Dokumentacja

Gra Refleks

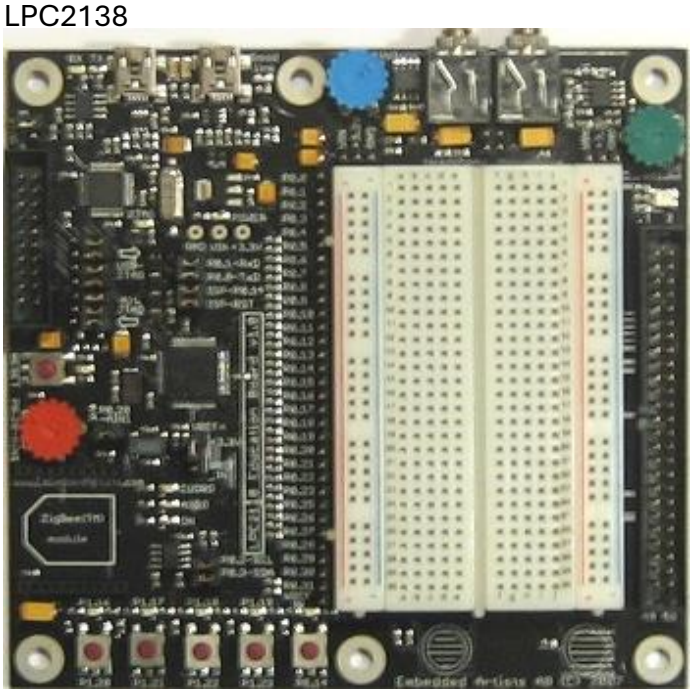
Zespół E04, środa 12:00, Parzyste

Skład zespołu:

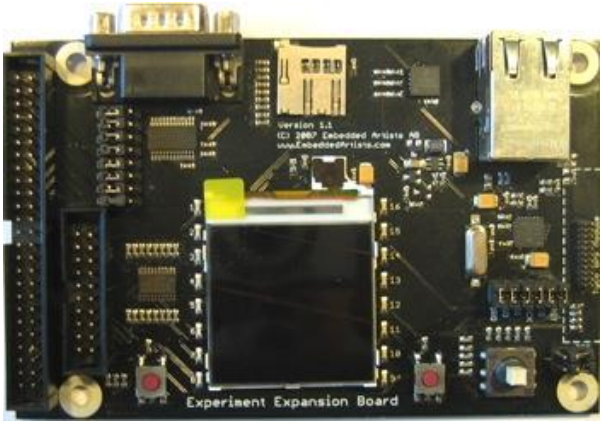
Kacper Białek (Kapitan)
Piotr Janiszek
Franciszek Pawlak

Wykorzystywane urządzenia:
LPC2138 + Expansion board
Nie wykorzystano żadnego dodatkowego sprzętu

LPC2138



Expansion board



Spis treści

1. Podział pracy i lista funkcjonalności

Kacper Białek - 33%
Piotr Janiszek - 34%
Franciszek Pawlak - 33%

Zaimplementowane funkcjonalności:

1. GPIO
2. SPI
3. EKRAN LCD
4. TIMER
5. I^2C
6. EEPROM
7. POTENCJOMETR
8. OBSŁUGA PRZERWAŃ

Odpowiedzialność za funkcjonalności	
TIMER	Piotr Janiszek
EKRAN LCD	Piotr Janiszek
GPIO	Kacper Białek
SPI	Kacper Białek
I^2C	Franciszek Pawlak
EEPROM	Franciszek Pawlak
OBSŁUGA PRZERWAŃ (IRQ)	Piotr Janiszek
POTENCJOMETR	Kacper Białek

2. Instrukcja użytkownika, opis rozgrywki

Po wciśnięciu dowolnego przycisku z P1.16 – P1.19 odpowiadającemu rozpoczęciu rozgrywki po odpowiednim czasie zostanie zapalona losowa dioda. Należy zgasić ją wciskając przyporządkowany jej przycisk w czasie odpowiednim ustawieniu trudności. Jeśli w danym czasie nie zostanie zarejestrowane wciśnięcie, gra zakończy się porażką gracza. Następnie zostanie zapalona kolejna dioda po ustawionym okresie przerwy. Sumarycznie diody zostaną zapalone 10 razy. Końcowo na ekranie zostanie wyświetlony obliczony punktowy wynik rozgrywki.

Przed grą należy dostosować poziom trudności rozgrywki do swoich potrzeb ustawiając odpowiednie położenie potencjometru. Na poziom trudności składa się okres czasu, w którym należy dokonać wciśnięcia odpowiedniego przycisku, zaś okres czasu jaki będzie dzieliło zgaszenie diody po naciśnięciu od zapalenia kolejnej diody wynosi stałe 1000ms.

Najlepszy uzyskany wynik jest automatycznie zapisywany w pamięci urządzenia. Najwyższy wynik wyświetla się na ekranie zawsze przed rozgrywką, aby móc potem porównać swoją rozgrywkę do najlepszego rezultatu i śledzić swoje postępy.

3. Opis działania programu

Funkcja init_irq inicjalizuje obsługę przerwań od Timera #1. Konfiguruje kontroler przerwań wektorowych (VIC), ustawia adres procedury przerwania oraz programuje timer w taki sposób, aby generował przerwania z określonym okresem i współczynnikiem wypełnienia. Służy do aktualizacji ziarna używanego do losowania diód.

Funkcja zapisz_wynik porównuje nowy wynik z aktualnym rekordem i jeśli nowy wynik jest wyższy, zapisuje go w pamięci EEPROM. Po udanym zapisie wyświetla komunikat na wyświetlaczu LCD.

Funkcja wczytaj_wynik odczytuje zapisany wynik z pamięci EEPROM i zwraca go jako wartość całkowitą.

Funkcja game implementuje logikę gry Refleks, w której użytkownik musi nacisnąć odpowiedni przycisk w odpowiednim czasie, aby zdobyć punkty. Gra składa się z 10 rund, a poziom trudności określa czas reakcji, jaki ma gracz na każdą rundę.

Główna funkcja programu składa się z 5 segmentów:

1. Inicjalizacja urządzeń peryferyjnych:

Inicjalizacja wyświetlacza LCD: Ustawienie i przygotowanie wyświetlacza do prezentacji informacji dla użytkownika.

Inicjalizacja komunikacji I2C: Konfiguracja magistrali I2C, która jest używana do komunikacji z pamięcią EEPROM oraz innymi peryferiami.

Inicjalizacja przetwornika ADC: Przygotowanie przetwornika do odczytywania wartości analogowych, co ma kluczowe znaczenie dla detekcji poziomu trudności opartego na analogowym wejściu użytkownika.

Inicjalizacja obsługi przerwań: Ustawienie timerów i odpowiednich przerwań, które są wykorzystywane do zarządzania ziarnem w grze oraz obsługą cyklicznych zdarzeń związanych z aktualizacją ziarna.

2. Konfiguracja pamięci i wyświetlania danych:

Odczytanie najwyższego wyniku: Odczyt z pamięci EEPROM najwyższego wyniku zapisanego z poprzednich sesji, co pozwala na śledzenie postępów gracza.

Wyświetlenie informacji startowych na LCD: Przedstawienie użytkownikowi bazowych informacji takich jak nazwa gry, aktualny najwyższy wynik, oraz instrukcje dotyczące rozpoczęcia gry.

3. Obsługa wejścia użytkownika:

Wybór poziomu trudności: Poziom trudności jest ustalany na podstawie odczytów z przetwornika ADC. Użytkownik może zmieniać poziom trudności poprzez analogowe sterowanie pokrętłem potencjometru.

Oczekiwanie na akcję użytkownika: Funkcja main w pętli oczekuje na naciśnięcie przycisku przez użytkownika, co jest sygnałem do rozpoczęcia gry.

4. Rozpoczęcie gry:

Po wyborze poziomu trudności i naciśnięciu przez użytkownika dowolnego z przycisków odpowiedzialnych za start, rozpoczyna się właściwa gra, gdzie w zależności od czasu reakcji użytkownika przydzielane są punkty. Dzieje się tak poprzez wywołanie funkcji game, która implementuje algorytm jednej sesji rozgrywki.

5. Zakończenie gry:

Po zakończeniu gry wynik jest wyświetlany na LCD, a w przypadku osiągnięcia nowego rekordu, wynik jest zapisywany w pamięci EEPROM.

Czas w którym należy dokonać wciśnięcia diody jest uzyskiwany dzięki obliczaniu poziomu trudności ze wzoru:
$LVL = AIN1 \,/\, 400$

Gdzie AIN1 jest wartością uzyskiwaną z potencjometru, a zakres możliwych wartości mieści się w [0; 1023], co przy dzieleniu przez 400 daje nam 3 poziomy trudności.

Odpowiednio dla wybranego poziomu trudności maksymalny czas reakcji wynosi:

LVL 0 = 3000ms

LVL 1 = 2000ms

LVL 2 = 1000ms

Czas który dzieli zgaśnięcie jednej diody od zapalenia drugiej obliczany jest stały i wynosi 1000ms. Jest on realizowany przez funkcję mdelay, która odczekuje podany w parametrze czas w milisekundach.

Końcowy wynik punktowy obliczany jest ze wzoru:

$$points = \sum_{i=1}^{10} 4000 - t_{LVL} - t_i$$

Gdzie t_{LVL} to maksymalny czas reakcji zależny od poziomu trudności. Należy zauważyć, że punktowo premiowane jest wybraniu wyższego poziomu. Natomiast t_i to czas reakcji użytkownika w danej rundzie.

PLCK w naszym układzie wynosi 58 982 400Hz.

Jest tak ponieważ PLCK = (FOSC * PLL_MUL)/ PBSD, jeśli USE_PLL 1

PLCK = (14745600Hz * 4)/1 = 58 982 400Hz

Ustawienia w naszym programie:

FOSC = 14745600UL

USE_PLL = 1

PLL_MUL = 4

PBSD = 1

4. Techniczny opis działania GPIO w programie

General Purpose Input/Output (GPIO) to uniwersalny interfejs wejścia/wyjścia dostępny w LPC2138. GPIO pozwala na komunikację mikrokontrolera z innymi urządzeniami i modułami. Każdy pin GPIO może być skonfigurowany jako wejście lub wyjście, co umożliwia odczyt stanu pinu lub sterowanie nim.

GPIO posiada następujące rejestry:
PINSEL: Rejestr ten służy do konfiguracji funkcji pinów. Każdy pin może pełnić różne funkcje, a PINSEL pozwala na ich wybór. Wartości w tym rejestrze decydują, czy dany pin będzie pełnił funkcję wejścia/wyjścia ogólnego (GPIO), czy też będzie używany do innych celów, takich jak komunikacja szeregowa.

IODIR: Rejestr kierunku wejścia/wyjścia (Input/Output DIRectory). Umożliwia ustawienie pinów jako wejścia (ustawienie bitu na 0) lub wyjścia (ustawienie bitu na 1).

IOSET: Rejestr ustawienia stanu wyjścia (Output SET). Zapisanie 1 do odpowiedniego bitu tego rejestru ustawia stan wyjścia na wysoki.

IOCLR: Rejestr czyszczenia stanu wyjścia (Output CLR). Zapisanie 1 do odpowiedniego bitu tego rejestru ustawia stan wyjścia na niski.

IOPIN: Rejestr stanu pinów (PIN State). Odczyt z tego rejestru zwraca aktualny stan pinów. Zapis do tego rejestru zmienia stan wyjść.

Przed użyciem pinów GPIO należy odpowiednio skonfigurować rejestry PINSEL, aby piny były ustawione jako wejścia/wyjścia ogólne. Następnie, za pomocą rejestrów IODIR, IOSET, IOCLR i IOPIN można kontrolować stan i kierunek tych pinów.

Aby skonfigurować rejestr PINSEL, w celu możliwości użycia potrzebnych diod P1.16-P1.19 i przycisków P1.20-P1.23 należy ustawić wartość 1 dla 3 bitu rejestru PINSEL2:
PINSEL2 &= ~(1 << 3);

Spowoduje to, że piny P1.25-16 są używane jako port śledzenia.

Konfiguracja pinów GPIO w LPC2138 jest realizowana za pomocą rejestrów IODIR (Input/Output Direction Register). Rejestr IODIR jest używany do ustawienia kierunku pinów GPIO jako wejście (0) lub wyjście (1). Na przykład, aby skonfigurować pin P0.0 jako wyjście, można użyć:
IO0DIR |= 0x00000001;

Do odczytu i zapisu stanu pinów GPIO służą rejestry IOSET (Output Set Register) i IOCLR (Output Clear Register).

Aby ustawić pin P0.0 na wysoki stan:

IO0SET = 0x00000001;

Aby zresetować pin P0.0 do stanu niskiego:

IO0CLR = 0x00000001;

Stan pinu wejściowego można odczytać za pomocą rejestru IOPIN. Aby odczytać stan pinu P0.0, można użyć:

unsigned int pinState = IO0PIN & 0x00000001;

Operacji tej używamy w kodzie, aby zweryfikować, czy dany przycisk został wciśnięty, co wywołuje zgaszenie diody:

```
if ((IOPIN1 & 0x00080000) == 0)
        IOSET1  = 0x00008000;
  else
        IOCLR1  = 0x00008000;
```

Przykład powyżej dla jednej z diód.

Używane przez nas diody LED charakteryzują się tym, że anoda podłączona jest do napięcia zasilania przez rezystor, zaś druga nóżka diody, katoda, podłączona jest do pinu GPIO, układ taki nazywamy “aktywnym niskim”, dla niego dioda świeci, gdy na pinie jest stan niski.

Aby uzyskać na diodzie stan wysoki, podnosząc napięcie na pinie zbliżone do napięcia zasilania układu, gasząc tym samym daną diodę:

IOSET1 = 0x00008000;

Aby uzyskać na diodzie stan aktywny niski, ustawiając na pinie niskie napięcie, bliskie 0V, zapalając tym samym daną diodę:

IOCLR1 = 0x00008000;

5. Techniczny opis działania SPI w programie

Interfejs SPI (Serial Peripheral Interface) jest jednym z kluczowych interfejsów komunikacyjnych dostępnych na płytce. Jest to interfejs synchroniczny, który umożliwia szybką komunikację między mikrokontrolerem a różnymi urządzeniami peryferyjnymi. W naszym programie został wykorzystany do komunikacji z wyświetlaczem.

Interfejs SPI na płytce LPC2138 obsługuje buforowanie i zmienną długość danych. Dzięki temu jest bardzo elastyczny i może być używany do komunikacji z szerokim zakresem urządzeń peryferyjnych.

Interfejs SPI obsługuje buforowanie, co oznacza, że może przechowywać dane w buforze przed ich wysłaniem lub po otrzymaniu. To pozwala na płynną i efektywną komunikację,

Zmienna długość danych: Interfejs SPI może obsługiwać dane o różnej długości. To oznacza, że może komunikować się z urządzeniami, które wymagają różnych formatów danych.

Liczba cykli PCLK oraz wartość rejestru SPI_SPCCR decyduje o szybkości zegara SPI. Dzięki temu użytkownik może dostosować szybkość transmisji do wymagań konkretnego urządzenia peryferyjnego. Wartość SPI_SPCCR musi być parzysta i nie mniejsza niż 8.

Szybkość SPI użyta w naszym programie jest obliczana ze wzoru:

$$v = \frac{PCLK}{SPI\_SPCCR} = \frac{58982400Hz}{8} = 7372800Hz$$

Rejestry SPI:

SPI Control Register (S0SPCR) - Rejestr S0SPCR steruje działaniem SPI, w tym konfiguracją trybu pracy, fazy i polaryzacji zegara oraz innymi ustawieniami, takimi jak ustawienia fazy i polaryzacji zegara oraz trybu master/slave. Umożliwia również ustawienie przesyłu danych z najmniej znaczącym bitem (LSB) lub najbardziej znaczącym bitem (MSB) na początku .

SPI Status Register (S0SPSR) - Rejestr S0SPSR monitoruje status interfejsu SPI, w tym zakończenie transferu danych oraz warunki wyjątkowe, takie jak WCOL (kolizja zapisu) oraz MODF (błąd trybu).

SPI Data Register (S0SPDR) - Dwukierunkowy rejestr S0SPDR służy do przesyłania i odbierania danych przez SPI. Dane do transmisji są zapisywane do tego rejestru, a odbierane dane mogą być z niego odczytywane. Zapis do tego rejestru inicjuje transfer przez SPI.

SPI Clock Counter Register (S0SPCCR) - Rejestr S0SPCCR określa częstotliwość zegara SPI w trybie master poprzez ustawienie liczby cykli zegara PCLK, które definiują jeden cykl zegara SPI. Wartość ta musi być liczbą parzystą i nie mniejszą niż 8 .

SPI Interrupt Register (S0SPINT) - Rejestr S0SPINT zawiera flagę przerwania dla interfejsu SPI.

Konfiguracja: Pierwszym krokiem jest skonfigurowanie kontrolera SPI. To obejmuje ustawienie odpowiednich bitów w rejestrach konfiguracyjnych kontrolera SPI. Należy skonfigurować takie parametry jak szybkość transmisji, tryb pracy (master/slave), długość danych itp.

Przykładowo:

```
SPI_SPCR = 0x00;  // Reset rejestrów

SPI_SPCR |= (1 << 5); // Ustawienie mikrokontrolera jako Master (MSTR = 1)

SPI_SPCR |= (1 << 4); // Ustawienie CPOL = 1 (aktywny niski stan zegara)

SPI_SPCR |= (1 << 3); // Ustawienie CPHA = 1 (zegar próbkujący na drugim zboczu)

SPI_SPCR &= ~(1 << 6); // Ustawienie kolejności MSB first (LSBF = 0)

SPI_SPCR |= (0x08);   // Ustawienie długości danych na 8 bitów (BITS = 0x8)

SPI_SPCCR = 8;  // wartość musi być większa lub równa 8 i parzysta, aby ustawić prędkość SPI
```

Dodatkowo potrzebna może być również konfiguracja pinów GPIO do komunikacji z innym urządzeniem, co potrzebowaliśmy wykorzystać, w celu komunikacji z wyświetlaczem. Dokładniejszy opis w sekcji LCD.

Transmisja danych: Po skonfigurowaniu kontrolera SPI, można rozpocząć transmisję danych. Dane do transmisji są zapisywane do rejestru danych SPI. Kontroler SPI następnie przesyła te dane do urządzenia peryferyjnego.

Przykładowo, aby wysłać dane:

```
void spi_send_data(tU8 data)

{

   SPI_SPDR = data;  // Zapis danych do rejestru danych SPI

   while (!(SPI_SPSR & (1 << 7)));  // Oczekiwanie na zakończenie transmisji (SPIF = 1)

}
```

Odbiór danych: Kontroler SPI może również odbierać dane od urządzeń peryferyjnych. Odebrane dane są przechowywane w rejestrze danych SPI, skąd mogą być odczytane.

Przykładowo, aby odebrać dane:

```
tU8 spi_receive_data(void)

{

   SPI_SPDR = 0xFF;  // Wysłanie przykładowych danych, aby wygenerować zegar SPI
```

```
  while (!(SPI_SPSR & (1 << 7)));  // Oczekiwanie na zakończenie transmisji (SPIF = 1)
  return SPI_SPDR;  // Odczyt odebranych danych z rejestru danych SPI
}
```

Obsługa przerwań: Kontroler SPI może generować przerwania w odpowiedzi na różne zdarzenia, takie jak na przykład zakończenie transmisji danych. Te przerwania mogą być obsługiwane przez odpowiednie procedury obsługi przerwań. Więcej na temat obsługi przerwań w rozdziale poświęconym temu tematowi. Aby obsługiwać przerwania należy skonfigurować VIC.

6. Techniczny opis działania Ekranu LCD w programie

Wyświetlacz służy do prezentowania różnego rodzaju informacji na ekranie i komunikacji z użytkownikiem, u nas takich jak ekran powitalny, menu, a także ekran gry. Na ekranie gry wyświetlane są istotne informacje, takie jak instrukcje dotyczące przycisków do naciśnięcia oraz czas reakcji. Urządzenie to może również wyświetlać najlepszy wynik osiągnięty przez gracza.

Wyświetlacz umieszczony jest na płytce rozszerzeń, zwanej expansion board, ma rozdzielczość 128x128 pikseli. Komunikacja z wyświetlaczem odbywa się za pomocą interfejsu SPI (Serial Peripheral Interface), który jest standardem komunikacji służącym do przesyłania danych między urządzeniami peryferyjnymi w systemach wbudowanych.

Aby móc korzystać z wyświetlacza, musimy najpierw zainicjować dla niego interfejs SPI. Robimy to za pomocą funkcji o nazwie initSpiForLcd. Ta funkcja jest odpowiedzialna za przygotowanie interfejsu SPI do komunikacji z wyświetlaczem, co jest niezbędne do prawidłowego działania urządzenia. Po jej wywołaniu, możemy zacząć korzystać z wyświetlacza i prezentować na nim różnego rodzaju informacje. Następnie możemy skorzystać z funkcji lcdInit odpowiedzialnej za inicjalizację ekranu, dalej ustawić kolor tła i tekstu funkcją lcdColor. Po tym możemy już swobodnie przemieszczać kursor po ekranie funkcją Gotoxy, wyświetlać różne rzeczy za pomocą niżej opisanych metod oraz czyścić ekran funkcją Clrscr.

Funkcja initSpiForLcd():

IODIR |= (LCD_CS | LCD_CLK | LCD_MOSI); //Ustawia piny GPIO odpowiedzialne za chip select (CS), zegar (CLK) i Master Out Slave In (MOSI) jako wyjścia.

IOSET = LCD_CS; //Dezaktywacja wyświetlacza poprzez ustawienie sygnału chip select (CS) w stan wysoki, co jest sygnałem dla wyświetlacza, że nie jest on aktualnie wybierany do komunikacji.

PINSEL0 |= 0x00001500; //Konfiguruje piny, które będą używane przez interfejs SPI. Ustawienie odpowiednich bitów w rejestrze PINSEL0 pozwala na skojarzenie fizycznych pinów mikrokontrolera z funkcjami peryferyjnymi SPI.

SPI_SPCCR = 0x08; //Konfiguracja częstotliwości taktowania SPI poprzez ustawienie wartości rejestru SPI_SPCCR. Wartość 0x08 sugeruje ustawienie dzielnika zegara.

SPI_SPCR = 0x20; //Ustawienie trybu pracy SPI jako master.

Funkcją selectLCD, w zależności od parametru o typie boolean możemy aktywować lub dezaktywować wyświetlacz, poprzez:

IOSET = LCD_CS; //Dezaktywacja wyświetlacza

IOCLE = LCD_CS; //Aktywacja wyświetlacza

Aby wysłać do ekranu dane należy użyć funkcji sendToLCD:

Funkcja ta wysyła 9-bitową ramkę danych do kontrolera LCD, gdzie pierwszy bit określa, czy następne bajty będą traktowane jako komendy czy dane.

void sendToLCD(tU8 firstBit, tU8 data);

firstBit - pierwszy bit ramki, który decyduje o typie przesyłanych danych (0 dla komendy, 1 dla danych).

data - bajt danych, który ma zostać przesłany po ustawieniu typu.

//Na początku funkcja deaktywuje interfejs SPI, aby umożliwić ręczne sterowanie pinami MOSI (Master Out Slave In) i zegarem SPI.

IOCLR = LCD_CLK; // Wyzerowanie zegara LCD

PINSEL0 &= 0xffffc0ff; // Konfiguracja pinów do kontroli GPIO zamiast SPI

if (1 == firstBit)

  IOSET = LCD_MOSI; // Ustawienie MOSI dla wysłania bitu danych

else

IOCLR = LCD_MOSI; // Resetowanie MOSI dla wysłania bitu komendy

//Przełączanie zegara (pinu LCD_CLK), najpierw ustawiając go w stan wysoki, a potem niski. Jest to konieczne, aby przesłać pierwszy bit do kontrolera LCD.

IOSET = LCD_CLK; // Ustawienie zegara wysokiego

IOCLR = LCD_CLK; // Resetowanie zegara niskiego

//Po przesłaniu pierwszego bitu, funkcja ponownie konfiguruje interfejs SPI, aby reszta danych mogła być wysłana w sposób automatyczny.

SPI_SPCCR = 0x08; // Ustawienie dzielnika zegara SPI

SPI_SPCR = 0x20; // Ustawienie rejestru kontrolnego SPI

PINSEL0 |= 0x00001500; // Ponowne podłączenie funkcji SPI do pinów

//Bajt danych jest wysyłany przez interfejs SPI. Funkcja czeka, aż transmisja zostanie zakończona, sprawdzając odpowiedni bit w rejestrze statusu SPI (SPI_SPSR).

SPI_SPDR = data; // Wysłanie danych

while((SPI_SPSR & 0x80) == 0); // Czekanie na zakończenie transmisji

Aby zainicjować ekran należy użyć funkcji lcdInit:

//bkgColor oraz textColor są ustawiane na 0. Są to zmienne odpowiedzialne za przechowywanie informacji o aktualnym kolorze tła i tekstu na wyświetlaczu.

bkgColor = 0;

textColor = 0;

//Skonfigurowania interfejsu SPI oraz aktywowacja kontrolera LCD

initSpiForLcd();

selectLCD(TRUE);

//Działanie mające na celu zresetowanie wszystkich wewnętrznych ustawień kontrolera LCD.

lcdWrcmd(LCD_CMD_SWRESET);

sdelay(1);

//Konfiguracja wyświetlacza

lcdWrcmd(LCD_CMD_SLEEPOUT); //Wybudza wyświetlacz z trybu uśpienia

lcdWrcmd(LCD_CMD_DISPON); //Włącza wyświetlacz

lcdWrcmd(LCD_CMD_BSTRON); //Aktywacja zasilania ekranu

sdelay(1);

lcdWrcmd(LCD_CMD_MADCTL); //Ustawienie trybu dostępu do pamięci

lcdWrdata(MADCTL_HORIZ); //Określa orientację wyświetlanego obrazu, poprzez lustrzane odbicie poziome, oraz format kolorów BGR zamiast RGB.

lcdWrcmd(LCD_CMD_COLMOD); //Konfiguracja formatu kolorów

lcdWrdata(0x02); //Wybranie trybu kolorów posiadającego 256 wartości, 8-bitowa głębia

lcdWrcmd(LCD_CMD_INVON); //Aktywacja trybu standardowego wyświetlania kolorów, braku inwersji kolorów

//Konfiguracja konwersji kolorów na wyższą głębię

lcdWrcmd(LCD_CMD_RGBSET); //Definicja look-up-table dla kolorów, która będzie konwertowała 8-bitowe wartości kolorów na 12-bitowe wartości

lcdWrdata(0); //Red

lcdWrdata(2);

lcdWrdata(4);

lcdWrdata(6);

lcdWrdata(9);

lcdWrdata(11);

lcdWrdata(13);

lcdWrdata(15);

lcdWrdata(0); //Green

lcdWrdata(2);

lcdWrdata(4);

lcdWrdata(6);

lcdWrdata(9);

lcdWrdata(11);

lcdWrdata(13);

lcdWrdata(15);

lcdWrdata(0); //Blue

lcdWrdata(6);

lcdWrdata(10);

lcdWrdata(15);

//Na końcu dezaktywujemy kontroler LCD, ustawiamy początkowy kontrast na 56 oraz czyścimy ekran

selectLCD(FALSE);

lcdContrast(56);

lcdClrscr();

Aby wyczyścić ekran należy użyć funkcji lcdClrscr:

tU32 i;

//Na początku zmienne lcd_x i lcd_y, które przechowują aktualną pozycję kursora na wyświetlaczu, są resetowane do wartości 0.

lcd_x = 0;

lcd_y = 0;

selectLCD(TRUE) //aktywacja kontrolera wyświetlacza

lcdWindow1(255,255,128,128); //Ustawia okno wyświetlania na całej powierzchni wyświetlacza.

lcdWrcmd(LCD_CMD_RAMWR); //Polecenie zapisu do pamięci RAM wyświetlacza, które przygotowuje wyświetlacz do przyjęcia danych pikseli.

//Iteracja 16900 razy, co odpowiada liczbie pikseli w typowym wyświetlaczu o rozmiarze 130x130 pikseli, do wyświetlacza wysyłany jest kolor tła, każda iteracji zapisuje wartość koloru do kolejnego piksela w pamięci RAM wyświetlacza.

for(i=0; i<16900; i++)

lcdWrdata(bkgColor);

//Dezaktywacja kontrolera wyświetlacza

selectLCD(FALSE);

Funkcja lcdOff() efektywnie zarządza wyłączeniem wyświetlacza LCD, minimalizując jego zużycie energii przez wprowadzenie go w tryb uśpienia.

Składa się z instrukcji:

lcdClrscr(); //Czyścimy wyświetlacz, wypełniając go kolorem tła

selectLCD(TRUE); //Aktywujemy kontroler wyświetlacza

lcdWrcmd(LCD_CMD_SLEEPIN); //Powoduje, że wyświetlacz przechodzi w tryb uśpienia

selectLCD(FALSE); //Dezaktywujemy kontroler wyświetlacza.

Funkcja lcdColor() służy do ustawienia aktualnego koloru tła i tekstu na wyświetlaczu LCD.

Składa się z instrukcji:

bkgColor = bkg; //Przypisuje wartość parametru bkg do zmiennej globalnej bkgColor.

textColor = text; //Przypisuje wartość parametru text do zmiennej globalnej textColor

Funkcja lcdContrast() umożliwia regulację kontrastu wyświetlacza LCD.

Składa się z instrukcji:

selectLCD(TRUE); //Aktywowacja kontrolera wyświetlacza

lcdWrcmd(LCD_CMD_SETCON); //Wysyła do kontrolera wyświetlacza komendę służącą do ustawienia kontrastu

lcdWrdata(cont); //Przesyła wartośc contr jako argument dla wcześniej wysłanej komendy

selectLCD(FALSE); //Deaktywowacja kontrolera wyświetlacza

Funkcja lcdRect() służy do rysowania prostokąta na wyświetlaczu LCD, wypełniając wybrany obszar jednolitym kolorem.

Parametry:

x (tU8): Współrzędna X lewego górnego rogu prostokąta.

y (tU8): Współrzędna Y lewego górnego rogu prostokąta.

xLen (tU8): Szerokość prostokąta.

yLen (tU8): Wysokość prostokąta.

color (tU8): Kolor, którym zostanie wypełniony prostokąt

Składa się z intstrukcji:

tU32 i;

tU32 len;

selectLCD(TRUE); //Aktywacja kontrolera wyświetlacza

lcdWindow1(x,y,x+xLen-1,y+yLen-1); //Definiuje obszar na wyświetlaczu, w którym będą realizowane operacje rysowania

lcdWrcmd(LCD_CMD_RAMWR); //Wysyła komendę rozpoczęcia zapisu do pamięci RAM wyświetlacza

//Wypełniamy każdy piksel w zdefiniowanym obszarze okna wybranym kolorem

len = xLen*yLen;

for(i=0; i<len; i++)

lcdWrdata(color);

selectLCD(FALSE); //Deaktywacja kontrolera wyświetlacza

Funkcja lcdRectBrd() służy do rysowania prostokąta na wyświetlaczu LCD, który ma różne kolory na krawędziach oraz inny kolor wypełnienia. Jest to rozszerzenie

funkcji lcdRect(), która pozwala również na rysowanie prostokąta, ale o różnych kolorach we wnętrzu i na obramowaniu.

Parametry:

x (tU8): Współrzędna X lewego górnego rogu prostokąta.

y (tU8): Współrzędna Y lewego górnego rogu prostokąta.

xLen (tU8): Szerokość prostokąta.

yLen (tU8): Wysokość prostokąta.

color1 (tU8): Kolor wewnętrzny prostokąta.

color2 (tU8): Kolor górnej i lewej krawędzi prostokąta.

color3 (tU6): Kolor dolnej i prawej krawędzi prostokąta.

Funkcja lcdIcon() służy do wyświetlania ikony na wyświetlaczu LCD, umożliwiając obsługę ikon w formacie skompresowanym lub nieskompresowanym.

Parametry:

x (tU8): Współrzędna X lewego górnego rogu obszaru, gdzie ikona ma zostać wyświetlona.

y (tU8): Współrzędna Y tego samego rogu.

xLen (tU8): Szerokość ikony w pikselach.

yLen (tU8): Wysokość ikony w pikselach.

compressionOn (tU8): Flaga określająca, czy dane ikony są skompresowane.

escapeChar (tU8): Znak ucieczki używany w skompresowanych danych do oznaczenia, że następujące bajty określają ilość powtórzeń piksela i jego kolor.

pData (const tU8*): Wskaźnik do tablicy danych ikony.

Składa się z instrukcji:

tU32 i,j;

tU32 len;

selectLCD(TRUE); //Aktywacja kontrolera wyświetlacza

lcdWindow1(x,y,x+xLen-1,y+yLen-1); //Definiowanie obszaru na wyświetlaczu, w którym będzie wyświetlana ikona

lcdWrcmd(LCD_CMD_RAMWR); //Wysłanie komendy do kontrolera LCD, aby rozpocząć proces zapisu danych do pamięci RAM wyświetlacza

len = xLen*yLen;

//Jeżeli compressionOn jest ustawione na FALSE, dane są traktowane jako ciąg pikseli, dane ikony są czytane i wyświetlane piksel po pikselu

//W przeciwnym wypadku, dane są przetwarzane jako ciąg zakodowanych pikseli, gdzie znak ucieczki (escapeChar) wskazuje na specjalną sekwencję kodowania:

//Gdy znaleziony jest znak ucieczki, następny bajt jest interpretowany jako ilość powtórzeń danego koloru, który jest odczytywany jako kolejny bajt. Ta sekwencja pozwala na efektywne kodowanie obszarów jednolitego koloru w ikonie.

```
if (compressionOn == FALSE)
{
 for(i=0; i<len; i++)
  lcdWrdata(*pData++);
}
else
 while(len > 0)
 {
  if(*pData == escapeChar)
  {
   pData++;
   j = *pData++;
   for(i=0;i<j;i++)
```

```
      lcdWrdata(*pData);
    pData++;
    len -= j;
  }
  else
  {
    lcdWrdata(*pData++);
    len--;
  }
 }
```

selectLCD(FALSE); //Deaktywacja kontrolera wyświetlacza

Funkcja lcdGotoxy() służy do ustawienia pozycji kursora na wyświetlaczu LCD w miejscu określonym przez współrzędne x i y. Pozycjonowanie kursora pozwala na odpowiednie wyświetlanie danych we wskazanych lokalizacjach.

Parametry:

x (tU8): Współrzędna x, określająca poziome umiejscowienie kursora na wyświetlaczu.

y (tU8): Współrzędna y, określająca pionowe umiejscowienie kursora na wyświetlaczu.

Składa się z instrukcji:

lcd_x = x;

lcd_y = y;

lcdWindow(x, y, 129, 129);

Funkcja lcdWindow() służy do ustawienia "okna" na wyświetlaczu LCD, które określa obszar, w którym możliwe będzie wyświetlanie danych.

Parametry:

xp (tU8): Współrzędna X początku okna.

yp (tU8): Współrzędna Y początku okna.

xe (tU8): Współrzędna X końca okna.

ye (tU8): Współrzędna Y końca okna.

Składa się z instrukcji:

selectLCD(TRUE); //aktywacja kontrolera wyświetlacza

lcdWindow1(xp, yp, xe, ye); //Wywoływanie w celu zdefiniowania granic okna, opisana jako następna funkcja

selectLCD(FALSE); //deaktywacja kontrolera wyświetlacza

Funkcja lcdWindow1() jest pomocniczą wersją funkcji lcdWindow(), używaną bezpośrednio do konfiguracji granic wyświetlania na wyświetlaczu LCD bez dodatkowego wywoływania aktywacji i dezaktywacji kontrolera LCD.

Parametry:

xp (tU8): Współrzędna x początku okna.

yp (tU8): Wge współrzędna y początku okna.

xe (tU8): Współrzędna x końca okna.

ye (tU8): Współrzędna y końca okna.

Składa się z instrukcji:

lcdWrcmd(LCD_CMD_CASET); //Wysyła komendę LCD_CMD_CASET (Column Address Set) do kontrolera LCD. Ta komenda pozwala na ustawienie zakresu kolumn, które będą aktywne podczas operacji zapisu i odczytu.

lcdWrdata(xp+2);

lcdWrdata(xe+2);

lcdWrcmd(LCD_CMD_PASET); //Wysyła komendę LCD_CMD_PASET (Page Address Set), która definiuje zakres wierszy (stron) aktywnych dla operacji wyświetlania.

lcdWrdata(yp+2);

lcdWrdata(ye+2);

//Dodajemy +2 do współrzędnych kolumn i wierszy, aby dopasować logiczne adresowanie pamięci RAM wyświetlacza do rzeczywistego, widocznego obszaru ekranu, który zaczyna się od współrzędnej (2, 2)

Funkcja lcdNewline() służy do aktualizacji pozycji kursora w taki sposób, aby przenieść go na początek nowej linii na wyświetlaczu LCD.

Składa się z instrukcji:

lcd_x = 0; //Ustawia kursor na początek linii, lewą krawędź ekranu

lcd_y += 14; //Przesuwa kursor w dół o wysokość jednego wiersza tekstu, który wynosi 14 pikseli

if (lcd_y >= 126)

lcd_y = 126;

//Sprawdzanie, czy wartość lcd_y nie przekracza granicy, która została ustalona na 126 pikseli. Jeśli lcd_y jest większe lub równe 126, zostaje ustawione na maksymalną wartość 126, aby uniknąć wyjścia kursora poza widoczny obszar ekranu.

Funkcja lcdData() służy do rysowania pojedynczego znaku na wyświetlaczu LCD, wykorzystując aktualnie ustawione kolory dla pierwszego planu i tła.

Parametry:

data (tU8): Kod ASCII znaku, który ma zostać wyświetlony.

Składa się z instrukcji:

selectLCD(TRUE); //aktywacja kontrolera wyświetlacza

if (data <= 127) //Sprawdzenie, czy wartość data mieści się w zakresie dozwolonych znaków ASCII (0-127)

{

tU32 mapOffset;

tU8 i,j,byteToShift;

data -= 30; //Odejmowanie wartości 30 od kodu ASCII znaku w funkcji, używane do dopasowania indeksowania do tablicy znaków zdefiniowanej w pliku ascii.h

mapOffset = 14*data; //Każdy znak ma wysokość 14 pixeli

lcdWrcmd(LCD_CMD_CASET); //Ustawienie zakresu kolumn

```
 lcdWrdata(lcd_x+2);
 lcdWrdata(lcd_x+9);
 lcdWrcmd(LCD_CMD_PASET); //Ustawienie zakresu wierszy
 lcdWrdata(lcd_y+2);
 lcdWrdata(lcd_y+15);
 lcdWrcmd(LCD_CMD_RAMWR); //Rozpoczęcie zapisu danych pikseli do pamięci RAM wyświetlacza

 for(i=0; i<14; i++) //iteracja po wierszach
 {
  byteToShift = charMap[mapOffset++]; //wartości wierszy znaku, o szerokości 8 pixeli, gdzie każdy bit oznacza czy dany pixel w wierszu to tekst czy tło
  for(j=0; j<8; j++) //Sprawdzamy bity w wartościach przypisanym każdemu wierszowi
  {
   if (byteToShift & 0x80) //Sprawdzamy, czy najstarszy bit jest ustawiony na 1
    lcdWrdata(textColor); //Bit ustawiony na 1: kolor tekstu
   else
    lcdWrdata(bkgColor);  //Bit ustawiony na 0: kolor tła
   byteToShift <<= 1; //Sprawdzamy kolejny bit poprzez przesunięcie bitowe
  }
 }
}

selectLCD(FALSE); //deaktywacja kontrolera wyświetlacza
lcd_x += 8; //Aktualizacja pozycji kursora, przesuwając go o 8 pikseli w prawo, co odpowiada szerokości jednego znaku
```

Funkcja lcdPutchar() jest używana do wyświetlania pojedynczego znaku na wyświetlaczu LCD. A także pozwalająca na kontrolowanie postępowania w przypadku wystąpenienia znaków specjalnych, takich jak nowa linia.

Parametry:

data (tU8): Znak (lub jego kod ASCII) do wyświetlenia na LCD.

Składa się z instrukcji:

if (data == '\n') //Jeśli przekazany znak to znak nowej linii, to funkcja wywołuje lcdNewline(), która resetuje pozycję kursora do początku następnej linii na wyświetlaczu

 lcdNewline();

else if (data != '\r') //Jeśli przekazany znak to znak powrotu karetki ('\r'), funkcja po prostu go ignoruje

{

 if (setcolmark == TRUE) //Obsługiwanie przypadku, w którym data wynosi 0xff. Jest to traktowane jako sygnał do ustawienia koloru tekstu. Po otrzymaniu takiego znaku, kolejny znak będzie interpretowany jako wartość koloru tekstu i przypisywany do zmiennej textColor.

 {

  textColor = data;

  setcolmark = FALSE;

 }

 else if (data == 0xff)

  setcolmark = TRUE;

 else if (lcd_x <= 124) //Jeśli znak nie jest specjalnym znakiem sterującym, funkcja sprawdza, czy pozycja lcd_x pozwala na umieszczenie znaku. Jeśli tak, wywoływana jest funkcja lcdData(), która zajmuje się rysowaniem znaku.

 {

  lcdData(data);

 }

}

Funkcja lcdPuts służy do wyświetlania ciągów znaków (łańcuchów tekstowych) na wyświetlaczu LCD, korzysta z funkcji lcdPutchar do obsługi pojedynczych znaków.

Parametry:

s (char*): Wskaźnik do łańcucha znaków, który ma zostać wyświetlony na LCD.

Składa się z instrukcji:

```
while(*s != '\0')
  lcdPutchar(*s++);
```

//Funkcja iteruje po znakach w ciągu, wyświetla je, aż nie osiągnie znaku końca (NUL). Podczas iterowania wyświetla znaki na ekranie korzystając z funkcji lcdPutchar.

Funkcje lcdWrcmd i lcdWrdata są używane do komunikacji z kontrolerem wyświetlacza LCD, pozwalającymi odpowiednio na wysyłanie komend i danych.

lcdWrcmd

Parametry:

cmd (tU8): 8-bitowa wartość komendy do wysłania do kontrolera LCD.

lcdWrdata

Parametry:

data (tU8): 8-bitowa wartość danych do wysłania do kontrolera LCD.

Obie funkcje korzystając z funkcji sendToLCD wysyłają do ekranu odpowiednio komendę lub dane, które zostały przekazane jako parametr funkcji.

Komendy używane w funkcjach i programie:

LCD_CMD_SWRESET (0x01) - Ta komenda powoduje zresetowanie kontrolera wyświetlacza do stanu początkowego, co jest równoznaczne z resetem sprzętowym.

LCD_CMD_BSTRON (0x03) - Komenda służy do włączenia napięcia wzmacniacza, co jest potrzebne do prawidłowego zasilania wyświetlacza.

LCD_CMD_SLEEPIN (0x10) - Wysyłając tę komendę, wyświetlacz zostanie przełączony w tryb uśpienia, co znacznie obniża zużycie energii poprzez wyłączenie niektórych funkcji.

LCD_CMD_SLEEPOUT (0x11) - Komenda przeciwna do SLEEPIN, służy do wybudzenia wyświetlacza z trybu uśpienia, przywracając jego normalne działanie.

LCD_CMD_INVON (0x21) - Ta komenda włącza inwersję kolorów na wyświetlaczu.

LCD_CMD_SETCON (0x25) - Komenda umożliwia ustawienie kontrastu wyświetlacza, co jest kluczowe dla zapewnienia odpowiedniej widoczności treści.

LCD_CMD_DISPON (0x29) - Służy do włączenia wyświetlacza, co jest wymagane po jego zainicjalizowaniu lub po wyjściu z trybu uśpienia.

LCD_CMD_CASET (0x2A) - Ustawia zakres adresów kolumn, co jest niezbędne do rysowania grafiki lub tekstu w określonych obszarach ekranu.

LCD_CMD_PASET (0x2B) - Podobnie jak CASET, ale dla zakresu adresów wierszy. Również używana do definiowania obszarów wyświetlania.

LCD_CMD_RAMWR (0x2C) - Rozpoczyna proces zapisu danych do pamięci RAM wyświetlacza, co jest używane do wyświetlania pikseli.

LCD_CMD_RGBSET (0x2D) - Umożliwia definiowanie palety kolorów, co jest używane przy mniejszej głębi kolorów, aby zdefiniować, jak poszczególne bity danych są interpretowane jako kolory.

LCD_CMD_MADCTL (0x36) - Służy do kontroli sposobu dostępu do pamięci, umożliwia zmianę kierunku skanowania ekranu, co może być przydatne do obracania obrazu na ekranie.

LCD_CMD_COLMOD (0x3A) - Ustawia format piksela interfejsu, czyli ile bitów danych jest potrzebnych do reprezentowania jakiegokolwiek koloru.

7. Techniczny opis działania Timera w programie

Timer jest kluczowym elementem podczas wykonywania operacji takich jak: pomiar czasu, generowanie opóźnień, generowanie sygnałów PWM i wiele innych. LPC2138 posiada dwa 32-bitowe timery ogólnego przeznaczenia (Timer0 i Timer1), które mogą być używane niezależnie.

Timer posiada rejestry:

Rejestr sterujący (TCR): Ten rejestr jest używany do kontroli działania timera. Posiada instrukcje, które pozwalają na włączanie i wyłączanie timera oraz resetowanie go.

PR (Prescale Register): Wartość tego rejestru określa, ile cykli zegara systemowego musi upłynąć, zanim licznik zostanie zwiększony o 1. Jest 32-bitową wartością liczbową.

Prescale Counter Register (PCR): Jest to 32-bitowy rejestr, który kontroluje podział PCLK (Primary Clock) przez pewną stałą wartość rejestru PR, zanim zostanie zastosowany do licznika timera. Licznik PCR jest zwiększany przy każdym PCLK. Kiedy osiągnie wartość przechowywaną w rejestrze PR, licznik timera jest zwiększany, a licznik

PCR jest resetowany przy następnym PCLK. Powoduje to zwiększenie TC (Timer Counter) przy każdym PCLK, gdy PR (Prescale Register) = 0, co 2 PCLK, gdy PR = 1, itd.

Rejestr licznika (TC): Zawiera aktualną wartość licznika timera. Jest 32-bitową wartością liczbową.

Rejestr przerwań (IR): Wskazuje, które zdarzenie dopasowania lub przechwycenia spowodowało przerwanie.

MR0-MR3 (Match Registers): Są to rejestry porównawcze. Gdy wartość w rejestrze licznika (TC) jest równa wartości w jednym z tych rejestrów, można wywołać przerwanie, zresetować i/lub zatrzymać licznik.

Capture Register (CR): Rejestr ten jest używany do przechwytywania i zapisywania wartości timera w momencie wystąpienia ustalonego wcześniej zewnętrznego zdarzenia.

Capture Control Register (CCR): Ten rejestr kontroluje, które zdarzenia na pinach przechwytywania mogą prowadzić do przechwycenia wartości timera do rejestru CR.

External Match Register (EMR): Rejestr ten kontroluje zachowanie pinów zewnętrznych w odpowiedzi na zdarzenia dopasowania wartości timera do wartości w rejestrze Match Register (MR).

MCR (Match Control Register): Ten rejestr kontroluje, co się dzieje, gdy wartość licznika (TC) pasuje do wartości w rejestrach MR0-MR3. Każdy bit w rejestrze MCR kontroluje inną funkcję. Odpiedni bit wartości MCR wywołuje przerwanie lub resetuje timer.

Zgodnie z tabelą:

Table 177: Match Control Register (MCR, TIMER0: T0MCR - address 0xE000 4014 and TIMER1: T1MCR - address 0xE000 8014) bit description

Bit	Symbol	Value	Description	Reset value
0	MR0I	1	Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC.	0
		0	This interrupt is disabled	
1	MR0R	1	Reset on MR0: the TC will be reset if MR0 matches it.	0
		0	Feature disabled.	
2	MR0S	1	Stop on MR0: the TC and PC will stop and TCR[0]=0 if MR0 matches the TC.	0
		0	Feature disabled.	
3	MR1I	1	Interrupt on MR1: an interrupt is generated when MR1 matches the value in the TC.	0
		0	This interrupt is disabled	
4	MR1R	1	Reset on MR1: the TC will be reset if MR1 matches it.	0
		0	Feature disabled.	
5	MR1S	1	Stop on MR1: the TC and PC will stop and TCR[0]=0 if MR1 matches the TC.	0
		0	Feature disabled.	
6	MR2I	1	Interrupt on MR2: an interrupt is generated when MR2 matches the value in the TC.	0
		0	This interrupt is disabled	
7	MR2R	1	Reset on MR2: the TC will be reset if MR2 matches it.	0
		0	Feature disabled.	
8	MR2S	1	Stop on MR2: the TC and PC will stop and TCR[0]=0 if MR2 matches the TC.	0
		0	Feature disabled.	
9	MR3I	1	Interrupt on MR3: an interrupt is generated when MR3 matches the value in the TC.	0
		0	This interrupt is disabled	
10	MR3R	1	Reset on MR3: the TC will be reset if MR3 matches it.	0
		0	Feature disabled.	
11	MR3S	1	Stop on MR3: the TC and PC will stop and TCR[0]=0 if MR3 matches the TC.	0
		0	Feature disabled.	
15:12	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Przykładowo, aby wygenerować przerwanie wykonywane stale co 5 minut:

PLCK naszego układu wynosi 58982400Hz.

5 minut = 300 sekund

Zdefiniujmy stałe:
#define SYSTEM_CLOCK_FREQUENCY 58982400
#define PRESCALER_VALUE 58982399
#define INTERRUPT_TIME_SECONDS 300

uint32_t match_register_value = (SYSTEM_CLOCK_FREQUENCY * INTERRUPT_TIME_SECONDS) / (PRESCALER_VALUE + 1);
Dzięki powyższemu wzorowi jesteśmy w stanie ustalić odpowiednią wartość MR0 dla dowolnej wartości PR.

Ustawmy rejestry:
T0TCR = 0x02; //Zatrzymuje i resetuje timer
T0PR = PRESCALER_VALUE;
T0MR0 = match_register_value;

Ustawmy rejestr MCR zgodnie z wcześniej pokazaną tabelą:
T0MCR = 0x3
Dla tej wartości, przy zgodności TC z MR0 dla Timera 0 zostanie wywołane przerwanie oraz licznik w rejestrze TC zostanie zresetowany, co pozwoli na kolejne odliczanie do 5 minut. Zaś licznik będzie zwiększany o 1 co każdą sekundę.

Na koniec możemy z powrotem aktywować timer:
T0TCR = 0x01;

8. Techniczny opis działania I^2C w programie

Interfejs I2C (Inter-Integrated Circuit) jest dwukierunkowowym szeregowym interfejsem komunikacyjnym, który umożliwia komunikację pomiędzy różnymi układami scalonymi w systemie, za pomocą dwóch linii: SDA (Serial Data Line) i SCL (Serial Clock Line).

W naszym programie interfejs I^2C został wykorzystany do komunikacji z potencjometrem oraz do komunikacji z pamięcią EEPROM. Dzięki potencjometrowi możemy ustawić poziom trudności rozgrywki. Zaś z pomocą EEPROM jesteśmy w stanie zapisywać i odczytywać najlepszy wynik.

Magistrala I2C dostępna na naszej płytce oferuje następujące funkcje:

1. Możliwość konfiguracji jako Master, Slave, lub Master/Slave.
2. Arbitraż między równocześnie transmitującymi masterami bez uszkodzenia danych szeregowych na magistrali.
3. Programowalne zegary umożliwiające dostosowanie prędkości transmisji I2C.
4. Dwukierunkowy transfer danych między masterami a sługami.
5. Synchronizacja zegara szeregowego umożliwia komunikację urządzeń o różnych szybkościach bitowych przez jedną magistralę szeregową.

Interfejsy I2C mogą funkcjonować w jednym z czterech trybów:

1. Tryb nadawcy-mastera (Master Transmitter Mode),

2. Tryb odbiornika-mastera (Master Receiver Mode),
3. Tryb nadawcy-slave'a (Slave Transmitter Mode),
4. Tryb odbiornika-slave'a (Slave Receiver Mode).

W każdym z trybów, I2C może inicjować transfer danych, przyjmować dane, a także generować odpowiednie sygnały START i STOP dla zarządzania komunikacją.

Rejestry I^2C:

Rejestr Adresowy (I2ADDR): jest kluczowym elementem protokołu komunikacji I^2C. Ten rejestr może być załadowany 7-bitowym adresem slave (7 najbardziej znaczących bitów), do którego blok I^2C będzie reagować, gdy zostanie zaprogramowany jako nadajnik lub odbiornik slave. Najmniej znaczący bit (LSB) jest używany do włączania rozpoznawania ogólnego adresu wywołania (0x00).

I2CONSET/I2CONCLR: Rejestry te są używane do ustawiania lub czyszczenia poszczególnych bitów w rejestrze kontrolnym I2C.

I2CONSET (I2C Control Set Register): Umożliwia ustawienie bitów kontrolnych I2C, takich jak:

1. AA (Assert Acknowledge Flag)
2. SI (I2C Interrupt Flag)
3. STO (STOP Flag)
4. STA (START Flag)
5. I2EN (I2C Interface Enable)

I2CONCLR (I2C Control Clear Register): Służy do czyszczenia wybranych bitów rejestru kontrolnego I2C, takich jak:

1. AAC (Acknowledge Clear)
2. SIC (Interrupt Clear)
3. STOC (STOP Clear)
4. STAC (START Clear)
5. I2ENC (I2C Interface Disable)

I2STAT: Rejestr statusu, który dostarcza szczegółowe kody statusów podczas operacji I2C. Zawiera kody statusu, które wskazują na różne stany magistrali I2C, takie jak START wysłany, danych otrzymanych, danych wysłanych, oraz inne stany.

I2DAT: Rejestr danych używany do transmisji lub odbioru danych. Wpisywanie do tego rejestru inicjuje transmisję danych na magistrali, a odczyt z tego rejestru daje dostęp do danych otrzymanych od innego urządzenia na magistrali I2C.

I2SCLH (I2C SCL Duty Cycle High Register): Określa ustawienia czasu trwania stanu wysokiego zegara SCL, co ma wpływ na prędkość transmisji na magistrali. Większa wartość w tym rejestrze oznacza dłuższy czas trwania stanu wysokiego zegara SCL, co skutkuje wolniejszą prędkością transmisji.

I2SCLL (I2C SCL Duty Cycle Low Register): Określa ustawienia czasu trwania stanu niskiego zegara SCL. Zwiększenie wartości tego rejestru spowalnia transmisję, ponieważ zegar SCL spędza więcej czasu w stanie niskim.

Prędkość magistrali I2C można obliczyć na podstawie wartości zapisanych w rejestrach I2SCLH i I2SCLL oraz częstotliwości zegara PCLK. Przykładowo, jeśli PCLK (tak jak w naszej płytce) wynosi 58982400Hz, a wartości w rejestrach I2SCLH i I2SCLL wynoszą odpowiednio 300 i 300, to prędkość magistrali I2C będzie obliczana następująco:

$$I^2C_{frequency} = \frac{PCLK}{I2CSCLH + I2CSCLL}$$

$$I^2C_{frequency} = \frac{58982400Hz}{300 + 300} = 98304Hz$$

W trybie Master, urządzenie inicjuje i kontroluje komunikację na magistrali I2C, wysyłając sygnały startu i stopu oraz zegar. Urządzenie master jest odpowiedzialne za ustanowienie zegara i sekwencji sterujących na magistrali.

W trybie Slave, urządzenie reaguje na sygnały sterujące generowane przez urządzenie master. Urządzenie slave oczekuje na adresację od mastera, a następnie bierze udział w transmisji danych, o ile zostało poprawnie zaadresowane.

Urządzenie w trybie slave reaguje na określony adres. Po otrzymaniu adresu, slave sprawdza, czy zgadza się on z jego ustawionym adresem. Jeśli tak, to odpowiada sygnałem ACK (acknowledge), w przeciwnym razie ignoruje komunikację.

Komunikacja na magistrali I2C obejmuje procedury potwierdzenia (ACK) i braku potwierdzenia (NACK), które są kluczowe dla zarządzania przepływem danych między urządzeniami.

ACK (Potwierdzenie): W trybie komunikacji po każdym przesłanym bajcie danych, odbierające urządzenie (master lub slave) wysyła bit ACK, w celu potwierdzenia poprawności przesłania bajtu.

NACK (Brak potwierdzenia): Jeśli urządzenie odbierające nie jest w stanie przyjąć więcej danych lub nie chce więcej danych, wysyła NACK. NACK jest również używane do informowania nadawcy o zakończeniu transmisji.

Aby zainicjalizować I2C i móc dowolnie konfigurować i korzystać z I2C należy użyć funkcji:

```
void i2cInit(void)
{
  PINSEL0  |= 0x50; // Funkcja konfiguruje piny P0.2 i P0.3 do używania ich jako linii I2C poprzez ustawienie odpowiednich wartości w rejestrze PINSEL0.

  I2C_CONCLR = 0x6c; //Czyści flagi w rejestrze sterującym I2C, dokładniej resetuje bity AA, SI, STA, I2EN w rejestrze I2CONSET

  I2C_SCLL   = ( I2C_SCLL   & ~I2C_REG_SCLL_MASK )   | I2C_REG_SCLL; //Resetuje rejestr I2C_SCLL, który kontroluje czas trwania impulsu zegarowego SCL w trybie niskim.
  I2C_SCLH   = ( I2C_SCLH   & ~I2C_REG_SCLH_MASK )   | I2C_REG_SCLH; //Resetuje rejestr I2C_SCLH, który kontroluje czas trwania impulsu zegarowego SCL w trybie wysokim.
  I2C_ADDR   = ( I2C_ADDR   & ~I2C_REG_ADDR_MASK )   | I2C_REG_ADDR; //Ustawia adres slave'a I2C w rejestrze I2C_ADDR
  I2C_CONSET = ( I2C_CONSET & ~I2C_REG_CONSET_MASK ) | I2C_REG_CONSET; //Aktywuje interfejs I2C poprzez ustawienie odpowiednich bitów
}
```

Adresy rejestrów:

I2C_CONSET - 0xE001C000

I2C_STAT - 0xE001C004

I2C_DATA - 0xE001C008

I2C_ADDR - 0xE001C00C

I2C_SCLH - 0xE001C010

I2C_SCLL - 0xE001C014

I2C_CONCLR - 0xE001C018

Wartości rejestrów i maski:

I2C_REG_CONSET 0x00000040 - Wartość Control Set Register

I2C_REG_CONSET_MASK 0x0000007C - Używane bity

I2C_REG_DATA 0x00000000 - Wartość Data register

I2C_REG_DATA_MASK 0x000000FF - Używane bity

I2C_REG_ADDR 0x00000000 - Wartość Slave address register

I2C_REG_ADDR_MASK 0x000000FF - Używane bity

I2C_REG_SCLH 0x00000100 - Wartość SCL Duty Cycle high register

I2C_REG_SCLH_MASK 0x0000FFFF - Używane bity

I2C_REG_SCLL 0x00000100 - Wartość SCL Duty Cycle low register

I2C_REG_SCLL_MASK 0x0000FFFF - Używane bity

9. Techniczny opis działania EEPROM w programie

W programie korzystamy z pamięci EEPROM w celu możliwości zapisywania i odczytywania najlepszego uzyskanego wyniku, gdzie jest on przechowywany jako dane nieulotne. Pamięć EEPROM sterujemy poprzez wykorzystanie interfejsu I2C. Pamięć dostępna na naszej płytce jest w stanie pomieścić do 2kb danych, co jest dla nas całkowicie wystarczające. Aby móc odczytywać i zapisywać dane z/do pamięci należy pierw zainicjalizować I2C (opisane w rozdziale I2C).

Aby zapisać wynik do pamięci korzystamy z funkcji eepromWrite. Pozwala na zapisanie ciągu bajtów o określonej długości pod określony adres w pamięci EEPROM.

Parametry:

tU16 addr: Adres początkowy w pamięci EEPROM, gdzie dane mają być zapisane.

tU8* pData: Wskaźnik na dane, które mają być zapisane w EEPROM.

tU16 len: Liczba bajtów do zapisania.

Typ zwracany:

tS8: Kod statusu operacji zapisu. Zwraca I2C_CODE_OK (0) jeśli zapis się powiódł, I2C_CODE_ERROR (-1) w przypadku wystąpienia błędu.

Składa się z instrukcji:

```
tS8 retCode = 0;

tU8 i      = 0;


do

{


  /* Funkcja rozpoczyna od wygenerowania warunku startu na magistrali I2C przy użyciu i2cStart(). Jeśli generacja warunku startu się nie powiedzie, funkcja natychmiast kończy działanie, zwracając kod błędu. */

  retCode = i2cStart();

  if(retCode != I2C_CODE_OK)

    break;



  /* Następnie funkcja wysyła adres urządzenia EEPROM na magistrali I2C, który jest wynikiem operacji I2C_EEPROM_ADDR, używając funkcji i2cWriteWithWait(). Ta funkcja zawiera również obsługę oczekiwania na zwolnienie magistrali (I2C_CODE_BUSY). W przypadku niepowodzenia w tym kroku, funkcja zwraca błąd i kończy działanie. */

  retCode = i2cWriteWithWait(I2C_EEPROM_ADDR);

  if(retCode != I2C_CODE_OK)

    break;


  /* Wysyła młodszy bajt adresu, który wskazuje dokładne miejsce, w którym dane mają zostać zapisane. Jeśli wystąpi błąd, pętla jest przerywana.  */

  retCode = i2cWriteWithWait( (tU8)(addr & 0xFF));
```

```
 if(retCode != I2C_CODE_OK)
   break;
```

```
 /* W pętli, dla każdego bajtu danych wskazywanego przez pData, funkcja próbuje zapisać bajt danych do EEPROM poprzez kolejne wywołania i2cWriteWithWait(). Jeśli w dowolnym momencie zapisu danych wystąpi błąd, pętla jest przerywana i funkcja zwraca błąd. */
 for(i = 0; i <len; i++)
 {
   retCode = i2cWriteWithWait(*pData);
   if(retCode != I2C_CODE_OK)
     break;

   pData++;
 }

} while(0);

/* Niezależnie od tego, czy operacja zapisu się powiodła, czy też wystąpił błąd, funkcja kończy operację na magistrali I2C przez wygenerowanie warunku stopu. */
i2cStop();

return retCode;
```

Aby odczytać wynik z pamięci korzystamy z funkcji eepromPageRead. Umożliwia odczytanie określonej ilości bajtów danych zaczynając od wskazanego adresu.

Parametry:

tU16 address: Adres początkowy w EEPROM, od którego rozpocznie się odczyt danych.

tU8 pBuf*: Wskaźnik na bufor, do którego będą wczytywane dane.

tU16 len: Liczba bajtów danych do odczytania.

Typ zwracany:

tS8: Funkcja zwraca kod statusu operacji.

Składa się z instrukcji:

```
tS8  retCode = 0; // Zmienna na kod wyniku operacji
tU8  status  = 0; // Zmienna na status transmisji I2C
tU16 i       = 0; // Licznik do iteracji przez dane
```

/* Funkcja jest wywoływana z adresem urządzenia EEPROM (I2C_EEPROM_ADDR) i adresem początkowym (address), gdzie zacznie się odczyt. Funkcja ta inicjuje transmisję I2C, wysyłając adres urządzenia oraz adres początkowy danych do odczytu. */

```
retCode = eepromStartRead(I2C_EEPROM_ADDR, address);
```

if(retCode == I2C_CODE_OK) //Jeśli inicjacja transmisji zakończyła się sukcesem, rozpoczyna się pętla odczytu danych.

```
{
 /* Pętla for iteruje po bajtach do odczytu, aż nie osiągnie wymaganej długości */
 for(i = 1; i <= len; i++ )
 {
  /* Oczekiwanie wewnątrz pętli while na zakończenie transmisji każdego bajtu. */
  while(1)
  {
   /* Sprawdza aktualny status magistrali I2C */
   status = i2cCheckStatus();
```

```
if(( status == 0x40 ) || ( status == 0x48 ) || ( status == 0x50 ))
{
```

/* Jeśli jest to ostatni bajt do odczytu, wysyła NACK (I2C_MODE_ACK1), w przeciwnym razie wysyła ACK (I2C_MODE_ACK0). Funkcja i2cGetChar odbiera bajt z magistrali I2C i zapisuje go w buforze pBuf. */

```
if(i == len )
{
 /* Wysyła NACK */
 retCode = i2cGetChar( I2C_MODE_ACK1, pBuf );
}
else
{
        /* Wysyła ACK */
 retCode = i2cGetChar( I2C_MODE_ACK0, pBuf );
}
```

/* Dane są odczytywane do bufora wskazywanego przez pBuf. Jeżeli i2cGetChar zwróci I2C_CODE_EMPTY, co oznacza, że bufor odczytu jest pusty, funkcja czeka aż dane będą dostępne. Po odczycie pBuf jest inkrementowany, aby wskazywał na następną pozycję w buforze. */

```
retCode = i2cGetChar( I2C_MODE_READ, pBuf );
while( retCode == I2C_CODE_EMPTY )
{
 retCode = i2cGetChar( I2C_MODE_READ, pBuf );
}
pBuf++;

break;
}
```

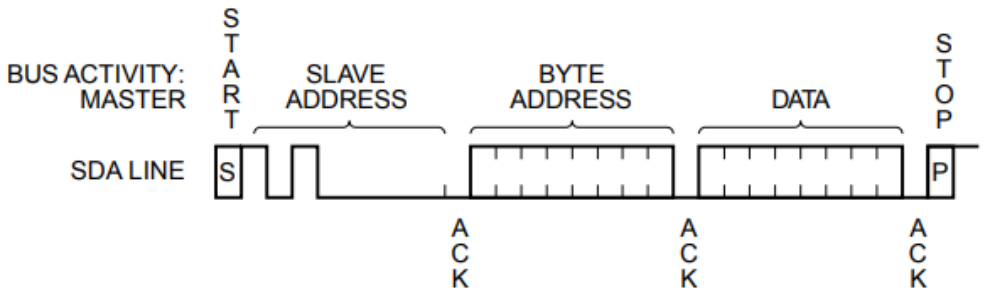
```
    else if( status != 0xf8 )
    {
      /* Jeśli status nie jest żadnym z oczekiwanych, to zakłada się wystąpienie błędu, przerywa pętlę i ustawia retCode na I2C_CODE_ERROR. */
      i = len;
      retCode = I2C_CODE_ERROR;
      break;
    }
   }
  }
}
```

/* Niezależnie od tego, czy operacja odczytu się powiodła, czy też wystąpił błąd, funkcja kończy operację na magistrali I2C przez wygenerowanie warunku stopu. */

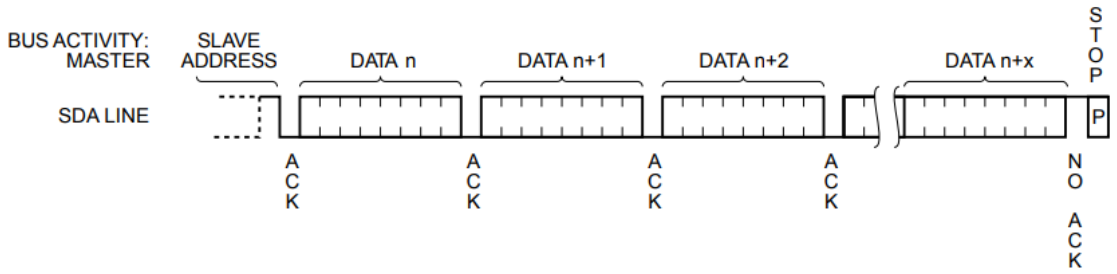
```
i2cStop();

return retCode;
```

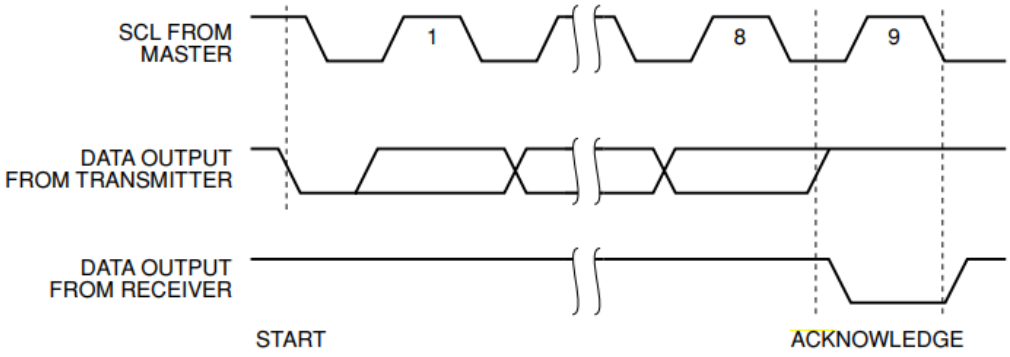
Rysunek przedstawiający kolejność instrukcji podczas zapisu danych do pamięci:



Rysunek przedstawiający kolejność instrukcji podczas odczytu danych z pamięci:



Po pomyślnym przesłaniu danych każde urządzenie odbierające jest zobowiązane do wygenerowania potwierdzenia. Urządzenie potwierdzające ustawia niski sygnał na linii SDA podczas dziewiątego cyklu zegara, sygnalizując, że odebrało 8 bitów danych.



10. Techniczny opis działania Potencjometru w programie

Potencjometr w LPC2138 wykorzystuje przetwornik analogowo-cyfrowy (ADC) do odczytu wartości napięcia, które jest następnie przetwarzane na wartość cyfrową. LPC2138 posiada dwa przetworniki ADC: ADC0 i ADC1.

Odczytaną wartość wykorzystujemy do ustawienia poziomu trudności rozgrywki, to znaczy, w jakim czasie ma nastąpić wciśnięcie odpowiedniego przycisku.

W przypadku naszej płytki, ADC jest 10-bitowy, co oznacza, że wartości zwracane przez ADC mieszczą się w zakresie od 0 do 1023.

Potencjometr zwraca wartości z zakresu [0; 1023] w zależności od położenia pokrętła.

Rejestry ADC:

Rejestr Kontrolny ADC (AD0CR, AD1CR) - Ten rejestr jest wykorzystywany do konfiguracji pracy przetwornika ADC. Umożliwia wybór kanałów, które mają być aktywne podczas próbkowania i konwersji, ustalenie dzielnika zegara APB dla zapewnienia odpowiedniej częstotliwości zegara ADC, kontrolę trybu burst (ciągłe konwersje), wybór liczby cykli zegara potrzebnych na każdą konwersję, zarządzanie rozpoczęciem konwersji, określenie zbocza sygnału rozpoczynającego konwersję oraz przełączanie przetwornika ADC pomiędzy trybem operacyjnym a trybem uśpienia.

Globalny Rejestr Danych (AD0GDR, AD1GDR) - Zawiera wynik ostatniej konwersji przeprowadzonej przez przetwornik. Rejestr ten również informuje, czy konwersja się zakończyła i czy doszło do przepełnienia, co oznacza, że nowe dane zostały zapisane zanim zdążono odczytać poprzednie. Ponadto wskazuje, z którego kanału pochodzi ostatni wynik konwersji.

Rejestr Statusu ADC (AD0STAT, AD1STAT) - Podaje informacje o statusie wszystkich kanałów przetwornika ADC, w tym czy konwersja dla danego kanału została zakończona oraz czy wystąpiło przepełnienie. Dodatkowo, rejestr ten zawiera flagę przerwania ADC, która jest ustawiana, gdy konwersja na dowolnym kanale zostanie zakończona.

Rejestr Rozpoczęcia Konwersji (ADGSR) - Umożliwia jednoczesne rozpoczęcie konwersji na obu przetwornikach ADC. Jest to użyteczne w aplikacjach wymagających synchronizacji konwersji między różnymi kanałami ADC.

Rejestr Przerwań ADC (AD0INTEN, AD1INTEN) - Pozwala na włączenie lub wyłączenie generowania przerwań przez poszczególne kanały ADC po zakończeniu konwersji. Umożliwia on włączenie lub wyłączenie przerwań na koniec konwersji dla każdego kanału oddzielnie.

Rejestry Danych Kanałów (AD0DR0-AD0DR7, AD1DR0-AD1DR7) - Każdy kanał ADC ma przypisany osobny rejestr danych, który przechowuje wynik ostatniej konwersji dla tego kanału. Rejestry te zawierają również informacje o zakończeniu konwersji i ewentualnym przepełnieniu danych.

Funkcja initAdc jest odpowiedzialna za inicjalizację przetwornika analogowo-cyfrowego (ADC) w programie. Proces inicjalizacji obejmuje konfigurację pinów wejściowych oraz ustawienia samego przetwornika.

Składa się z instrukcji:

volatile tU32 integerResult;

//Inicjalizacja ADC: AIN1 na pinie P0.28

PINSEL1 &= ~0x03000000; //Czyści bity 24 i 25 w rejestrze PINSEL1, co przygotowuje pin P0.28 do konfiguracji jako wejście analogowe.

PINSEL1 |= 0x01000000; //Ustawia bit 24 w rejestrze PINSEL1, co konfiguruje pin P0.28 jako wejście analogowe AIN1.

//Inicjalizacja ADC

ADCR = (1 << 0) | //SEL = 1, ustawia bit 0 w rejestrze ADCR, co wybiera kanał 0 jako kanał wejściowy.

((CRYSTAL_FREQUENCY *

PLL_FACTOR /

VPBDIV_FACTOR) / 4500000 - 1) << 8 | //oblicza wartość dzielnika zegara ADC, aby uzyskać częstotliwość zegara ADC równą 4,5 MHz. Wynik obliczenia jest przesunięty o 8 bitów w lewo i ustawiany w rejestrze ADCR.

(0 << 16) | //BURST = 0, ustawia bit 16 na 0 w rejestrze ADCR, co oznacza, że konwersje ADC będą kontrolowane programowo (tryb burst wyłączony).

(0 << 17) | //CLKS = 0, ustawia bit 17 na 0 w rejestrze ADCR, co oznacza, że przetwornik ADC będzie używał 11 cykli zegara na konwersję, co daje 10-bitowy wynik.

(1 << 21) | //PDN = 1, ustawia bit 21 na 1 w rejestrze ADCR, co włącza przetwornik ADC.

(1 << 24) | //START = 1, ustawia bit 24 na 1 w rejestrze ADCR, co powoduje natychmiastowe rozpoczęcie konwersji.

(0 << 27); //EDGE = 0, ustawia bit 27 na 0 w rejestrze ADCR, co określa zbocze sygnału startowego jako narastające. Jest to nieistotne, gdy START = 1.

//Wprowadza krótkie opóźnienie (1 milisekunda), aby dać czas przetwornikowi ADC na wprowadzenie ustawień.

mdelay(1);

integerResult = ADDR; //Wykonuje odczyt próbny z rejestru danych ADC.

Funkcja getAnalogueInput jest odpowiedzialna za przeprowadzenie konwersji analogowo-cyfrowej (ADC) na wybranym kanale wejściowym i zwrócenie wyniku tej konwersji jako 10-bitową wartość cyfrową. Służy do odczytu wartości potencjometru.

Parametry:

tU8 channel: Określa numer kanału analogowego wejściowego, który ma zostać przetworzony przez przetwornik analogowo-cyfrowy (ADC). Numer kanału powinien być w zakresie od 0 do maksymalnej liczby dostępnych kanałów ADC, w naszym przypadku od 0 do 6, czyli AIN0 do AIN6.

Składa się z instrukcji:

tU16 returnResult; //zmienna używana do przechowywania wyniku konwersji ADC

//Ustawia odpowiednie bity w rejestrze kontrolnym ADC (ADCR), a następnie rozpoczyna konwersję na wybranym kanale

ADCR = (ADCR & 0xFFFFFF00) | (1 << channel) | (1 << 24);

//(ADCR & 0xFFFFFF00) - czyszczenie dolnych 8 bitów rejestru ADCR, aby przygotować miejsce na nowe ustawienia kanałów.

//(1 << channel) - ustawienie bitu odpowiadającego wybranemu kanałowi, podanemu jako argument funkcji.

//(1 << 24) - ustawienie bitu 24, co inicjuje natychmiastowe rozpoczęcie konwersji.

//Oczekiwanie na zakończenie konwersji

while((ADDR & 0x80000000) == 0)

;

//Oczekiwanie, aż bit 31 w rejestrze danych ADC (ADDR) zostanie ustawiony na 1. Bit ten (DONE) wskazuje, czy konwersja została zakończona. Dopóki bit 31 jest równy 0, pętla będzie się powtarzać i oczekiwać na zakończenie konwersji.

//Pobranie wyniku i dostosowanie do 10-bitowego formatu

returnResult = (ADDR>>6) & 0x3FF;

//(ADDR >> 6) - przesunięcie wyniku o 6 bitów w prawo, aby usunąć niepotrzebne bity.

//& 0x3FF - maskowanie wyniku, aby uzyskać tylko 10 bitów (0x3FF to nasza 10 bitowa maska).

return returnResult;

11. Techniczny opis działania Obsługi przerwań w programie

IRQ (Interrupt Request) to sygnały żądania przerwania generowane przez różne źródła, natomiast VIC (Vectored Interrupt Controller) to kontroler przerwań, który zarządza przerwaniami poprzez priorytetyzację, przypisywanie przerwań do wektorów lub nie oraz automatyczne zarządzanie flagami przerwań, co pozwala na szybką i efektywną obsługę przerwań w systemie. W naszym programie IRQ oraz VIC jest używane do generowania ziarna dla gry, dzięki czemu zyskujemy "losowość" wyboru diody. Dzieje się to poprzez okresowe wywoływanie przerwań, kiedy to wywołujemy funkcję aktualizującą wartość zmiennej, z której pobieramy wartość do przypisania wyboru diody.

Rejestry VIC:

VICIRQStatus: Pokazuje, które przerwania są obecnie obsługiwane jako przerwania IRQ. Każdy bit rejestru odpowiada przerwaniu jednego urządzenia, przy czym ustawiony bit oznacza aktywne przerwanie. Rejestr tylko do odczytu.

VICFIQStatus: Podobnie jak VICIRQStatus, ale dla przerwań typu FIQ. Ustawione bity wskazują na aktywne przerwania FIQ. Rejestr tylko do odczytu.

VICRawIntr: Zawiera nieprzetworzone stany żądań przerwań, pokazując bieżący stan przerwań bez uwzględnienia ich klasyfikacji oraz czy są włączone. Rejestr tylko do odczytu.

VICIntSelect: Umożliwia wybór, czy dane źródło przerwania generuje przerwanie typu IRQ, czy FIQ. Bit ustawiony na '1' oznacza przerwanie FIQ. '0' oznacza IRQ.

VICIntEnable: Rejestr włączania przerwań, gdzie każdy bit odpowiada jednemu źródłu przerwania. Ustawienie bitu na '1' włącza obsługę danego przerwania.

VICIntEnClear: Umożliwia wyłączenie przerwań. Wpis '1' na odpowiedniej pozycji wyłącza dane przerwanie. Rejestr tylko do zapisu.

VICSoftInt: Umożliwia programowe wywołanie przerwania. Ustawienie bitu generuje przerwanie danego typu.

VICSoftIntClear: Służy do czyszczenia przerwań programowych ustawionych przez VICSoftInt.

VICProtection: Rejestr kontroli dostępu, określa, czy dostęp do rejestrów VIC jest chroniony. Ustawiając wartość rejestru na '1' dostęp można uzyskać tylko w trybie uprzywilejowanym.

VICVectAddr0-15: Zawiera adresy procedur obsługi przerwań. Gniazdo 0 ma najwyższy priorytet, a gniazdo 15 najniższy.

VICVectCntl0-15: Służy do kontrolowania priorytetu i aktywacji przerwań wektorowych, umożliwiając przypisanie przerwań do konkretnych wektorów i ustawienie ich priorytetów. Gniazdo 0 ma najwyższy priorytet, a gniazdo 15 najniższy.

VICDefVectAddr: Jest to domyślny rejestr adresów wektorowych. Zawiera on adres procedury obsługi przerwań dla przerwań nie wektorowych.

Funkcja inicjująca okresową obsługę przerwań składa się z parametrów określających okres wywoływanych przerwań oraz współczynnik wypełnienia.

```
static void init_irq (tU32 period, tU8 duty_cycle)
{
        //Zainicjuj VIC dla przerwań od Timera #1
  VICIntSelect &= ~TIMER_1_IRQ;        //Przerwanie od Timera #1 przypisane do IRQ (nie do FIQ)
  VICVectAddr5  = (tU32)IRQ_Test;      //adres procedury przerwania
  VICVectCntl5  = VIC_ENABLE_SLOT | TIMER_1_IRQ_NO;
  VICIntEnable  = TIMER_1_IRQ;         // Przypisanie i odblokowanie slotu w VIC od Timera #1

  T1TCR = TIMER_RESET;                 //Zatrzymaj i zresetuj
  T1PR  = 0;                           //Preskaler nieużywany
  T1MR0 = ((tU64)period)*((tU64)PERIPHERAL_CLOCK)/1000;
  T1MR1 = (tU64)T1MR0 * duty_cycle / 100; //Wypełnienie
  T1IR  = TIMER_ALL_INT;               //Resetowanie flag przerwań
  T1MCR = MR0_I | MR1_I | MR0_R;       //Generuj okresowe przerwania dla MR0 i dodatkowo dla MR1
```

```
   T1TCR = TIMER_RUN;                  //Uruchom timer
}
```

Dokładne wyjaśnienie elementów:

1. Konfiguracja VIC dla Timera #1:

VICIntSelect &= ~TIMER_1_IRQ; - Ta instrukcja zapewnia, że przerwanie generowane przez Timer #1 będzie traktowane jako przerwanie typu IRQ (Interrupt Request), a nie FIQ (Fast Interrupt Request). Bit odpowiedzialny za Timer #1 jest negowany, co oznacza przypisanie do IRQ.

VICVectAddr5 = (tU32)IRQ_Test; - Przypisuje adres procedury obsługi przerwań, która ma być wykonana, gdy wystąpi przerwanie od Timera #1. W tym przypadku, funkcja IRQ_Test będzie wywołana, aby zaktualizować ziarno.

VICVectCntl5 = VIC_ENABLE_SLOT | TIMER_1_IRQ_NO; - Konfiguruje slot VIC (nr 5) do obsługi przerwania od Timera #1. Slot jest aktywowany i przypisany do konkretnego numeru przerwania (TIMER_1_IRQ_NO).

VICIntEnable = TIMER_1_IRQ; - Aktywuje możliwość przerwań od Timera #1 poprzez ustawienie odpowiedniego bitu w rejestrze VICIntEnable. Tym samym przerwania mogą być obsługiwane przez system.

2. Konfiguracja Timera #1:

T1TCR = TIMER_RESET; - Zatrzymuje działanie timera i resetuje go.

T1PR = 0; - Ustawia wartość preskalera na 0, co oznacza, że timer będzie inkrementowany z szybkością podaną przez zegar peryferyjny (58982400Hz).

T1MR0 = ((tU64)period)*((tU64)PERIPHERAL_CLOCK)/1000; - Ustawia wartość, po której osiągnięciu timer wygeneruje przerwanie (MR0 - Match Register 0). Wartość ta zależy od zadanego okresu i szybkości zegara peryferyjnego oraz jest podana w milisekundach.

T1MR1 = (tU64)T1MR0 * duty_cycle / 100; - Ustawia wypełnienie timera. Timer generuje dodatkowe przerwanie, kiedy osiągnie wartość T1MR1.

T1IR = TIMER_ALL_INT; - Kasuje wszystkie flagi przerwań dla Timera #1.

T1MCR = MR0_I | MR1_I | MR0_R; - Konfiguruje Timer Match Control Register, ustawiając generowanie przerwań na MR0 i MR1 oraz restart timera po osiągnięciu wartości MR0.

T1TCR = TIMER_RUN; - Wznawia działanie timera, który zaczyna odliczanie od zera.

12. Analiza potencjalnych skutków awarii

Moduł	Szansa	Skutki Awarii	Wykrycie(koszt)	Reakcja(koszt)	Iloczyn
GPIO	0.4	Krytyczne(10)	1	4	16
EKRAN LCD	0.4	Krytyczne(10)	2	4	32
TIMER	0.1	Krytyczne(10)	4	8	32
I^2C	0.1	Niegroźne(1)	4	8	3,2
SPI	0.1	Krytyczne(10)	4	8	32
EEPROM	0.3	Niegroźne(1)	4	5	6
OBSŁUGA PRZERWAŃ (IRQ)	0.1	Średnie(5)	3	8	12
POTENCJOMETR	0.5	Niegroźne(1)	2	5	5

Opis i Analiza

GPIO:

Możliwy tryb awarii: Brak reakcji na wejścia/wyjścia.

Skutki awarii: Utrata funkcjonalności przycisków i LED, co może uniemożliwić interakcję z użytkownikiem.

Przyczyny awarii: Uszkodzenie portu, błędna konfiguracja pinów, przeciążenie.

Sposoby wykrycia usterek i środki zapobiegawcze: Monitorowanie stanu GPIO w czasie rzeczywistym, wykonywanie autotestów przy starcie systemu.

SPI:

Możliwy tryb awarii: Nieprawidłowe przesyłanie danych lub brak takiej możliwości.

Skutki awarii: Błędna komunikacja z urządzeniami peryferyjnymi, co może prowadzić do nieprawidłowego działania systemu, w naszym programie doprowadzi to do usterek związanych z działaniem ekranu.

Przyczyny awarii: Błędna konfiguracja, przeciążenie, uszkodzenie.

Sposoby wykrycia usterek i środki zapobiegawcze: Regularne testy funkcjonalne, testy integralności danych przesyłanych przez SPI, zaimplementowanie mechanizmów chroniących przed przeciążeniem, limitujących łącze, sprawdzanie poprawności wyświetlania informacji.

EKRAN LCD:

Możliwy tryb awarii: Brak wyświetlania lub błędne wyświetlanie informacji.

Skutki awarii: Brak lub niepoprawne wyświetlanie informacji dla użytkownika, co skutkuje zabużoną komunikacją.

Przyczyny awarii: Błędna konfiguracja, uszkodzenie ekranu, uszkodzenie SPI.

Sposoby wykrycia usterek i środki zapobiegawcze: Sprawdzanie poprawności wyświetlania informacji, wykonywanie autotestów wyświetlacza LCD i SPI.

TIMER:

Możliwy tryb awarii: Nieprawidłowe odliczanie czasu.

Skutki awarii: Błędne działanie funkcji zależnych od czasu, co może następnie spowodować nieprawidłowe wysyłanie żądań przerwań.

Przyczyny awarii: Błędna konfiguracja, uszkodzenie timera.

Sposoby wykrycia usterek i środki zapobiegawcze: Testy dokładności timerów, porównanie wyników timerów, wyniku jednego timera drugim oraz wyników z wzorcowym, obliczonym wynikiem, regularne monitorowanie dokładności odmierzania czasu.

I^2C:

Możliwy tryb awarii: Brak komunikacji z urządzeniami na magistrali.

Skutki awarii: Utrata funkcjonalności EEPROM, uniemożliwia zapisywanie i odczytywanie najlepszego wyniku gry, jak i też utrata uzyskiwania wartości z potencjometru.

Przyczyny awarii: Błędna konfiguracja, zakłócenia, przeciążenia, uszkodzenie.

Sposoby wykrycia usterek i środki zapobiegawcze: Regularne testowanie poprawności połączeń I^2C poprzez diagnostyczne odczyty i zapisy z/do urządzeń peryferyjnych.

EEPROM:

Możliwy tryb awarii: Błędny zapis/odczyt danych.

Skutki awarii: Utrata danych lub ich uszkodzenie.

Przyczyny awarii: Błędna komunikacja I^2C, uszkodzenie EEPROM.

Sposoby wykrycia usterek i środki zapobiegawcze: Weryfikacja danych za pomocą sum kontrolnych (CRC), regularne testy integralności danych zapisanych w EEPROM oraz stosowanie różnych kodów korekcyjnych, chroniących w pewnym stopniu przed uszkodzeniem danych.

POTENCJOMETR:

Możliwy tryb awarii: Niemożliwość odczytu wartości lub odczyt niepoprawnej wartości.

Skutki awarii: Niemożność wyboru poziomu trudności.

Przyczyny awarii: Uszkodzenie potencjometru lub I^2C, błędna konfiguracja przetwornika ADC.

Sposoby wykrycia usterek i środki zapobiegawcze: Monitorowanie odczytów ADC w czasie rzeczywistym, wykonywanie autotestów potencjometru.

OBSŁUGA PRZERWAŃ:

Możliwy tryb awarii: Brak reakcji na zdarzenia zewnętrzne, co skutkuje brakiem obsługi żądań przerwań.

Skutki awarii: Błędne działanie funkcji wywołującej okresowo funkcję aktualizującą ziarno.

Przyczyny awarii: Błędna konfiguracja przerwań, przeciążenie systemu, uszkodzenie timera lub VIC.

Sposoby wykrycia usterek i środki zapobiegawcze: Monitorowanie zmiany ziarna w czasie, weryfikacja działania timera.