Informatyka, studia dzienne, inż I st. semestr VII

Programowanie współbieżne **2013/2014**

Prowadzący: czwartek, 12:45

Data oddania: _____ Ocena: _____

Łukasz Kosma (Lider)	165445
Maciej Czarnecki	165384
Michał Zarychta	165557

E7: Gra Snake

Spis treści

1. Zakres projektu

Funkcjonalność	Stan	Osoba odpowiedzialna
Ekran LCD	Wykonane	Maciek Czarnecki
SPI	Wykonane	Maciek Czarnecki
Joystick	Wykonane	Michał Zarychta
Timer	Wykonane	Michał Zarychta
Brzęczyk	Wykonane	Maciek Czarnecki
Diody	Wykonane	Łukasz Kosma
Bluetooth	Wykonane	Łukasz Kosma
Uśpienie procesora	Częściowo wykonane	Michał Zarychta

2. Wykorzystany sprzęt

Projekt jest oparty o układ Gameboard. W skład sprzętu wchodzą między innymi:
— mikrokontroler LPC2104 z 128 KB Flash oraz 128 KB SRAM
— kolorowy wyswietlacz LCD
— 5-pozycyjny joystick
— diody LED
— brzęczyk
— bluetooth
— interfejs SPI

3. Dokumentacja użytkownika

3.1. Zasady rozgrywki

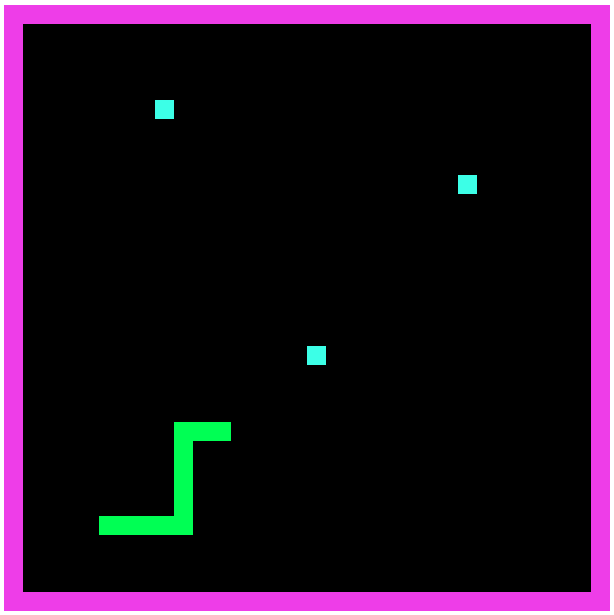
Celem gry jest zebranie jak największej liczby przekąsek symbolizowanych przez kolorowe kwadraty. Po konsumpcji nasz wąż rośnie co zwiększa poziom trudności. W przypadku gdy nastąpi kolizja węża z jego własnym ogonem następuje koniec rozgrywki. Przekąski są ustawiane w losowych pozycjach.

3.2. Sterowanie

Sterowanie w grze przebiega przy pomocy joysticka. Poruszając nim wybieramy kierunek ruchu węża (ruch przebiega samoczynnie). Wciśnięcie kontrolera powoduje zatrzymanie sterowanego węża (dodatkowym efektem jest zapalanie/gaszenie się diod).

3.3. Menu główne

W głównym menu poruszamy się przy pomocy joysticka i zatwierdzamy wybór przy pomocy przyciśnięcia. Są tutaj dostępne 2 opcje - nowa rozgrywka



Rysunek 1: *Okno rozgrywki.*

oraz opcja wysłania wyniku przy pomocy Bluetooth'a. W przypadku wysyłania wyniku konieczne jest wcześniejsze sparowanie urządzeń Bluetooth. Aby móc odebrać wynik na komputerze PC, potrzebna jest stworzona przez nas do tego druga aplikacja.

3.4. Plansza

Plansza składa się z kwadratów ułożonych na siatce o wymiarach 32x32. W pamięci aplikacji przetrzymywane są obiekty znajdujące się na planszy, czyli wąż, przekąska i obramowanie planszy. Tablica jest wykorzystywana zarówno do zapisu jak i do odczytu danych. Przy jej pomocy jest roztrzygana kolizja, oraz jest wykorzystywana w pętli rysowania.

4. Realizacja projektu

4.1. Checkpoint 1

Podczas tego checkpointu skupiliśmy się na inicjalizacji LCD oraz wyświetlaniu obiektów. Dodatkowo udało nam się zmieniać stan diod po wciśnięciu joysticka. Wykonana została również podstawowa obsługa brzęczyka, która nie zakładała zmian tonów.

Zaplanowana funkcjonalność	Stan
Ekran LCD	Wykonane
Joystick	Wykonane
Obsługa diod	Dodatek [1]
Podstawowa obsługa brzęczyka	Dodatek

4.2. Checkpoint 2

Podczas tego punktu kontrolnego chcieliśmy rozbudować funkcjonalności już zaczęte. Została zaimplementowana logika gry, która opierała się na połączeniu LCD z reakcją na joystick. Dodatkowo przy pomocy timera udało nam się zagarć kilka różnych tonów prtz pomocy brzęczyka.

Zaplanowana funkcjonalność	Stan
Połączenie LCD z joystickiem	Wykonane
Akordy grane przez brzęczyk	Dodatek
Usypianie procesora	Dodatek

5. Opis funkcjonalności

5.1. Buzzer

Brzęczyk posłużył nam do odegrania kilku krótkich melodyjek. Jedna z nich to melodia na początku rozgrywki (odegranie C D E F G A B C). Podczas gdy wąż zje przekąskę, zostanie odegrana druga, krótka melodia składająca się z coraz to wyższych tonów wraz ze zwiększaniem się długości węża. Do odgrywania melodii użyliśmy następującej funkcji:

```
static void
startMusic(void)
{
    setBuzzer(toggle == 0 ? FALSE : TRUE);

    toggle = ~toggle;
    count++;

    if(count == notes[note]*10)
    {
      if(note < 8)
      note++; // Przejscie do
      if(note == 8)            // nastepnego dzwieku
      {
        T1MCR = 0x06;
        setBuzzer(FALSE);
        setLED(LED_GREEN, FALSE); // Obsluga
        setLED(LED_RED, FALSE);   // diod
        note = 0;
      }
      T1TCR = 0x02; //Zablokuj i zrestartuj zegar
      T1MR0 = CORE_FREQ / (100*notes[note]);
      T1TCR = 0x01;
      count = 0;
    }

    T1IR         = 0x000000ff;//reset flag IRQ
```

```
    VICVectAddr = 0x00000000;//zakonczenie obslugi przerwania
}
```

Listing 1: Funkcja odpowiedzialna za odegranie melodii

Brzęczyk posiada połączenie przez 1 pin GPIO, więc może mieć 2 stany - włączony i wyłączony. Ton dźwięku uzyskujemy przy pomocy włączania i wyłączania brzęczyka z odpowiednią częstotliwością. Jest to realizowane przy pomocy zegara, a dokładniej rejestru oznaczonego przez T1MR0 (Match Register 0 timera 1). Częstotliwość jest dostosowana do taktowanai procesora, tak, by dźwięk nut się zgadzał.

5.2. Timer

Płytka LPC2104 pozwala obsługiwać dwa timery (timer 0 i timer 1), z których wybraliśmy timer 1 do realizacji naszych zadań (głównie do ograniczenia czasu odgrywania pewnych melodii, w określonych momentach gry). Przykład takiego zastosowania timera został przedstawiony na listingu 1. Obecność przedrostka *T1* przy odpowiednich nazwach rejestrów związanych z timerem wynika właśnie z numeru używanego timera.

Użyte przez nas rejestry to:

— T1MCR (Match Control Register) pozwala określić operacje, które zostaną wykonane, gdy timer osiągnie jedną z określonych wartości.

— T1TCR (Timer Control Register) kontroluje działanie timera (umożliwia rozpoczęcie odliczania oraz resetowanie licznika).

— T1MR0 (Match Register 0) jeden z czterech rejestrów służących głównie do zapamiętania wartości timera, dla której zostanie wygenerowane przerwanie.

— T1PC (Prescale Counter Register) jest zwiększany aż do osiągnięcia wartości zapisanej w T1PR (Prescale Register). Gdy to nastąpi, wartość T1TC (Timer Counter) jest zwiększana, a T1PC zostaje wyczyszczony.

— T1IR (Interrupt Register) służy do czyszczenia wybranych przerwań (z ośmiu możliwych) albo do sprawdzenia, czy któreś z nich nastąpiło.

By wykorzystywać funkcjonalność timera, przygotowywaliśmy funkcję, której adres wpisany został do wektora przerwań, tak by przerwania wybranego przez nas timera obsługiwane były przez naszą funkcję.

```
static void
initTimer1(tU32 functionAdress)
{
//initialize and start Timer 1
T1TCR = 0x00000002; //disable and reset Timer 1
T1PC  = 0x00000000; //no prescale of clock
//calculate no of timer ticks (for note[0] its 3.8ms)
T1MR0 = CORE_FREQ / (100*notes[note]);
T1IR  = 0x000000ff; //reset all flags before enable IRQs
T1MCR = 0x00000003; //reset counter and generate IRQ on MR0 match

//initialize VIC for Timer0 interrupts
VICIntSelect &= ~0x20; //Timer1 interrupt is assigned to IRQ(!FIQ)
VICVectAddr5  = functionAdress; //register ISR address
VICVectCntl5  = 0x25; //enable vector interrupt for timer1
```

```
VICIntEnable  |= 0x20; //enable timer1 interrupt
T1TCR = 0x00000001; //start Timer1
}
```

Listing 2: Funkcja inicjalizująca Timer 1.

5.3. LCD

Płytka wyposażona jest w wyświetlacz LCD 130x130. Sterujemy nim przy pomocy SPI przy pomocy funkcji bibliotecznych. Dane są wysyłane za pomocą 9-bitowych paczek.

5.3.1. Inicjalizacja

Z uwagi na to, że w aplikacji wysyłamy do LCD jedynie 9 bitów, musimy zawęzić paletę kolorów. Podczas inicjalizacji należy wykonać indeksowanie barw. Dla formatu GBR o przestrzeniach kolejno: 3, 3 i 2 bitów musieliśmy zdefiniowac 20 poziów jasności piksela.

5.3.2. Rysowanie

Rysowanie na ekranie LCD przebiega przy pomocy SPI. Przed wysłaniem danych do zapisu w kontrolerze LCD jest wybierane okno do rysowania. Pozwala to oszczędzić czas przesyłając mniej danych.

```
void
lcdRect(tU8 x, tU8 y, tU8 xLen, tU8 yLen, tU8 color)
{
  tU32 i;
  tU32 len;

  selectLCD(TRUE);     // Wybranie kontrolera LCD.

  lcdWindow1(x,y,x+xLen-1,y+yLen-1); // Wybranie okna do zapisu.

  lcdWrcmd(LCD_CMD_RAMWR);// Wyslanie komendy "zapis do pamieci".

  len = xLen*yLen;  // Zapis danych.
  for(i=0; i<len; i++)
    lcdWrdata(color);

  selectLCD(FALSE); // Odznaczenie kontrolera LCD.
}
```

Listing 3: Wyrysowanie na wyświetlaczu prostokąta.

Komunikacja z wyświetlaczem przebiega przy pomocy magistrali SPI.

5.4. Diody

Płytka wyposażona jest w dwie diody LED: czerwoną i zieloną.

Interakcja z diodami realizowana jest przez GPIO (General Purpose Input/Output). Ustawiamy na odpowiednich pinach kierunek na wysyłanie informacji. Następnie ustawiamy stan wysoki, aby diody były zgaszone. Służy do tego poniższa funkcja:

```
void
immediateIoInit(void)
{
(...)
    IODIR |= (LED_GREEN_PIN | LED_RED_PIN);
    IOSET  = (LED_GREEN_PIN | LED_RED_PIN);
(...)
}
```

Listing 4: Ustawienie pinów odpowiedzialnych za diody na kierunek wyjściowy (wartości 1) oraz określenie ich wartości początkowych.

Do zarządzania diodami wykorzystaliśmy następującą funkcję, która pozwala włączyć/wyłączyć diodę. Zapalanie i gaszenie diod polega na ustawieniu na odpowiednim pinie stanu niskiego (rejestr IOCLR – GPIO Port Output Clear register) lub wysokiego (rejestr IOSET – GPIO Port Output Set register).

```
void
setLED(tU8 ledSelect, tBool ledState)
{
  tU8 commandLedGreen[] = {0x07, 0x00};
  //                              40 = LED on
  //                              00 = LED off

  tU8 commandLedRed[] =   {0x08, 0x00};
  //                              01 = LED on
  //                              00 = LED off

  //check if ver 1.0 of HW
  if (TRUE == ver1_0)
  {
    if (LED_GREEN == ledSelect)
    {
      if (TRUE == ledState)
        IOCLR = LED_GREEN_PIN;
      else
        IOSET = LED_GREEN_PIN;
    }
    else if (LED_RED == ledSelect)
    {
      if (TRUE == ledState)
        IOCLR = LED_RED_PIN;
      else
        IOSET = LED_RED_PIN;
    }
  }
}
```

Listing 5: Obsługa diod (czyli ustawienie wybranej diody na wskazany stan).

5.5. Tryb Power-down

Wejście w tryb *Power-down* procesora odbywa się domyślnie poprzez ustawienie wartości drugiego bitu (PD) rejestru PCON (Power Control) na

1. Gdy oba bity (pierwszy i drugi) zostaną ustawione na wartość 1, wtedy również nastąpi przejście w tryb *Power-down*.

```
void
enterSleepMode()
{
    PCON = 0x2;
}
```

Listing 6: Funkcja przejścia procesora w tryb uśpienia tzn. w tryb *Power-down*

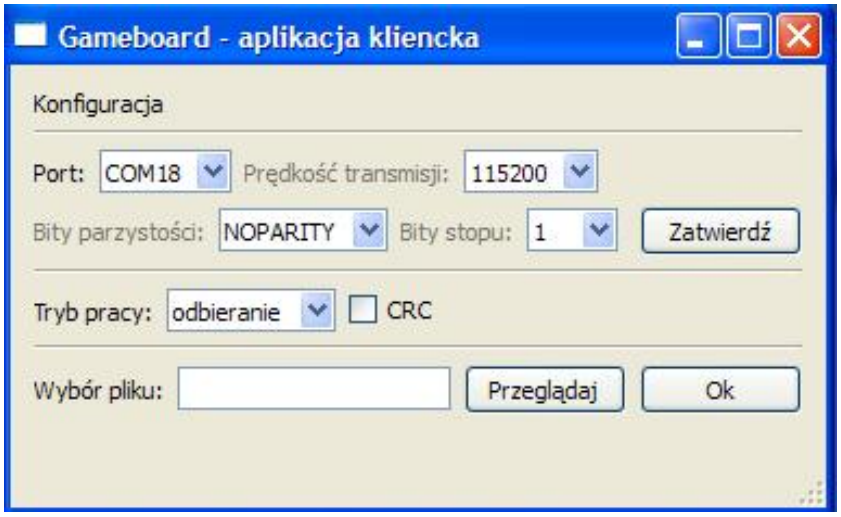
5.6. Bluetooth

Płytka jest wyposażona w moduł Bluetooth, Philips BGB203-S06. Pozwala on wykryć pobliskie urządzenia i skomunikować się z nimi. Do komunikacji między urządzeniami wykorzystywany jest profil portu szergowego (SPP - Serial Port Profile). Parametry transmisji są następujące:
— Prędkość transmisji: 115200 bodów
— Bity danych: 8
— Bity parzystości: brak
— Bity stopu: 1
Do transmisji przez Bluetooth wykorzystaliśmy protokół bazujący na XMODEM. Dane podzielone są na bloki, z których każdy zawiera: numer sekwencyjny bloku, 16 bajtowe dane i 1 bajtową sumę kontrolną, która jest wyliczana z bloku danych.
Obsługa Bluetooth'a realizowana jest przez poniższą funkcję. Ustawiane są w niej parametry transmisji. W nieskończonej pętli realizowane jest odbieranie danych.

```
static void
procBt(void* arg)
{
  tU8 error;

  //reset BGB203 modules
  resetBT(TRUE);
  osSleep(2);
  resetBT(FALSE);
  osSleep(5);

  //extra sleep
  osSleep(50);

  //initialize uart1: 115200 kbps, 8N1, FIFO
  initUart1(B115200((CORE_FREQ) / PBSD), UART_8N1, UART_FIFO_16);

(...)

  while(1)
  {
    osSemTake(&recvSem, 0, &error);

    while (stopRecvProc == FALSE)
```

```
25      {
          tU8 rxChar;
27        tU8 nothing;


29
          nothing = TRUE;
31
(...)
33
          //check if any character has been received from BT
35        if (uart1GetChar(&rxChar) == TRUE)
          {
37          printf("%c", rxChar);
            nothing = FALSE;
39        }

41        if (nothing == TRUE)
            osSleep(1);
43      }
        stopRecvProc = FALSE;
45    }
}
```

Listing 7: Funkcja odpowiedzialna za obsługę bluetootha.



Rysunek 2: *Interfejs aplikacji na komputer PC do odbierania danych wysyłanych przez Bluetooth.*

Poniżej zamieszczona została funkcja inicjalizująca Uart niezbędna do przesyłania i odbierania danych przez Bluetooth'a oraz opis poszczególnych rejestrów

Podstawowe rejestry wykorzystywane w trakcie tej inicjalizacji to (przedrostek *U1* oznacza *UART 1*):

— PINSEL0 (Pin function select register 0)
— U1IER (Interrupt Enable Register)
— U1IIR (Interrupt Identification Register)
— U1RBR (Receiver Buffer Register)

— U1LSR (Line Status Register)
— U1LCR (Line Control Register)
— U1FCR (FIFO Control Register)
— U1MCR (Modem Control Register)

```
void
initUart1(tU16 div_factor, tU8 mode, tU8 fifo_mode)
{
  volatile tU32 dummy;

  //initialize the receive semaphore
  osSemInit(&receiveSem, 0);

  //enable uart 1 pins in GPIO (P0.8  = TxD1, P0.9  = RxD1)
  //                           (P0.10 = RTS1, P0.11 = CTS1)
  PINSEL0 = (PINSEL0 & 0xff00ffff) | 0x00550000;

  U1IER = 0x00;                 //disable all uart interrupts
  dummy = U1IIR;                //clear all pending interrupts
  dummy = U1RBR;                //clear receive register
  dummy = U1LSR;                //clear line status register

  //set the bit rate = set uart clock (pclk) divisionfactor
  U1LCR = 0x80; //enable divisor latches (DLAB bit set, bit 7)
  U1DLL = (tU8)div_factor;      //write division factor LSB
  U1DLM = (tU8)(div_factor >> 8);//write division factor MSB

  //set transmissiion and fifo mode
  U1LCR = (mode & ~0x80);       //DLAB bit (bit 7) must be reset
  U1FCR = fifo_mode;

  //initialize the interrupt vector
  VICIntSelect &= ~0x00000080;  // UART1 selected as IRQ
  VICVectCntl7  =  0x00000027;
  VICVectAddr7  =  (tU32)uart1ISR;// address of the ISR
  VICIntEnable |=  0x00000080;  // UART1 interrupt enabled

  //initialize the transmit data queue
  uart1TxHead    = 0;
  uart1TxTail    = 0;
  uart1TxRunning = FALSE;

  //initialize the receive data queue
  uart1RxHead   = 0;
  uart1RxTail   = 0;
  uart1RxInBuff = 0;

  //set RTS high (= accept received bytes)
  U1MCR = 0x02;

  //enable receiver interrupts
  U1IER = 0x0d; //0x01;  //incl. modem
}
```

Listing 8: Fragment kodu odpowiedzialny za inicjalizację UART.

5.7. Joystick

Joystick zamontowany na płytce ma pięć możliwych stanów poza stanem początkowym. Cztery z nich dotyczą wychylenia w kierunkach: dół, góra, lewo i prawo, a ostatni dotyczy wciśnięcia joysticka na samym środku. Sprawdzanie stanu joysticka odbywa się przy użyciu GPIO (General Purpose Input/Output). Podstawowe rejestry GPIO, z których korzystamy, by móc obsługiwać joystick to:

— IODIR (GPIO Port Direction control register) służący do określenia kierunku dla każdych portów pinów. Wartość 0 oznacza, że wybrany pin jest wejściem, a wartość 1, że wyjściem.
— IOPIN (GPIO Port Pin value register) pozwala zawsze na odczyt stanów pinów niezależnie od wybranych dla nich kierunków.

Piny (32 bitowego portu 0) odpowiedzialne za kolejne kierunki (wraz z nazwami używanymi w naszym programie):

— KEYPIN_CENTER 0x00004000 (środek: pin P0.14)
— KEYPIN_UP 0x00010000 (góra: pin P0.16)
— KEYPIN_DOWN 0x00800000 (dół: pin P0.23)
— KEYPIN_LEFT 0x00008000 (lewo: pin P0.15)
— KEYPIN_RIGHT 0x00400000 (prawo: pin P0.22)

```
IODIR &= ~(KEYPIN_CENTER | KEYPIN_UP | KEYPIN_DOWN |
          KEYPIN_LEFT | KEYPIN_RIGHT);
```

Listing 9: Wybór kierunku wejściowego dla pinów związanych z joystickiem.

By sprawdzić czy joystick został wciśnięty w wybranym kierunku, wystarczy sprawdzić stan pinu odpowiadającego temu kierunkowi. Jeśli jest on równy 0, to joystick został wciśnięty w oczekiwany sposób.

```
readKeys = KEY_NOTHING;
if ((IOPIN & KEYPIN_CENTER) == 0) readKeys |= KEY_CENTER;
if ((IOPIN & KEYPIN_UP) == 0)     readKeys |= KEY_UP;
if ((IOPIN & KEYPIN_DOWN) == 0)   readKeys |= KEY_DOWN;
if ((IOPIN & KEYPIN_LEFT) == 0)   readKeys |= KEY_LEFT;
if ((IOPIN & KEYPIN_RIGHT) == 0)  readKeys |= KEY_RIGHT;
```

Listing 10: Zapisanie w zmiennej readKeys kierunków związanych ze stanem joysticka.

5.8. SPI

Z interfejsu korzystamy przy okazji obsługi LCD. Transmisja przebiega w trybie duplex, więc mamy tutaj przesył danych w obu kierunkach jednocześnie. W danej chwili może trwać komunikacja jedynie między 2 urządzeniami w układzie master-slave. Nasza aplikacja dokonuje przesyłu danych do LCD. W SPI mamy do dyspozycji 4 linie:

- SCK (ang. Serial CLocK) - sygnał zegarowy generowany przez układ master.
- MOSI (ang. Master Output Slave Input) - na tej linii przebiega komunikacja master → slave.
- MISO (ang. Master Input Slave Output) - na tej linii przebiega komunikacja

slave → master.
- SSEL (ang. Slave SELect) - używana do wyboru urządzenia w trybie slave.

6. Analiza skutków awarii

Przede wszystkim przeanalizujemy wpływ awarii na części systemu związane ze zreazlizowanymi funkcjonalnościami.

— Awaria buzzera jest niegroźna. Jest to dodatkowy element systemu mający na celu uatrakcyjnienie gry. Żadne istotne dla gracza informacje nie są przez niego udostępniane.
— Awaria a jest niegroźna. Wpłynie ona głównie na buzzer. Główna pętla gry opóźniana jest natomiast przy pomocy funkcji *osSleep(delay);*
— Awaria wyświetlacza LCD jest krytyczna. Uniemożliwia całkowicie granie (nawet graczom którzy ekranu nie potrzebują, a to dlatego, że część istotnych zależności w grze jest odczytywana wprost z wyświetlacza np. czy doszło do kolizji), które jest głównym przeznaczeniem systemu.
— Awaria diod jest niegroźna. Ich miganie jest całkowicie nieistotne z punktu widzenia gracza.
— Awaria przejścia w tryb *Power-down* jest niegroźna jeśli nie będzie można w niego wejść, ale można ją uznać za krytyczną, gdyby nie można było opuścić tego trybu w trakcie gry. Zmusiłoby to gracza do zresetowania całego systemu, który przestałby odpowiadać na działania użytkownika.
— Awaria bluetooth'a jest średnim zagrożeniem. Może zniechęcić gracza do kontynuowania rozgrywki, gdyż związana jest ona z przekazywaniem wyników gry, ale nie uniemożliwia korzystania z systemu. Gra traci jednak na atrakcyjności, z powodu zaniknięcia czynnika rywalizacji.
— Awaria joystick'a jest krytyczna. Użytkownik traci całkowicie kontrolę nad rozgrywką, która po chwili kończy się kolizją węża. Nie ma również możliwości skorzystania z wymiany danych (w tym przesłania wyniku) przy pomocy bluetooth'a.
— Prawidłowe działanie SPI jest niezbędne. Z magistrali korzysta do przesyłu danych ekran, więc bez niej system przestałby działać.

7. Podsumowanie

Aplikacja została napisana wykorzystując funkcje biblioteczne do obsługi urządzeń. Dzięki bezpośredniej komunikacji mogliśmy zauważyć bardzo szybką reakcję sprzętu. Głównym problemem podczas implementacji była mała ilość materiałów opisujących działanie układu, jednak oparcie się na dokumentacji urządzenia, pozwoliło na zrealizowanie założeń projektu.

8. Bibliografia

— Dokumentacja użytkownika:
UM10275 LPC2104/2105/2106 (Rev. 02—8 April 2009)
— Kod źródłowy przykładowego programu na Gameboard'a:
http://www.zsk.p.lodz.pl/ morawski/SCR&ES/Demo/GameBoard/lcd_src_v1.9.zip