define HD44780_FUNCTION_SET            0x20
    #define HD44780_FONT5x7             0
    #define HD44780_FONT5x10            4
    #define HD44780_ONE_LINE            0
    #define HD44780_TWO_LINE            8
    #define HD44780_4_BIT               0
    #define HD44780_8_BIT               16
#define HD44780_CGRAM_SET               0x40
#define HD44780_DDRAM_SET               0x80
```

Dzięki tym stałym możemy w łatwy sposób konstruować maski i przekazywać je w metodach.

Sygnał RS jest sygnałem wejściowym wyświetlacza i decyduje o tym, czy dana transmisja będzie dotyczyła rejestrów sterujących (stan 0), czy też transmitowane będą dane przeznaczone do wyświetlenia (stan 1). Dlatego przed wysłaniem każdej litery ustawiamy stan wysoki wejścia RS.

W użytej bibliotece przesyłanie liter odbywa się w trybie 4-bitowym. Do przesłania informacji potrzeba dwóch cykli transmisji: najpierw dla bardziej znaczącej, a następnie dla mniej znaczącej połówki bajtu za pomocą linii D4...D7.

Za transmisję danych na ekran odpowiadają poniższe metody wykorzystując metodę wspomnianą wyżej:

```
void _lcd_OutNibble(unsigned char nibbleToWrit
    if(nibbleToWrite & 0b00000001)
        LCD_PORT.OUTSET = LCD_D4_bm;
    else
        LCD_PORT.OUTCLR = LCD_D4_bm;

    if(nibbleToWrite & 0b00000010)
        LCD_PORT.OUTSET = LCD_D5_bm;
    else
        LCD_PORT.OUTCLR = LCD_D5_bm;

    if(nibbleToWrite & 0b00000100)
        LCD_PORT.OUTSET = LCD_D6_bm;
    else
        LCD_PORT.OUTCLR = LCD_D6_bm;

    if(nibbleToWrite & 0b00001000)
        LCD_PORT.OUTSET = LCD_D7_bm;
    else
        LCD_PORT.OUTCLR = LCD_D7_bm;
}

void _lcd_Write(unsigned char dataToWrite) {
    LCD_PORT.OUTSET = LCD_E_bm;
    _lcd_OutNibble(dataToWrite >> 4);
    LCD_PORT.OUTCLR = LCD_E_bm;
    asm volatile("nop");
    LCD_PORT.OUTSET = LCD_E_bm;
    _lcd_OutNibble(dataToWrite);
    LCD_PORT.OUTCLR = LCD_E_bm;
    _delay_us(50);
}
```

Aby wyczyścić ekran wystarczy wywołać metodę:

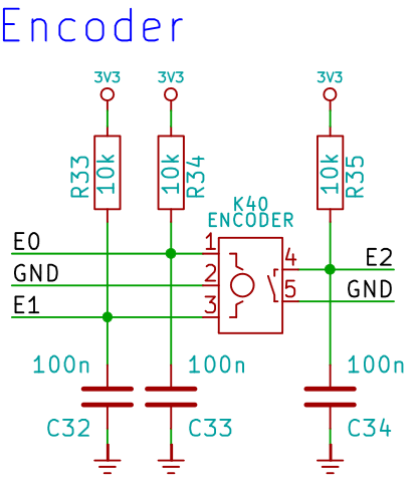
```
void LcdCommand(unsigned char commandToWrite)
```

z parametrem:

```
#define HD44780_CLEAR                   0x01
```

We wszystkie pola zostanie wstawiona spacja. Wynika to z tablicy ROM zawartej w dokumentacji wyświetlacza.

Enkoder Obrotowy



W tej części dokumentacji warto wspomnieć że bardzo ważne są rejestry PINxCTRL, pozwalające skonfigurować bardziej zaawansowane opcje poszczególnych pinów. Co ważne, każdy pin ma osobny rejestr kontrolny. Wpisując do niego odpowiednie wartości, możemy włączać rezystory pull-up, pull-down, keeper. Przy pomocy tych rejestrów konfiguruje się także przerwania lub szybkość narastania zbocza.

W elektronice pull-up to podłączenie linii lub wyprowadzenia układu scalonego do napięcia zasilania poprzez rezystor o wartości kilku kiloomów. Tego typu połączenie powoduje utrzymywanie się na magistrali stanu wysokiego, kiedy żadne urządzenie nie nadaje, a zarazem zapobiega występowaniu stanów nieustalonych.

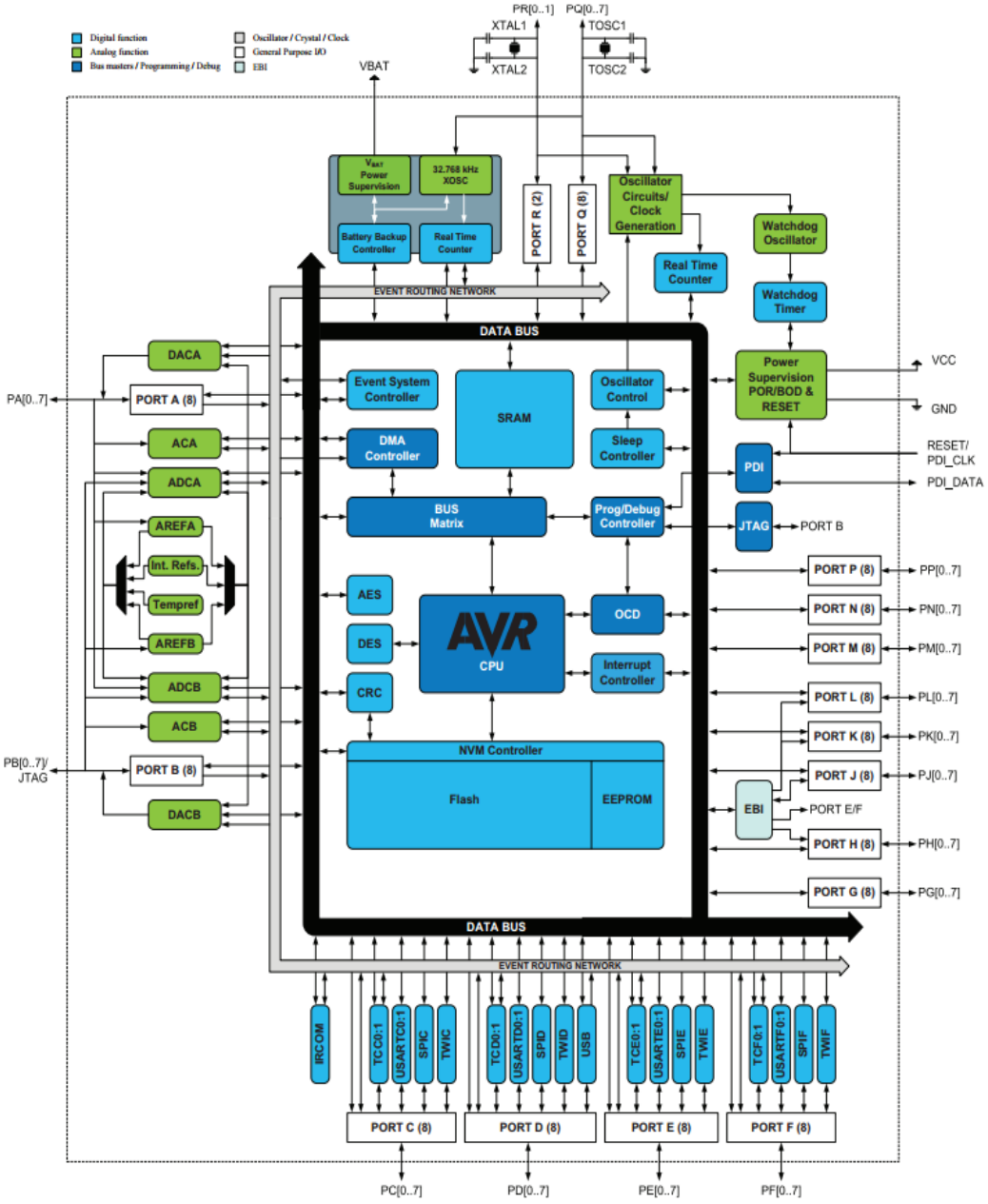
Abyśmy mogli zliczać impulsy jako nadajnik musimy ustawić nasz dekoder kwadraturowy a odbiornikiem musi być timer. W tym celu wykorzystaliśmy timer C1. Przed ustawieniami timera należy skonfigurować system zdarzeń. W rejestrze CH0MUX ustawiamy który z pinów naszego portu będzie wywoływać zdarzenie (czyli Port E - Pin 0). Następnie za pomocą rejestru CH0CTRL włączamy dekoder kwadraturowy oraz filtr cyfrowy. Filtr cyfrowy powoduje ignorowanie impulsów trwających krócej niż wyznaczoną liczbę cykli zegara systemowego. Maksymalna wartość, jaką możemy wybrać to 8 cykli zegarowych – taką właśnie wybraliśmy w naszym projekcie.

Pozostaje tylko skonfigurowanie timera. Jest ono opisane w ostatnim akapicie kolejnego rozdziału.

Timer/Couter i przerwania

2.1 Block Diagram

Figure 2-1. Atmel AVR XMEGA AU block diagram.



W mikrokontrolerach XMEGA timery są ściśle powiązane z portami. Dostępne są po 2 timery na każdym z portów C,D,E oraz tylko 1 na porcie F, a oznaczone są odpowiednio TCC0, TCC1, TCD0, TCD1, TCE0, TCE1, TCF0. Wejścia i wyjścia timerów są dostępne na pinach portu, który posiada dany timer.

Wszystkie timery w XMEGA są 16-bitowe.

Do mierzenia czasu i mrugania wylosowaną diodą wykorzystujemy TCC0.
Nasze timery posiadają kilka trybów pracy, różniących się między sobą kierunkiem liczenia i momentami przepełnienia. My korzystamy z trybu NORMAL, ustawiając w rejestrze CTRLB bit TC_WGMODE_NORMAL_gc:

```
TCC0.CTRLB      =      TC_WGMODE_NORMAL_gc
TCD0.CTRLB      =      TC_WGMODE_NORMAL_gc
```

Timery w XMEGA są taktowane z częstotliwością 2MHz. Interesuje nas mierzenie czasu z dokładnością do 0,1 s więc ustawiamy preskaler na wartość 64 za pomocą rejestru CTRLA wpisując do niego wartość 0101 TC_CLKSEL_DIV64_gc: (TCC0.CTRLA = TC_CLKSEL_DIV64_gc). Ustawienie tego rejestru jednocześnie uruchamia timer. Następnie w 16-bitowym rejestrze PER odpowiadającym za wartość przy której couter zostaje przepełniony ustawiamy 3125: TCC0.PER = 3125.
2^6 /64 /10 = 3125.

Przepełnienie timera wywołuje u nas przerwanie, co konfigurujemy w rejestrze INTCTRLA podając mu wartość TC_OVFINTLVL_LO_gc dzięki czemu ustawiamy również priorytet przerwania na LOW. Należy również odblokować przerwania o tym priorytecie (PMIC.CTRL = PMIC_LOLVLEN_bm) oraz globalnie odblokować przerwania (sei()).

Przerwanie obsługujemy funkcją ISR podając jej wektor przerwania TCC0 (TCC0_OVF_vect):
ISR(TCC0_OVF_vect).

Dzięki powyższej obsłudze przerwania inkrementujemy zmienną „timer” uzyskując pomiar czasu oraz mruganie diodą.

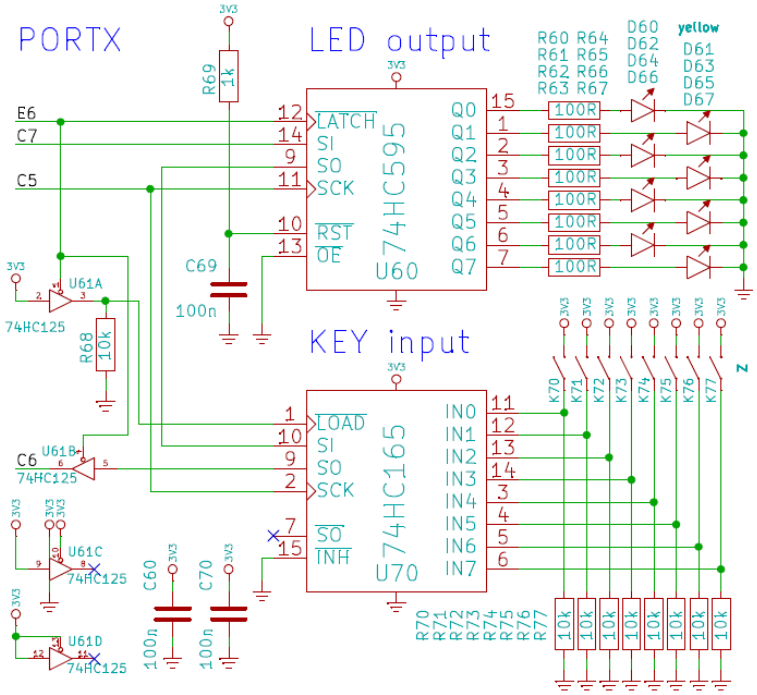
Korzystając z kolejnego timera – TCD0 i konfigurując go na tryb normalny oraz ustawiając preskaler na wartość 1 obsługujemy losowanie diody, do której przyległy przycisk należy wcisnąć w trakcie gry:

```
TCD0.CTRLA      =      TC_CLKSEL_DIV1_gc;
TCD0.CTRLB      =      TC_WGMODE_NORMAL_gc;
```

Kolejny timer – TCC1 służy do taktowania systemem zdarzeń. Ustawiamy na nim źródło taktowania na kanał zdarzeń o nr 0 poprzez rejestr CTRLA podając mu wartość bitu - TC_CLKSEL_EVCH0_gc.
Następnie ustalamy akcję jaką wykonywał będzie timer przy wystąpieniu sygnału na kanale zdarzenia ustawiając dekodowanie kwadraturowe na rejestrze CTRLD poprzez wartość TC_EVACT_QDEC_gc. Na tym samym rejestrze podajemy wartość TC_EVSEL_CH0_gc aby zliczać impulsy z kanału 0.

Ostatni wykorzystywany przez nas timer TCE0 posłużył do zmniejszenia jasności diody RGB. Odpowiednią jasność uzyskaliśmy poprzez pojedyncze „miganie” diodą z częstotliwością około 3KHz. Przerwanie w tym przypadku powoduje tylko pojedyncze zapalenie i zgaszenie diody, natomiast częstotliwość 3KHz wywołana jest wartością 333 w rejestrze PER i preskalowaniem z wartością 1.

Klawiatura



PORTX to wirtualny układ peryferyjny na płytkach eXtrino XL. Jest on jak najbardziej rzeczywisty i funkcjonalny, a słowo wirtualny oznacza to, że dzięki odpowiednim sztuczkom programistycznym, działa on zupełnie tak, jak zwyczajne standardowe porty w mikrokontrolerach XMEGA.

PORTX, tak jak wszystkie inne porty w XMEGA, mają rejestry PORTX.OUT oraz PORTX.IN.

Jako PORTX.OUT oznaczony jest nasz rejestr 74595. Jest to 8-bitowy rejestr wyjściowy, który na płytce eXtrino XL odpowiada za zapalanie ośmiu diod LED.

Natomiast rejestr PORTX.IN czyli nasz rejestr 74165 to 8-bitowy rejestr wejściowy, który połączony jest ośmioma przyciskami. Wciśnięcie przycisku powoduje pojawienie się 1 na odpowiadającym bicie, a wartość 0 oznacza przycisk zwolniony.

W celu kożystania z klawiatury niezbędne jest zdefiniowanie struktury PORTX, w której znajdować się będą rejestry IN i OUT, analogicznie do zwykłych portów XMEGA. Ważne by nie zapomnieć i modyfikatorze volatile, gdyż zmienne te będą aktualizowane w przerwaniach, a bez tego kompilator mógłby nieprawidłowo je zoptymalizować. Struktura ta znajduje się w naszym przypadku w bibliotece `extrino_spi.h`

Poniżej znajduje się instrukcja warunkowa, sprawdzająca czy wciśnięto przycisk przy świecącej się diodzie:

```
if(PORTX.IN == PORTX.OUT)
```

co w naszym programie zostało zapisane jako:

```
if(PORTX.IN == dioda)
```

Zastanówmy się, co się dzieje w tym układzie podczas transmisji jednego bajtu. Sygnał CS zmienia swój stan z 1 na 0, po czym moduł SPI w XMEGA nadaje osiem bitów. Wraz z każdym taktem sygnały

zegarowego SCK, rejestry „przesuwają” swoje bity. W ten sposób, bajt danych z XMEGA trafia do 74595. 8 bitów dotychczas przechowywanych z 74595 przesyłane jest do 74165 i de facto są to dane, które nas nie interesują. Natomiast 8 bitów z 74165, odpowiadające sygnałom wejściowym tego rejestru, przesyła się do modułu SPI w XMEGA. Na zakończenie, zmiana CS z 0 na 1 powoduje odświeżenie wartości tych rejestrów. W ten elegancki sposób, transmitując 8 bitów, ustaliliśmy stan wyjść oraz odczytaliśmy stan wejść.

Aby ręcznie odświeżyć PORTX, należy wywołać funkcję `PortxRefresh();`

Analiza skutków awarii

Element	Wada	Znaczenie
Mikrokontroler	Utrata głównej funkcjonalności projektu	**Krytyczne**
Wyświetlacz LCD	Brak interakcji z użytkownikiem	**Krytyczne**
Kabel USB	Utrata zasilania	**Krytyczne**
Przyciski	Utrata głównej funkcjonalności projektu	**Krytyczne**
Diody	Utrata głównej funkcjonalności projektu	**Krytyczne**
Dioda RGB	Brak informacji o poprawności przyciśniętego przycisku	Niegroźne
Enkoder	Brak możliwości wybrania ilości tur	**Krytyczne**

Do poprawnego działania programu konieczne jest aby działały prawie wszystkie komponenty użyte w projekcie. Mikrokontroler – całe urządzenie musi działać bez zarzutu. Przykładowo uszkodzenie któregoś z wyjść może spowodować awarię kluczowych funkcjonalności projektu.

Awaria wyświetlacza LCD, przycisków, diod oraz enkodera spowoduje brak możliwości interakcji z użytkownikiem. Bez wyświetlacza LCD nie będzie możliwości wybrania ilości tur gry oraz zobaczenia wyniku co jednoznacznie pokazuje krytyczność tej awarii. Bez działającego enkodera obrotowego użytkownik nie będzie w stanie wybrać ilości tur gry co wiążę się z brakiem możliwości rozpoczęcia gry.

Najmniejsze znaczenie dla projektu ma awaria diody RGB. Zniszczenie bądź awaria tego podzespołu spowoduję, że użytkownik nie będzie otrzymywał informacji czy wciśnięty przycisk jest poprawny. Wpłynie to jedynie na komfort gry lub raczej obniży to pozytywne odczucia z gry.

Bibliografia

- http://docplayer.net/2273588-For-messrs-wc1602a-page-1-12-contents.html
- http://inst.eecs.berkeley.edu/~ee40/calbot/webpage/ProgrammingLCD.htm
- http://atmega32.republika.pl/34.htm

- http://www.atmel.com/Images/Atmel-8331-8-and-16-bit-AVR-Microcontroller-XMEGA-AU_Manual.pdf