styczeń 2018

Stacja pogodowa
Dokumentacja projektu

Zajęcia: poniedziałek 10:15

Grupa: B13

Spis treści

1. Podstawowe informacje

Skład zespołu:	**Nr indeksu:**	**Adres e-mail:**	**Wkład pracy:**
Rafał Plinzner (lider)	210293	210293@edu.p.lodz.pl	42%
Michał Białecki	210141	210141@edu.p.lodz.pl	42%
Łukasz Zysiak	210362	210362@edu.p.lodz.pl	16%

Ciężko jest jednoznacznie określić kto jest odpowiedzialny za który kawałek kodu, gdyż pracowaliśmy wspólnie nad praktycznie każdą z funkcjonalności. Zdecydowaliśmy się więc na wpisanie procentowego udziału w realizacji zadania.

1.1. Sprzęt użyty do realizacji projektu

1. Aktualnie wykorzystywany sprzęt dostępny w laboratorium:
    - Eduboard LPC2148 wersja 3
    - płytka rozszerzająca LPC2148 (Experiment Expansion Board)
2. Własny sprzęt zewnętrzny:
    - czujnik wilgotności i temperatury DHT11

1.2. Zakres projektu

Założona funkcjonalność	Stopień zaawansowania	Osoba odpowiedzialna
Obsługa dwuwierszowego wyświetlacza LCD	Działa	Rafał Plinzner
Obsługa termometru znajdującego się na płytce	Działa	Łukasz Zysiak
Obsługa termometru i higroskopu zewnętrznego	Działa	Michał Białecki
Obsługa pamięci EEPROM	Działa	Łukasz Zysiak
Obsługa RTC (Real Time Clock)	Działa	Michał Białecki
Obsługa Timera	Działa	Rafał Plinzner
Obsługa magistrali I2C	Działa	Michał Białecki
Obsługa przerwań	Działa	Rafał Plinzner
Obsługa silnika krokowego	Działa	Rafał Plinzner
Obsługa indywidualnego interfejsu czujnika DHT11	Działa	Michał Białecki
Obsługa GPIO	Działa	Rafał Plinzner

2. Instrukcja użytkownika

Program pozwala na odczyt temperatury z dwóch czujników: jednego znajdującego się wewnątrz pomieszczenia oraz drugiego – na zewnątrz. Ponadto drugi czujnik pozwala na pomiar wilgotności na zewnątrz. Wszystkie odczytywane z czujników wartości wyświetlane są na dwuwierszowym wyświetlaczu LCD. Jako że wszystkie informacje nie zmieściłyby się na ekranie, zaimplementowane zostały trzy tryby wyświetlana:

1. Pierwszy tryb – wyświetlanie czasu
2. Drugi tryb – wyświetlanie temperatur: wewnątrz i na zewnątrz
3. Trzeci tryb – wyświetlanie wartości wilgotności

Pomiędzy powyższymi trybami pracy wyświetlacza, przełączamy się za pomocą przytrzymania przycisku. Ponadto w przypadku, gdy temperatura wewnątrz pomieszczenia przekroczy temperaturę 28 stopni Celsjusza, uruchamia się klimatyzacja.

3. Funkcjonalności

3.1. Obsługa wyświetlacza LCD

Wyświetlacz LCD zamontowany na płytce Eduboard LPC 2148 v3 obsługiwany jest za pomocą pinów P1.16-P1.23 oraz P1.24, P0.22, P1.25 i P0.30.

Poszczególne piny są odpowiedzialne za:

P1.16-P1.23	Odpowiedzialne za obsługę bitów danych w wyświetlaczu (DATA) DB0- DB7
P1.24	Bit wyjścia RS
P0.22	Bit zapisu/odczytu RW
P1.25	Bit kontrolny E
P0.30	Bit kontrolujący podświetlenie wyświetlacza (Backlight)

Aby wydawać wyświetlaczowi komendy takie jak czyszczenie ekranu, czy ustawianie kursora należy ustawić RS na stan niski (IOCLR1 = 0x01000000), a następnie wysłać do rejestru odpowiedni kod. Aby móc wysyłać dane do wyświetlacza należy ustawić stan RS na wysoki (IOSET1 = 0x01000000) i wysłać do rejestru wartość liczbową odpowiadającą znakowi w formacie ASCII, który chcemy wyświetlić.

Dostępne komendy wyświetlacza zostały przedstawione w tabelce:

Code (Hex)	Command to LCD Instruction Register
1	Clear Display screen
2	Return home
4	Decrement cursor (Shift cursor to left)
6	Increment cursor (Shift cursor to Right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display on, cursor off
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning of 1st line
0C0	Force cursor to beginning of 2nd line
38	2 lines and 5x7 Matrix www.embetronicx.com

Rysunek 1 Tabelka komend dla wyświetlacza
Źródło: embetronicx.com

Uwaga! Ponieważ każda z operacji wykonywanych na wyświetlaczu ma pewien czas realizacji, zastosowaliśmy pętlę:

```
for (i = 0; i < 6 * 2500; i++)
        asm volatile (" nop");
```

Aby móc wejść w interakcje z wyświetlaczem musimy odpowiednio skonfigurować piny odpowiedzialne za wysyłanie danych.

Zrobimy to za pomocą interfejsu GPIO (General Purpose Input/Output):

Uwaga! Domyślnie wszystkie piny są ustawione jako GPIO. Dzięki temu nie musimy za każdym razem wpisywać zer do rejestrów PINSELx.

1. Ustawiamy piny P1.16-P1.23, P1.25 oraz P1.24 jako wyjście wpisując je do rejestru IODIR1
   IODIR1 |= (0x00ff0000 | 0x02000000 | 0x01000000)
2. Na tych samych pinach ustawiamy stan niski
3. Ustawiamy pin P0.22 jako wyjście wpisując go do rejestru IODIR1
   IODIR0 |= 0x00400000
4. Również na nim ustawiamy stan niski
5. Ostatnim krokiem jest zapewnienie możliwości sterowania podświetleniem wyświetlacza. W tym celu ustawiamy pin P0.30 jako wyjście wpisując go do rejestru IODIR0
   IODIR0 |= 0x40000000
6. Na nim również ustawiamy stan niski

Proces inicjacji wyświetlacza został zaimplementowany następująco:

Uwaga! Przed wysłaniem komendy ustawiamy stan RS na niski.

1. Wysyłając komendę 0x38 ustawiamy tryb wyświetlacza
2. Wysyłając komendę 0x08 i wyłączamy wyświetlacz
3. Wysyłając komendę 0x01 czyścimy wyświetlacz z jakichkolwiek śmieci
4. Wysyłając komendę 0x06 ustawiamy tryb wprowadzania na inkrementację
5. Wysyłając komendę 0x0C włączamy wyświetlacz
6. Na końcu komendą 0x02 ustawiamy kursor na początek

3.1.1. Obsługa termometru LM75

Rysunek 2 Czujnik temperatury LM75
źródło: elcodis.com

Specyfikacja techniczna czujnika LM75:

• Zakres pomiaru temperatury:	-55°C - +125°C	
• Dokładność pomiaru temperatury:	± 2°C	dla -25°C - +100°C
	± 3°C	dla -55°C - -25°C
		dla 100°C - 125°C
• Obsługiwane zasilanie:	2.8V - 5.5V	
• Liczba pinów:	8	

Temperaturę z czujnika LM75 odczytujemy za pomocą magistrali I2C i pamięci EEPROM.

Implementacja w programie:

1. Oczekujemy na wysłanie danych z magistrali I2C.

```
if (retCode == I2C_CODE_OK) {
    for (i = 1; i <= len; i++)
```

2. Sprawdzamy stan magistrali I2C i przypisujemy go do zmiennej *status.*

```
status = i2cCheckStatus()
```

3. Wysyłamy bit NACK jeśli wszystkie dane zostały odebrane, w przeciwnym wypadku zostanie wysłany bit ACK, wykorzystujemy do tego zmienną *retCode*.

```
if (i == len) {
        retCode = i2cGetChar(I2C_MODE_ACK1, pBuf);
{ else {
        retCode = i2cGetChar(I2C_MODE_ACK0, pBuf);
}
```

4. Ustawiamy pamięć EEPROM na tryb odczytu.

```
retCode = i2cGetChar(I2C_MODE_READ, pBuf);
```

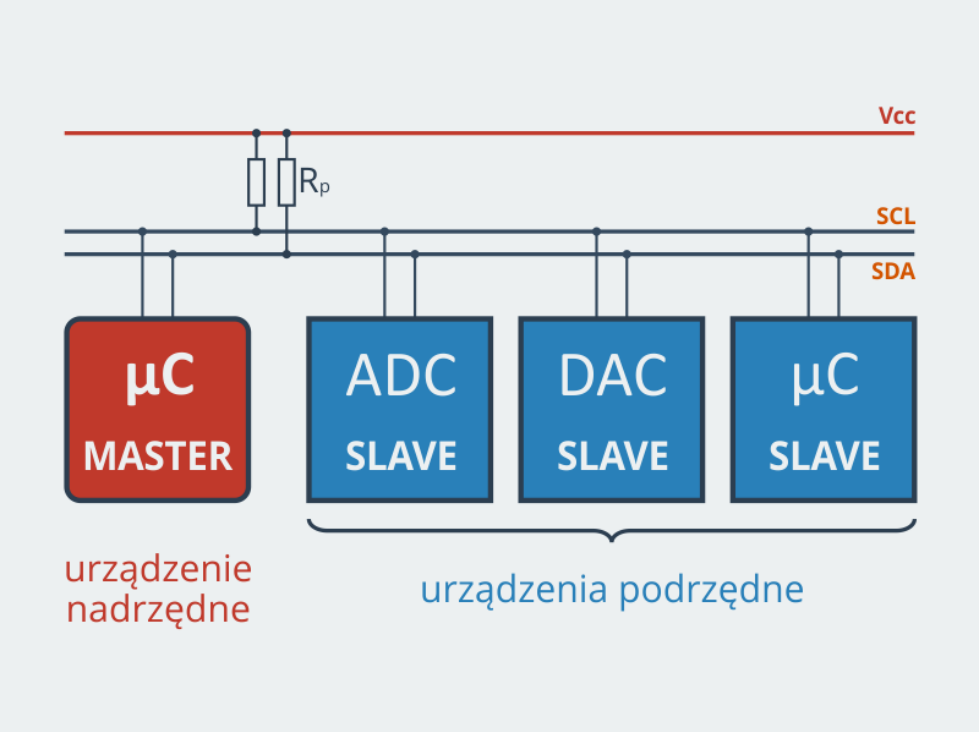
5. Następnie odczytujemy dane i wypisujemy na ekran LCD płytki w poprawnym formacie.

3.2. Obsługa magistrali I2C

Magistrala I2C do transmisji wykorzystuje dwie dwukierunkowe linie:

- SDA – linia danych
- SCL – linia zegara

I2C używa logiki dodatniej, a więc stan niski na magistrali odpowiada „0” logicznemu, natomiast stan wysoki „1” logicznej.



Rysunek 3 Magistrala I2C
źródło: forbot.pl

Komunikacja przez I^2C wygląda następująco:

1. Układ nadrzędny rozpoczyna komunikację poprzez wystawienie sygnału **START** – jest to specjalna kombinacja przebiegów na liniach **SDA** i **SCL** informująca o początku transmisji.
2. Następnie wysyła adres układu, z którym chce się komunikować oraz bit informujący o kierunku transmisji (nadawanie albo odbiór). Układ podrzędny odpowiada wysyłając bit potwierdzenia ACK.
3. Po wybraniu układu, następuje właściwa transmisja:
    1. jeśli zostało wybrane nadawanie, *master* wysyła kolejne bajty, a *slave* potwierdza odebranie każdego bitem ACK
    2. w przypadku odbioru, *slave* wysyła kolejne bajty, a master odbiera i potwierdza za pomocą ACK
4. Na koniec master wysyła sygnał **STOP** informujący o zakończeniu transmisji

Na początku należy zainicjować magistralę:

```
PINSEL0 |= 0x50;
Ustawienie pinów SDA i SCL jako output
I2C_CONCLR = 0x6c;
Czyszczenie flag
I2C_SCLL = (I2C_SCLL & ~I2C_REG_SCLL_MASK) | I2C_REG_SCLL;
I2C_SCLH = (I2C_SCLH & ~I2C_REG_SCLH_MASK) | I2C_REG_SCLH;
I2C_ADDR = (I2C_ADDR & ~I2C_REG_ADDR_MASK) | I2C_REG_ADDR;
I2C_CONSET = (I2C_CONSET & ~I2C_REG_CONSET_MASK) | I2C_REG_CONSET;
Reset rejestrów
```

Następnie w celu uruchomienia komunikacji:
I2C_CONSET |= 0x20;
Ustawienie flagi startowej na 1
if ((status == 0x08) || (status == 0x10))
Sprawdzenie czy magistrala ma odpowiedni status tzn. czy komunikat START został przesłany, jeśli tak to można przystąpić do transmisji, jeśli nie to zwracany jest odpowiedni kod błędu (0xf8).

Kolejnym krokiem jest sprawdzenie statusu przez funkcję odpowiadającą za transmisję danych z danego urządzenia. Jeśli status jest odpowiedni następuje wysłanie adresu urządzenia.

```
if ((I2C_CONSET & 0x08) != 0)  //sprawdzenie czy SI = 1 (czy mozna uzyskac dostep do rejestru danych)
{
  I2C_DATA = data;          //wyslane znaku
  I2C_CONCLR = 0x08;
 retCode = I2C_CODE_OK;   //powodzenie wyslania
```

Następnie sprawdzany jest stan flagi SI, czyli czy magistrala jest obecnie wolna. Jeśli tak następuje transmisja danych, czyli wpisanie znaku do rejestru I2C_DATA. Po wysłaniu znaku, czyszczona jest flaga SI i wywoływana informacja o zwolnieniu się magistrali.

Kolejnym krokiem jest odbiór danych

```
I2C_CONSET |= 0x04; //ustawienie ACK=0 (informacja dla slave aby ten wyslal kolejny bajt)
I2C_CONCLR = 0x08;  //czyszczenie flagi przerwan
```

Jeśli chcemy odebrać dane z urządzenia, należy ustawić tryb urządzenia master na przyjmowanie danych

```
I2C_CONCLR = 0x04;
I2C_CONCLR = 0x08;
```

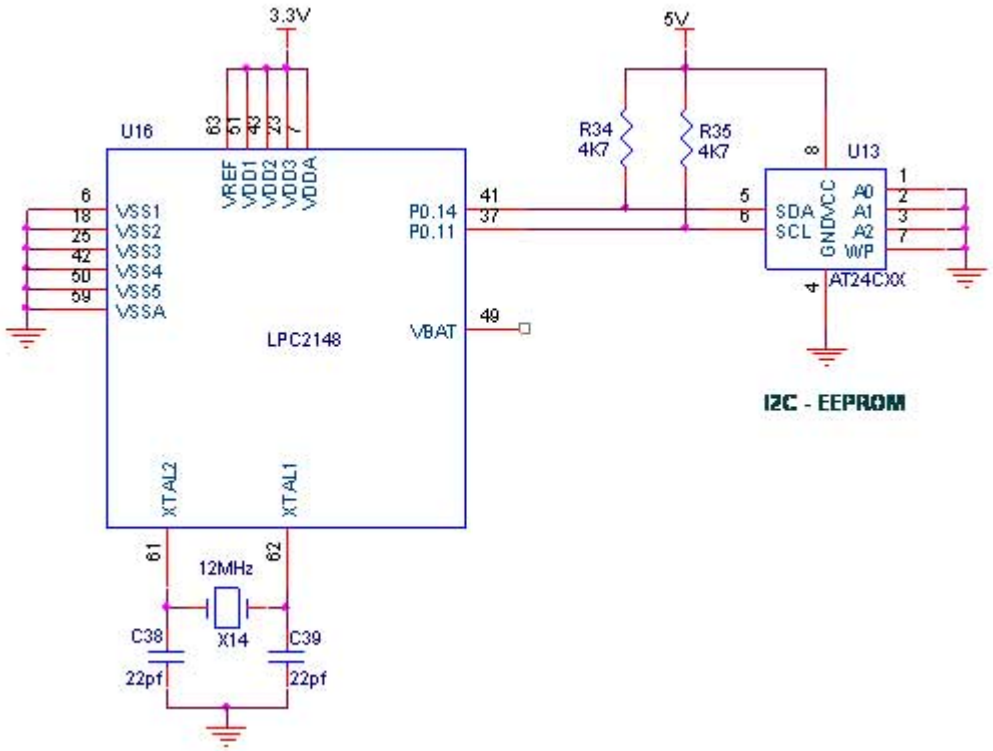
Wysłanie potwierdzenia ACK, które jest informacją dla urządzenia slave aby wysłało ostatni bajt danych.

```
*pData = (tU8) I2C_DATA
```

Ostatnim krokiem jest odczytanie danych z rejestru I2C_DATA

3.3. Obsługa pamięci EEPROM

Pamięć EEPROM jest to pamięć nieulotna o pojemności 128B. Producent deklaruje 1 000 000 cykli zapisu i kasowania oraz nieograniczoną liczbę odczytów pamięci.



Rysunek 4 Diagram komunikacji I2C-EEPROM na płytce LPC2148
źródło: pantechsolutions.net

Na płytce Eduboard LPC 2148 v3 linie pamięci EEPROM są kontrolowane przez magistralę I2C.Tak więc operacje odczytu i zapisu pamięci EEPROM są odbywane przy użyciu linii SDA i SCL magistrali I2C.

Podczas zapisu danych powinniśmy odwołać się do adresu 0xA0, a odczytywać z 0xA1.

Aby odczytać dane z pamięci EEPROM należy skonfigurować pin P0.11 oraz P0.14 do obsługi linii SCL i SDA magistrali I2C.

Następnie należy uruchomić procedurę startu magistrali I2C. Wysłać sygnał SLA+R oraz oczekiwać na odbiór danych. Sprawdzamy status I2C, jeśli dane zostały odebrane generujemy bit NACK. W przeciwnym razie generowany jest bit ACK.

```
retCode = i2cGetChar(I2C_MODE_ACK1, pBuf);

else

retCode = i2cGetChar(I2C_MODE_ACK0, pBuf);
```

I przełączamy się w tryb odczytywania danych.

```
retCode = i2cGetChar(I2C_MODE_READ, pBuf);
```

Na koniec należy zatrzymać transmisję.

Zapis odbywa się podobnie. W tym przypadku potwierdzeniem jest bit ACK. Następnie wysyłamy adres komórki pamięci, do której chcemy zapisać. Pamięć ponownie potwierdza odbiór danych za pomocą bitu ACK.

Następnie wysyłamy jedną lub więcej wartości, każda jest potwierdzana za pomocą ACK. Na koniec wysyłamy **STOP**, aby zakończyć transmisję i umożliwić pamięci rzeczywisty zapis danych.

3.4. Obsługa termometru i higroskopu zewnętrznego DHT11

Zewnętrzny czujnik wilgotności i temperatury powietrza podłączyliśmy do pinu XP0.4, uziemienie oraz zasilanie (+3,3V) na płytce Expansion Board.

Rysunek 5 Czujnik DHT11
źródło: botland.com.pl

Specyfikacja techniczna czujnika DHT11:

- Zakres pomiaru wilgotności powietrza: 20% - 90%
- Zakres pomiaru temperatury: 0°C - 50°C
- Dokładność pomiaru wilgotności powietrza: ± 5% przy 25°C
- Dokładność pomiaru temperatury: ± 2°C
- Obsługiwane zasilanie: 3.3V - 5.5V
- Liczba pinów: 4 (1 nieużywany)

Linia danych w czujniku jest podpięta do napięcia za pomocą rezystora podciągającego o oporze równym 4.7 kΩ. Pozwala to utrzymać określony stan logiczny na linii danych.

Temperaturę oraz wilgotności powietrza z zewnętrznego czujnika DHT11 odczytujemy za pomocą indywidualnego interfejsu dla tego czujnika. Szczegółowe informacje o sposobie działania przedstawiamy w punkcie dotyczącym obsługi tego interfejsu (3.5).

3.5. Obsługa indywidualnego interfejsu czujnika DHT11

Działanie interfejsu czujnika DHT11 składa się z kilku etapów:

1. Jako że na linii danych łączącej MCU z czujnikiem utrzymuje się stan wysoki, aby rozpocząć odbiór danych MCU ustawia na linii stan niski, a tym samym wysyła sygnał „start" trwający co najmniej 18 milisekund.

   ```
   #define DHT_PIN 4
   (IODIR0 |= (1<<DHT_PIN));
   ```
   Ustawienie pinu 0.4 jako wyjście (output)
   ```
   (IOCLR0 = (1<<DHT_PIN));
   delay_ms(18);
   ```
   Utrzymanie stanu niskiego przez 18 milisekund – sygnał dla czujnika
   ```
   (IOSET0 = (1<<DHT_PIN));
   Przywrócenie stanu wysokiego
   ```

2. Po wysłaniu sygnału przez MCU następuje przerwa (czyli stan wysoki) trwający od 20 do 40 mikrosekund, po której czujnik (który odebrał komunikat „start") wysyła sygnał potwierdzając w ten sposób gotowość do transmisji danych.

   ```
   (IODIR0 &= ~(1<<DHT_PIN));
   ```
   Ustawienie pinu 0.4 jako wejście (input) - przygotowanie do odbioru danych

   ```
   while (IOPIN0 & 0x00000010);
   ```
   Oczekiwanie na stan niski na linii, czyli sygnał o gotowości urządzenia do transmisji

   ```
   while ((IOPIN0 & 0x00000010) == 0) ;
   ```
   Oczekiwanie na stan wysoki

   ```
   while (IOPIN0 & 0x00000010);
   ```
   Oczekiwanie na stan niski, czyli zakończenie sygnału od urządzenia (powinno nastąpić po co najmniej 80 mikrosekundach).

3. Teraz następuje właściwa transmisja danych z czujnika. Poniższa pętla posłuży do odbioru jednego bajtu danych.

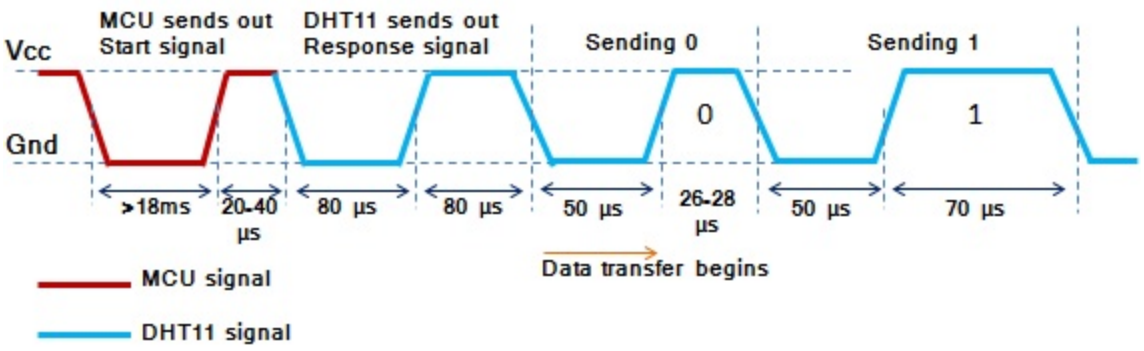
   ```
    for (count = 0; count < 8; count++)    /* 8 bits of data */
   {
   while ((IOPIN0 & 0x00000010) == 0)    /* Wait till response is LOW */
   ```
   Oczekiwanie na stan wysoki, czyli rozpoczęcie transmisji pojedynczego bitu
   ```
   delay_us(30);    /* delay greater than 24 usec */
    if (IOPIN0 & 0x00000010) /* If response is HIGH, 1 is received */
   data = ((data << 1) | 0x01);
   ```
   Jeśli stan wysoki nadal trwa po odczekaniu 30 mikrosekund, oznacza to, że wysłana została jedynka.
   ```
   else    /* If response is LOW, 0 is received */
   data = (data << 1);
   ```
   Jeśli po 30 mikrosekundach na linii ustawiony jest stan niski, oznacza to, że odebrane zostało zero.
   ```
   while (IOPIN0 & 0x00000010) ;
   ```

Oczekiwanie na zakończenie stanu wysokiego, czyli transmisji bitu z czujnika.

Z każdą kolejną iteracją pętli, odbierany jest jeden bit danych. Każda informacja wysyłana jest przez czujnik w postaci stanu wysokiego. Aby odpowiednio zinterpretować odbierany bit (czy jest to 0 czy 1) należy sprawdzić długość jego trwania (za zero odpowiada stan wysoki trwający ok. 24-28 mikrosekund, natomiast jedynkę reprezentuje stan wysoki trwający ok. 70 mikrosekund).

W implementacji obsługi tego czujnika, wzięliśmy pod uwagę możliwość jego awarii. Dlatego każda pętla while została wzbogacona o zmienną, która inkrementuje z każdym wykonaniem pętli. Jeśli zmienna ta osiągnie wartość 1000000 (co oznacza, że zbyt długo oczekiwano na odpowiedź czujnika) pętla zostaje zatrzymana.

```
err++;
if (err >= 1000000) return 0;
```



Rysunek 6 Schemat interfejsu czujnika DHT11
źródło: embedded-lab.com

3.6. Obsługa RTC (Real Time Clock)

Aby obsłużyć moduł RTC, niezbędne jest jego zainicjalizowanie:

```
CCR  = 0x00000012;
CCR  = 0x00000010;
```

CCR - licznik "tyknięć zegara" przedostatni bit ustawiony na 1 resetuje licznik, a ostatni bit uruchamia timer counters

```
ILR  = 0x00000000;
```

Interrupt Location Register - ustawienie dwóch ostatnich bitów jako 0 dezaktywuje funkcje przerwań RTC

```
CIIR = 0x00000000;
```

Counter Increment Interrupt Register – wszystkie bity ustawione na zero dezaktywują generowanie przerwań za każdym razem, gdy licznik jest inkrementowany

```
AMR  = 0x00000000;
```
Alarm Mask Register - wprowadzenie wartości 0 dezaktywuje funkcję alarmu

```
delay_us(500000);
(…)
CCR  = 0x00000011;
```
ustawienie częstotliwości rezonatora kwarcowego na 32.768 KHz

Następnie do rejestrów RTC wprowadzone zostały:

-godzina (rejestr HOUR)
-minuta (rejestr MINx)
-sekunda (rejestr SEC)

```
SEC  = 0;    //rejestr sekund
MINx  = 25;   //rejestr minut
HOUR = 15;   //rejestr godzin
```

Za każdym uruchomieniem programu zegar będzie "wskazywał" wprowadzone do rejestrów wartości i wraz upływającym czasem działania będzie te wartości aktualizował (czyli z każdą kolejną sekundą zmienia się wartość rejestru SEC, po osiągnięciu 60 sekund wartość rejestru MIN zwiększy się o 1 itd.).

Czas odmierzany przez RTC, wyświetlany jest na dwuwierszowym ekranie LCD.

3.7. Obsługa Timera

Timer jest niezbędnym urządzeniem do poprawnego funkcjonowania naszego programu. Przykładowo czujnik DHT11, potrzebuje bardzo precyzyjnych wartości do prawidłowego działania i my musimy je zapewnić.

Aby ułatwić korzystanie z timera zaimplementowaliśmy funkcje z odpowiednimi jednostkami czasu: sekundy, milisekundy i nanosekundy.

Na płytce LPC2148 znajdują się 2 timery.

Dla celów implementacji opóźnień służy nam timer 1.

Timer posiada licznik (Timer Counter – TC) oraz rejestr preskalera (Prescale Register – PR) z nim powiązany. Gdy resetujemy i uruchamiamy timer TC ustawiane jest na 0, a następnie zwiększane co 1 co każdy „PR+1” cykl zegara. Dzięki preskalerowi możemy uzyskać zwiększanie rejestru TC co określoną jednostkę (np. cykl zegara peryferyjnego - peripheral clock , milisekundę, sekundę). W ten sposób jeśli preskaler mamy ustawiony na 0 to, TC inkrementowane jest co 1 cykl zegara peryferyjnego. Jeśli ustawimy natomiast preskaler na 1, wtedy TC inkrementuje się co 2 cykle zegara, itd.

Timer posiada również cztery 32-bitowe rejestry dopasowania oraz cztery 32-bitowe rejestry przechwycenia.

Rejestry dopasowania mogą być wykorzystywane np. do: zatrzymywania timera przy zrównaniu wartości (np. gdy wartość w rejestrze TC jest taka sama jak w rejestrze dopasowania), zresetowaniu

timera i wywołania opcjonalnego przerwania, do liczenia ciągłego i wywołania przerwania przy zrównaniu wartości.

Poniżej omówiłem podstawowe rejestry potrzebne do obsługi timera:

- **PR** – Rejestr preskalera (Prescale Register) (32 bit) – Przechowuje maksymalną wartość licznika preskalera, po której licznik ten się resetuje
- **PC –** Rejestr licznika preskalera (Prescale Counter Register) (32 bit) – Rejestr ten inkrementuje się co cykl zegara peryferyjnego. Ten rejestr kontroluje „skalę, jednostkę" timera. Gdy **PC** dotrze do wartości z **PR**, resetowany jest z powrotem do 0, a **TC** (licznik timera) zwiększany jest o 1
- **TC –** Rejestr licznika timera (Timer Counter Register) (32 bit) – główny rejestr liczący. Inkrementuje się, gdy **PC** osiągnie maksimum określone w **PR.** Jeśli nie zostanie zatrzymany lub zresetowany, to po osiągnięciu maksymalnej wartości (0xFFFFFFFF) przeskoczy z powrotem do 0
- **TCR –** Rejestr kontroli timera (Timer Control Register) – rejestr używany do włączania, wyłączania i resetowania TC. Gdy bit0 jest ustawiony na 1 timer jest włączony, natomiast gdy jest 0 – wyłączony. Gdy bit1 jest ustawiony na 1 to **TC** oraz **PC** ustawiane są na 0.
- **CTCR** – Rejestr kontroli liczenia (Count Control register) – używany do wybrania trybu: zegara lub licznika. Dla naszych zastosowań korzystać będziemy z trybu zegara. Jest on włączony gdy wartość CTCR jest ustawiona na 0
- **MCR** – Rejestr kontroli dopasowania (Match Control register) – rejestr używany do określenia jakie operacje mają być wykonane po zrównaniu wartości (gdy wartość w TC będzie taka sama jak wpisana w MR – rejestrze dopasowania). I tak dla MR0:
    - Bit 0 – przerwanie na MR0 – np. wywołuje przerwanie gdy w MR0 jest taka sama wartość jak w TC. Przerwanie jest włączone gdy ustawiona jest jedynka, a wyłączone gdy zero,
    - Bit 1 – reset na MR0. Gdy ustawione na 1, TC zresetuje się, gdy osiągnie wartość jak w MR0. Wyłączone gdy ustawione na 0
    - Bit 2 – Stop na MR0. Gdy ustawione na 1 TC oraz PC zatrzymają się gdy TC zrówna się z MR0

      Uwaga! podobnie bity 3-5, 6-8, 9-11 są odpowiednio dla MR1, MR2, MR3
- **IR** - Rejestr przerwań (Interrupt Register) – zawiera flagi przerwań dla 4 dopasowań i 4 przechwyceń. Bit0 do bit3 są odpowiednio dla MR0 do MR3 i podobnie następne cztery dla przerwań CR0 do CR3. Wpisanie 1 do odpowiedniej lokalizacji bitu zresetuje przerwanie (powiadomi o wykonaniu przerwania)

Na potrzeby implementacji przetłumaczę aliasy użyte w programie:

- T1TCR = TIMER_RESET (_BIT(1)) - używane z rejestrem TCR ustawia bit1, czyli TC i PC na 0
- T1IR = TIMER_ALL_INT - ustawia bity wszystkich przerwań na 1, a więc resetuje je
- T1MCR = MR0_S (_BIT(2)) – używane z rejestrem MCR ustawia bit2 – zatrzymuje TC i PC po osiągnięciu zadanej wartości
- T1TCR = TIMER_RUN (_BIT(0)) – ustawia bit0 – użyte z rejestrem TCR włącza timer
- Znaczenie pętli:
  while (T1TCR & TIMER_RUN) {
  }
    - Podczas gdy obie wartości są takie same pętla będzie działać

- Po zakończeniu odliczania czasu przez timer wartość rejestru TCR ulegnie zmianie i pętla się zakończy

3.8. Obsługa przerwań

Przerwania obsługiwane są poprzez timer. Działanie timera mocno opisałem w stosownym podrozdziale dokumentacji. Na potrzeby przerwań wykorzystujemy timer 0. W naszym programie przerwania wykorzystywane są do sterowania silnikiem (zapewniają wykonanie zadanej funkcji co dokładny okres czasu, wpływ innych opóźnień zawartych w kodzie nie ma znaczenia).

Wyróżniamy różne rodzaje przerwań: szybkie, wektoryzowane oraz niewektoryzowane.

Na potrzeby naszego programu wykorzystamy przerwania wektoryzowane.

Do przerwań wykorzystywane są następujące rejestry:

- VICIntSelect(R/W) – Rejestr używany do wyboru pomiędzy IRQ (Interrupt Request), a FIQ (Fast Interrupt Request)
- VICIntEnable (R/W) – Rejestr używany do włączenia przerwań
- VICVectCntl0 do VICVectCntl15 (16 rejestrów) – Rejestry używane do przypisania danego źródła przerwania do odpowiedniego slotu. Slot 0 ma najwyższy priorytet, slot 15 – najwyższy
- VICVectAddr0 do VICVectAddr15 (16 rejestrów) – Rejestry używane do przechowania adresu funkcji do wywołania podczas przerwania. Muszą odpowiadać Slotom przypisanym w rejestrze VICVectCntlX
- VICVectAddr **Uwaga:** Nie mylić z VICVectAddrX! – Rejestr trzyma adres wykonywanej funkcji. Wpisanie wartości do tego rejestru sygnalizuje zakończenie przerwania.

Procedura przerwania:

1. W rejestrze VICIntSelect należy ustawić bit odpowiadający wybranemu wejściu na 0 aby wybrać wektoryzowane przerwania.
   VICIntSelect &= ~TIMER_0_IRQ; //Przerwanie od Timera #0 przypisane do IRQ
2. W rejestrze VICVectAddrX z odpowiednim numerem slotu (u nas 5) należy wpisać adres funkcji wykonywanej podczas przerwania
   VICVectAddr5 = (tU32) IRQ_Test;
3. W rejestrze VICVectCntlX w odpowiednim slocie (u nas 5) przypisujemy odpowiednie urządzenie (u nas timer 0)
   VICVectCntl5 = VIC_ENABLE_SLOT | TIMER_0_IRQ_NO; //Przypisanie i odblokowanie slotu w VIC
4. W rejestrze VICIntEnable włączenie obsługi przerwania dla wybranego urządzenia (u nas timer 0) – ustawienie odpowiedniego bitu określającego wejście

Cała procedura przerwania kończy się wpisaniem bitu do odpowiedniego rejestru IR (patrz podrozdział o timerze) oraz 0 do rejestru VICVectAddr:

T0IR = TIMER_MR0_INT;
VICVectAddr = 0x00;

3.9. Obsługa silnika krokowego

Silnik krokowy znajdujący się na płytce jest typu bipolarnego. Zasilany jest napięciem 3V.

W ciągu całego obrotu wykonuje 20 kroków, co znaczy, że jeden krok wynosi 18 stopni.

Sterowanie odbywa się za pomocą wysyłania odpowiedniej sekwencji sygnałów za pomocą pinów P0.12 oraz P0.21.

Aby zapewnić zadaną prędkość silnika sygnał jest wysyłany poprzez przerwania (omówione w innym podrozdziale tej dokumentacji), co niweluje wpływ jakichkolwiek innych opóźnień.

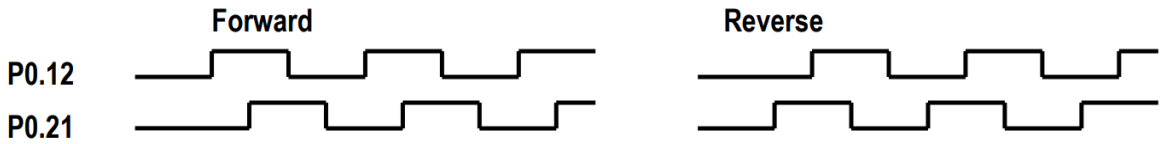
Kolejność wysyłania sygnałów określa kierunek obrotu silnika. Zostało to przedstawione na załączonym schemacie.

W programie zostało to zaimplementowane w następujący sposób:

1. Ustawiamy stan obu pinów na niski (IOCLR0) podając 0x00201000, co odpowiada pinom P0.12 oraz P0.21
2. Podczas kolejnych przerwań ustawiamy na stan wysoki (IOSET0) piny zgodnie z następującą sekwencją:
    a. 0x00201000 (P0.12 oraz P0.21)
    b. 0x00200000 (P0.21)
    c. 0x00000000 (Żaden pin nie jest ustawiony, oba są w stanie niskim)
    d. 0x00001000 (P0.12)

W naszym programie przerwanie zostaje wykonane co 10 ms, co sprawia, że 1 krok silnika wykonywany jest co 10ms, a cały obrót zajmuje 200ms.

Dzięki tej wiedzy możemy obliczyć, że wiatraczek kręci się z prędkością 300 obr/min.



Rysunek 7 kolejność wysyłania sygnałów do silnika krokowego
Źródło: LPC2148 Education Board - User's Guide

3.10. Obsługa GPIO (przycisk + proste menu)

Do zaprezentowania wszystkich danych na wyświetlaczu musieliśmy dodać implementację przycisku z prostym menu.

Wybrany przez nas przycisk jest na pinie 14.

Implementacja w programie:

1. Ustawiamy pin 14 w rejestrze IODIR1 jako wejście
   IODIR1 |= (1 << 14)

2. Odczytujemy stan przycisku – jeśli stan jest wysoki (0) zmienna *switchStatus* poprzez koniunkcję logiczną ustawi się na 0. Gdy przycisk nie jest wciśnięty – jego stan jest niski (1) i poprzez koniunkcję logiczną do zmiennej wpisywana jest jedynka.
   switchStatus = (IOPIN0 >> 14) & 0x01
3. Następnie ustawiony jest warunek, który wykonuje się tylko gdy zmienna *switchStatus* przyjmuje wartość 0
   if (switchStatus == 0)
4. Gdy powyższy warunek zostanie spełniony uruchomione zostaje menu, które ustawia kolejną opcję (*mode*) z przedziału <1,3>. Gdy zostanie osiągnięta opcja 3, kolejną jest 1. Odpowiada ona za wyświetlany aktualnie ekran.
   if (mode < 3) mode++;
   else mode = 1;
5. Na koniec zostało wprowadzone opóźnienie o wartości 1 sekundy. Dzięki temu użytkownik ma czas na puszczenie przycisku i kontrolę nad tym na jakim ekranie chce się znaleźć. Gdyby nie zostało to zaimplementowane praktycznie niemożliwym byłoby wybranie interesującego nas ekranu, gdyż zmieniały by się w bardzo szybkim tempie. Opóźnienie to zostało zrealizowane za pomocą timera opisanego w innym podrozdziale tej instrukcji
   delay_sec(1)

4. Analiza skutków awarii

W tym punkcie przedstawimy skutki awarii poszczególnych elementów. Jako awarię będziemy traktować zarówno uszkodzenie samego urządzenia, jak i połączenia pomiędzy mikrokontrolerem.

Głównym zadaniem naszego programu jest interakcja z użytkownikiem. Realizowana jest ona poprzez wyświetlanie czasu, temperatur, wilgotności, a także włączanie modułu klimatyzacji. Użytkownik wchodzi w interakcję z płytką poprzez przycisk. Biorąc to pod uwagę sytuację, w której interakcja po obu stronach jest niemożliwa będziemy traktować jako awarię krytyczną.

Jeżeli uszkodzeniu ulegnie któraś z funkcjonalności, ale nie wpłynie na działanie pozostałych będziemy ją traktować jako średnią. Jeśli uszkodzeniu ulegną wszystkie funkcjonalności z tej kategorii, awarię należy traktować jako krytyczną.

Awaria mikrokontrolera – uniemożliwia działanie programu.

Awaria Ekranu LCD – Sprawia, że interakcja z użytkownikiem jest niemożliwa. Nie jesteśmy w stanie odczytać żadnej z danych nam przekazywanych.

Awaria przycisku – Bez działającego przycisku nie mamy możliwości korzystania z funkcjonalności płytki. Eliminuje on możliwość przełączania się pomiędzy ekranami, dlatego traktujemy ją jako krytyczną.

Awaria wiatraczka/silnika krokowego – Jako jedyna sklasyfikowana jako awaria niegroźna. Jest to moduł dodatkowy, który nie wpływa na funkcjonalności płytki jako stacji pogodowej.

Awaria timera – Uniemożliwia realizowanie opóźnień, przerwań itp. Program nie może wykonać się prawidłowo, jest to więc awaria krytyczna.

Awaria wbudowanego termometru – Uniemożliwia odczyt temperatury wewnątrz pomieszczenia.

Awaria czujnika DHT11 - Uniemożliwia odczyt temperatury i wilgotności na zewnątrz pomieszczenia. Nie przerywa jednak działania programu.

Expansion Board – Uniemożliwia działanie DHT11.

Urządzenie	Skutki awarii
Mikrokontroler	Krytyczne
Ekran LCD	Krytyczne
Przycisk	Krytyczne
Wiatraczek/silnik krokowy	Niegroźne
Timer	Krytyczne
Wbudowany termometr	Średnie
DHT11	Średnie
Expansion Board	Średnie

5. Spis ilustracji

6. Załączniki

1. 20by20l033.pdf –dokumentacja silnika krokowego
2. DHT11 Documentation.pdf – dokumentacja czujnika DHT11
3. KS0070B documentation.pdf – dokumentacja sterownika ekranu LCD
4. LPC2148_Education_Board_Users_Guide-Version_3.0_Rev_D.pdf – podręcznik użytkowania LPC2148v3