Politechnika Łódzka

Wydział Fizyki Technicznej, Informatyki
i Matematyki Stosowanej

Kierunek: Informatyka
Przedmiot: Systemy wbudowane

Dokumentacja gry zręcznościowej Procesja™ LPC1343 + Expansion Board

lider: mgr Filip Turoboś
Nr albumu: 210344 / 801147
Jan Filipowicz
Nr albumu: 203875
Krzysztof Wierzbicki
Nr albumu: 210347

Łódź,

Spis treści

1 Podsumowanie i technikalia

1.1 Skład zespołu

Funkcja:	Imię i nazwisko:	Nr albumu:
lider	Filip Turoboś	210344, 801147
członek	Jan Filipowicz	203875
członek	Krzysztof Wierzbicki	210347

1.2 Sprzęt

Na potrzeby projektu nie wykonywano żadnego własnego sprzętu. Do projektu wykorzystano LPC1343 i Expansion Board.

1.3 Wykorzystane funkcjonalności i ich autorzy

Timer	Krzysztof Wierzbicki
GPIO	Jan Filipowicz
ADC	Turoboś Filip
SPI/SSP	Krzysztof Wierzbicki
I^2C	Jan Filipowicz
Wyświetlacz siedmiosegmentowy	Turoboś Filip
Wyświetlacz OLED	Krzysztof Wierzbicki
Przerwania	Jan Filipowicz
Melodia	Krzysztof Wierzbicki
Akcelerometr	Turoboś Filip
Czujnik światła	Turoboś Filip
Diody LED – Ekspander (PCA 9532)	Krzysztof Wierzbicki

1.3.1 Dokumentacja, analiza FMEA i inne uwagi

Algorytm Gry: Jan Filipowicz;
Analiza FMEA: Cały zespół;
Skład i opracowanie całości dokumentu: Filip Turoboś;

W naszym odczuciu praca w zespole została rozłożona stosunkowo równomiernie. Nad niemal każdą z funkcjonalności (a także opisem w dokumentacji) pracowaliśmy całym zespołem.

1.3.2 Procentowy udział poszczególnych członków zespołu w tworzeniu końcowej wersji projektu

Imię i nazwisko:	Procentowy udział:
Filip Turoboś	33%
Jan Filipowicz	33%
Krzysztof Wierzbicki	34%

2 Skrótowy opis działania programu

Podczas gry w Procesja^{TM} gracz manewruje przy pomocy joysticka i/lub akcelerometru białą łamaną symbolizującą procesję i stara się poruszać nią w taki sposób, aby:

- nie przecinać tworzonej przez siebie łamanej;
- zbierać pojawiające się na planszy kropki symbolizujące zbłąkane owieczki.

W przypadku gdy gracz najedzie łamaną na pulsującą kropkę, całość procesji zostaje przedłużona. Radując się z ilości zgromadzonych wiernych możemy odgrywać pieśń religijną *"Barka"*.

3 Algorytm gry

Gra działa na zasadzie popularnej gry Snake – procesja porusza poprzez usunięcie ostatniej kropki (ostatniego segmentu symbolizującego wiernego) i dodaniu kropki symbolizującej proboszcza na początku procesji. Rozpoczęcie gry polega na zainicjalizowaniu pięciu kropek tworzących procesję na środku poziomu (poziom ma wymiary 32×21 punktów) oraz utworzeniu kropki wiernego w losowym miejscu na planszy.

Procesja porusza się ruchem jednostajnym w ostatnim wskazanym przez gracza kierunku (początkowo w górę). Pochód może przechodzić przez *ścianki* poziomu, kontynuuje on wówczas swój ruch po przeciwnej stronie ekranu (poziom jest *de facto* homeomorficzny z torusem). Przy każdym ruchu procesji obliczana jest odległość proboszcza (początku procesji) od zbłąkanej owieczki, która mruga. Gdy odległość ta wynosi nie więcej niż `TOLERANCE` (zmienna oznaczająca precyzję wymaganą od gracza, domyślnie ma wartość zero) wierny dołącza do pochodu, wydłużając długość procesji o jeden. Fakt ten sygnalizowany jest błyśnięciem diodami LED. Ponadto w losowym miejscu pojawia się nowa zbłądana owieczka. Jeżeli czoło pochodu zderzy się z jego częścią (lub zostanie zakryty czujnik światła) to gra się restartuje, wracając do stanu początkowego.

4 Funkcjonalności

4.1 Timer

Układ, który dekrementuje lub inkrementuje wartość jednego ze swoich rejestrów w zależności od częstotliwości otrzymywanego sygnału nazywamy **timerem**. Każdy timer jest wyposażony w dwa podstawowe rejestry – licznik timera i rejestr kontroli timera. W przypadku płytki LPC 1343 timer jest dodatkowo wyposażony w preskaler. Zwiększenie licznika timera (TC) – *timer counter* następuje po spełnieniu następujących warunków:

- Wartość rejestru preskalera (PR) jest ustawiona na pewną wartość $K \in \mathbb{N}_0$ – domyślnie $K = 0$;
- 32-bitowy licznik preskalera (PC) osiągnie wartość $K + 1$;

Po inkrementacji (TC) następuje wyzerowanie licznika preskalera.

Aby uruchomić timer należy:

- Uruchomić przesyłanie sygnału zegarowego do 32-bitowego timera o numerze zero (CT32B0) [1]:

```
LPC_SYSCON->SYSAHBCLKCTRL |= (1<<9);
```

- Wybrać pin 5 z portu 1 do odbierania sygnału z wejścia input 0 CT32B0 ;
- Ustawiamy wejście sygnału z match registrów 0 - 2 porty 1.6, 1.7, 0.1 ;
- W match registrze timera ustawiamy interwał tj. odległość między kolejnymi dopasowaniami;
- W match control registrze TMR32B0MCR ustawiamy `0b11` - wyślij przerwanie i zresetuj jeżeli MR0 (zgodnie ze stroną 287 [1]);
- Włączamy przerwania rejestrem ISER (Interrupt Set Enable Register):

```
NVIC->ISER[((uint32_t)(IRQn) >> 5)] = (1 << ((uint32_t)(IRQn) & 0x1F));
```

4.2 GPIO

Skrót GPIO oznacza General Purpose Input/Output, czyli interfejs wejścia/wyjścia ogólnego przeznaczenia. Przy pomocy GPIO obsługiwane są joystick oraz głośnik, w które wyposażona jest płytka.

Przed wykorzystaniem GPIO włączamy jego dostęp do zegara systemowego poprzez ustawienie szóstego bitu w rejestrze SYSCON SYSAHBCLKCTRL. Następnie dla każdego wykorzystanego pina ustawiamy kierunek wejścia/wyjścia. Robi się to modyfikując zawartość rejestrów GPIODIR. W zależności od portu, wybieramy rejestr GPIO0DIR, GPIO1DIR, GPIO2DIR, lub GPIO3DIR, i ustawiamy n-ty bit na wartość B, gdzie n jest wybranym numerem pina, zaś B jest równe 0, gdy chcemy skonfigurować pin jako wejście, lub 1, gdy jako wyjście. W przypadku naszego programu, joystick wykorzystujący piny od 0 do 4 portu 2 jest konfigurowany jako wejście, natomiast głośnik wykorzystujący piny 2, 9, 10 portu 1 oraz 0, 1, 2 portu 3 jest konfigurowany jako wyjście.

Zapis i odczyt stanu GPIO odbywają się poprzez modyfikację i odczyt wartości rejestrów GPIODATA. Mają one podobną postać do rejestrów GPIODIR, tj. jest ich 4, po jednym na port, i każdy bit odpowiada jednemu pinowi. Wartość bitowa 1 na danej pozycji oznacza stan wysoki pina, zaś wartość 0 - stan niski.

GPIO może także zostać skonfigurowane, aby generować przerwania w zależności od stanu pinów, ale ta funkcjonalność nie została przez nas użyta.

4.3 PCA9532

Diody przez ekspander nie wymagają inicjalizacji. Ich użycie sprowadza się do:

- Ustalenia, które diody mają zostać włączone;
- Przez I2C pod adres `(0x60 << 1)` wysyłamy bajt kontrolny, którego 4 dolne bity oznaczają adres pierwszego rejestru, do którego chcemy zapisać informacje - `0x06`. Piąty bit oznacza flagę inkrementacji, dzięki której podany przez nas adres jest zwiększany o 1 przy każdym kolejnym przesłanym bajcie. Istotne jest to, by trzy górne bity były zerami [4];
- Pod ten sam adres przesłania 32 bitów informacji do ekspandera, gdzie na każdą diodę przypadają dwa bity informacji, określające stan, w którym ma się znaleźć. Możliwe są cztery stany [4] – ON, OFF, Blink 1 i Blink2 (korzystamy jedynie z pierwszych dwóch).

W przypadku naszego programu diody LED zapalają się sześciokrotnie, gdy do procesji dołączy kolejny wierny (kolory pojawiają się naprzemiennie).

4.4 ADC

Analog-Digital Converter, czyli **Konwerter analogowo-cyfrowy** (*ADC*) jest przetwornikiem pozwalającym na zmianę zewnętrznego sygnału analogowego (w naszym przypadku napięcia) na ciąg 10 bitów. *ADC* na stosowanym przez nas mikrokontrolerze używa algorytmu sukcesywnej aproksymacji, którego krótki opis załączony zostanie w podrozdziale **Algorytm sukcesywnej aproksymacji w** *ADC*.

Przy pomocy *ADC* gracz jest w stanie kontrolować prędkość poruszanej przez siebie procesji. Precyzyjnie, jeden krok procesji zajmuje

$$\text{new\_snek\_speed} = \left(\left\lfloor \frac{4000}{\text{ADCread}(0)} \right\rfloor + 20 \right) \ ms.$$

Do odczytu wartości z ADC stosowana jest funkcja *ADCread(0)*, która odczytuje wartość napięcia na zerowym pinie. Odczytywane przez nas wartości z *ADC* są liczbami całkowitymi z przedziału $[39; 1023]$. Dzięki temu nigdy nie dochodzi do dzielenia przez zerową wartość. Funkcja odczytująca wartość wywoływana jest przy każdym obrocie głównej pętli programu. Dzięki temu wartość prędkość procesji jest dostosowywana na bieżąco do aktualnego napięcia na zerowym pinie *ADC*, które kontroluje użytkownik przy pomocy pokrętła A22K. Zgodnie z [1] napięcie to mieści się w zakresie od 0 do 3.6 V. Inicjalizacja konwertera analogowo-cyfrowego przebiega w następujący sposób:

1. Zerujemy bit odpowiadający za odłączenie od zasilania bloku konwertera:

   ```
   LPC_SYSCON->PDRUNCFG &= ~(0x1<<4);
   ```

2. Uruchamiamy zegar AHB dla bloku ADC

   ```
   LPC_SYSCON->SYSAHBCLKCTRL |= (1<<13);
   ```

3. Do rejestru `IOCON_PIO1_4` na pierwszych dwóch bitach wpisywana jest wartość 0x01, odpowiadająca funkcji AD5, która zwraca przez input 5 [1] oraz do `JTAG_TDI_PIO0_11` wpisujemy 0x02, co odpowiada inicjalizacji funkcji AD0, zwracającej przez input 0.

4. Konfigurujemy rejestr kontrolny przetwornika następująco:

   ```
   LPC_ADC->CR = ( 0x01 << 0 ) |
   (((SystemCoreClock/LPC_SYSCON->SYSAHBCLKDIV)/ADC_Clk-1)<<8) |
   ```

```
( 0 << 16 ) |
( 0 << 17 ) |
( 0 << 22 ) |
( 0 << 24 ) |
```

Poszczególne elementy konfiguracji opisane są poniżej

- `0x01<<0` – wybór zerowego pina (AD0) do konwersji;
- `(((SystemCoreClock/LPC_SYSCON->SYSAHBCLKDIV)/ADC_Clk-1)<<8)` – dzielnik zegara dla ADC;
- `0<<16` – wybór trybu kontroli oprogramowaniem (*software controlled mode*), pozwalającej na wykorzystanie tylko jednego kanału (co zgadza się z naszą konfiguracją, bo jedynie pierwszy bit z pierwszego bajta tego rejestru ma wartość 1);
- `0<<17` – konwersja na 10 bitów/11 cykli zegarowych;
- `0<<24` – nie zaczynamy konwersji;

Na szczególną uwagę zasługuje ustawienie dzielnika zegara dla ADC. Ponieważ w naszym przypadku wartość zmiennej ADC_Clk wynosi 1 MHz, to dzielnik zegara ADC wynosić będzie (współczynnik bez jednostki):

$$CLKDIV = (60000000 \text{ Hz } /1)/1000000 \text{ Hz } - 1 = 59;$$

Aby odczytać wskazanie z ADC stosujemy funkcję ADCread(0) wybieramy (w naszym wypadku) kanał zerowy i rozpoczynamy konwersję, przestawiając wartość 24 bitu rejestru kontrolnego ADC na 1, co oznacza natychmiastowe rozpoczęcie konwersji.

```
LPC_ADC->CR |= (1 << 24) | (1 << 0);
```

Wynik konwersji zapisywany jest do rejestru o adresie `0x4001 C010` (na bitach od 6 do 15 włącznie), a następnie do zmiennej regVal. Fakt, że odczyt się powiódł oznacza że ostatni bit ma wartość 1 (zostaje on ustawiony przy zakończeniu konwersji i wyzerowany po odczytaniu). Przy korzystaniu z trybu sprzętowego BURST, należy dodatkowo sprawdzić, czy któreś konwersje nie zostały utracone (mówi nam o tym wartość 1 na przedostatnim bicie).

Ostateczny wynik odczytu, który zwracany jest przez funkcję ADCRead(0) ma postać:

```
ADC_Data = ( regVal >> 6 ) & 0x3FF;
```

przesuwamy na początek zmiennej wynik konwersji i zerujemy bity o numerach większych niż 10 (bo tyle bitów zajmuje wynik konwersji).

4.4.1 Algorytm sukcesywnej aproksymacji *ADC* [2]

Algorytm sukcesywnej aproksymacji jest w istocie sprzętową wersją przeszukiwania binarnego. Rejestr SAR (*Successive Approximation Register*), do którego zapisywana bedzie odczytana z konwertera wartość jest pierwotnie zainicjalizowany samymi zerami, zaś najbardziej znaczący bit jest przestawiany na wartość 1. Następnie wartość rejestru jest konwertowana na sygnał analogowy, po czym komparator porównuje napięcie wejściowe z konwertera z napięciem odpowiadającym ciągowi bitów z rejestru SAR. Jeżeli napięcie wejściowe w konwerterze (czyli sygnał otrzymywany dzięki ustawieniu pokrętła na odpowiedniej pozycji) przekracza wartość napięcia wygenerowanego na podstawie rejestru – bit wiodący pozostaje w stanie 1, w przeciwnym razie przypisana zostaje mu wartość 0. Następnie analogiczne postępowanie zostaje przeprowadzane dla sukcesywnego bitu. Procedura powtarzana jest dziesięciokrotnie, do zapełnienia całego rejestru.

4.5 SSP/SPI

Serial Peripheral Interface / Synchronous Serial Port, szeregowy interfejs SSP jest używany, by komunikować się z wyświetlaczami. W tym przypadku używamy protokołu SPI. Komunikacja przebiega synchronicznie poprzez trzy równoległe linie. Są nimi

- *MOSI* - *Master Input Slave Output* pozwala na wysyłanie danych do zewnętrznego układu (w naszym wypadku wyświetlaczy);
- *MISO* - *Master Output Slave Input* służy do pobierania danych z układu zewnętrznego;
- *SCLK* - *Serial CLocK* zegar taktujący – precyzyjniej sygnał zegarowy.
- *SSEL* - *Slave SELect* służy do ustalenia które urządzenie jest slave.

Najpierw podłączamy zegar systemowy do SSP (przez szynę AHB) w następujący sposób [1]:

```
LPC_SYSCON->PRESETCTRL |= (0x1<<0); // zaprzestań resetowania SSP
LPC_SYSCON->SYSAHBCLKCTRL |= (1<<11); // włącz zegar dla SSP
```

Dalej, ustawiana jest częstotliwość zegara przez jako odpowiednia część częstotliwości zegara peryferalnego:

```
LPC_SYSCON->SSPCLKDIV = 0x02; //dzielnik zegara peryferalnego
```

Następnie w rejestrach kontroli wejścia/wyjścia ustawiamy linie MISO i MOSI na port 0, piny 8 i 9

```
LPC_IOCON->PIO0_8   |= 0x01; // MISO
LPC_IOCON->PIO0_9   |= 0x01; // MOSI
```

Kolejnym krokiem jest ustawienie głównej płytki jako master – 2 bit portu 0 GPIO jest wybierany jako wyjście i ustawiany jest na nim stan wysoki.

Następnie ustawiamy linię SCKL - wartość 1 na porcie 2 pin 11.

Aby można było użyć zegara na tym pinie, należy także wybrać funkcję 1 przez rejestr lokacji SCK

```
LPC_IOCON->SCKLOC = 0x01;
```

Następnie ustawiamy DSS (Data Size Select) na transmisję 8-bitową, format ramki na SPI, polarność zegara, jego fazowość, oraz SCR (Serial Clock Rate) na 15 – ilość sygnałów z preskalera na bit na szynie

```
LPC_SSP->CR0 = 0x0707;
```

Ustawiamy SSP preskaler

```
LPC_SSP->CPSR = 0x2;
```

Ostatecznie częstotliwośc SPI to 30MHz.

Następnie w kontrolerze przerwań włączamy przerwania SSP.

Transmisja po SPI (dla każdego bajtu informacji) polega kolejno na: odczekaniu aż zniknie status "zajęty" (BSY) w rejestrze kontrolnym `LPC_SSP->SR`, a następnie przesłniu do rejestru danych `LPC_SSP->DR` jednego bajtu z bufora.

4.6 I^2C

I2C (Inter-Integrated Circuit) jest szeregową, dwukierunkową magistralą służącą przesyłowi danych. Nasz program wykorzystuje ją do komunikacji z akcelerometrem, czujnikiem światła, oraz expanderem z diodami.

Program inicjalizuje I2C w trybie MASTER. W celu inicjalizacji I2C wykonujemy następujące czynności: W rejestrze SYSCON PRESETCTRL ustawiamy pierwszy bit, aby wyłączyć resetowanie I2C. W rejestrze SYSCON SYSAHBCLKCTRL ustawiamy piąty bit, aby I2C korzystało z zegara systemowego. W rejestrach IOCON PIO0_4 oraz PIO0_5 ustawiamy zerowy bit i resetujemy kolejne pięć bitów.

Ma to na celu ustawić funkcjonalność pinów na odpowiednio SCL i SDA wykorzystywane przez I2C. W rejestrze I2C CONCLR ustawiamy bity drugi, trzeci, piąty, i szósty, co zeruje flagi rejestru I2C CONSET, przygotowując go do użycia.

Ustawiamy wartości rejestrów I2C SCLL oraz SCLH na 384. Efektywnie kontrolują one czas trwania i okres impulsów SCL używanych do synchronizacji przesyłu danych. Ustawienie ich na jednakową wartość skutkuje współczynnikiem wypełnienia impulsu równym 50%. Otrzymana częstotliwość wynosi

$$PCLK/(SCLL + SCLH) = 72MHz/768 = 93.75kHz,$$

czyli w przybliżeniu pożądane 100 kHz.

Włączamy przerwania z I2C poprzez ustawienie bitu ósmego w rejestrze NVIC ISER1. Wreszcie w rejestrze I2C CONSET ustawiamy szósty bit, co włącza interfejs I2C i pozwala na jego wykorzystanie.

Podczas przesyłu danych w dowolnym kierunku program wykonuje następujące operacje: W rejestrze I2C CONSET ustawiamy piąty bit, tj. flagę startu, sygnalizującą gotowość do transferu, i oczekujemy na wywołane przezeń przerwanie.

Wykorzystujemy przerwania do przesyłu kolejnych bajtów danych. Bajty są pisane do lub odczytywane z rejestru I2C DAT. Po każdej obsłudze przerwania resetujemy flagę przerwania. Odczyt lub zapis powtarzamy do momentu wyczerpania buforu przeznaczonego na wiadomość. W rejestrze I2C CONSET ustawiamy czwarty bit, tj. flagę stopu.

4.7 Wyświetlacz siedmiosegmentowy

W zaprojektowanej grze wyświetlacz siedmiosegmentowy służy jako element dekoracyjny i nie ma wpływu na rozgrywkę. Przez cały czas trwania rozgrywki wyświetla on kolejne litery napisu.

1. Inicjalizacja wyświetlacza siedmiosegmentowego polega na ustawieniu kierunku w pierwszym porcie GPIO. Na 11 bicie ustawiana jest wartość 1, co oznacza kierunek wyjściowy

```
LPC_GPIO1->DIR |= (0x1<<11);
```

a następnie ustawienie na tym bicie wartości 1, przy czym tak zainicjalizowany wyświetlacz nie wyświetla jeszcze żadnego znaku.

```
GPIOShadowPort1 |= (1<<11);
```

2. Deklarujemy wartość zmiennej letters_speed, która opisywać będzie co ile iteracji głównej pętli programu nastąpi zmiana wyświetlanego znaku (poprzedzona jednoiteracyjną przerwą):

```
const uint8_t letters_speed = 10;
```

Tablica znaków do wyświetlania przez wyświetlacz siedmiosegmentowy ma następującą deklarację:

```
const uint8_t letters[] = "NO STEP ON SNECC ";
```

3. W pojedynczej iteracji głównej pętli programu wykonujemy następujące kroki:

3.1 Upewniamy się, że zmienna kontrolująca wartość wyświetlanej litery letters_state nie przekroczy wartości **sizeof**(letters)·letters_speed.

```
letters_state %= letters_size * letters_speed;
```

3.2 Wybieramy instrukcją warunkową czy obecnie wyświetlany jest znak z tablicy letters, czy też następuje przerwa, która czyni napis czytelniejszym dla użytkownika. Zestaw stosowanych w programie znaków pozwala wyświetlić każdy z nich na wyświetlaczu siedmiosegmentowym. Zero przekazywane jako drugi argument funkcji sprawia, że nie wchodzimy w tryb surowego przesyłu danych (tryb w którym korzystamy z funkcji led7seg_setChar sprawia, że litery odczytywane są jako znaki ASCII):

```
led7seg_setChar(letters_state % letters_speed ?
letters[letters_state / letters_speed] : ' ', FALSE);
```

Funkcja ta przy pomocy metody SSPSend przekazuje stablicowane ośmiobitowe wartości całkowite odpowiadające poszczególnym wyświetlanym znakom. Przykładowo, wartością odpowiadającą literze '**S**' jest 0x12. Przed przesłaniem przez SSP jakichkolwiek danych musimy jednak zmienić wartość jedenastego bitu pierwszego portu GPIO na 0.

```
GPIOShadowPort1 &= ~(1<<bitPosi);
```

Po zakończeniu przesyłania zmiennej zostaje on ponownie przywrócony do stanu 1 tak samo jak przy inicjalizacji.

3.3 Zwiększamy wartość letters_state o jeden.

```
letters_state++;
```

4.8 Wyświetlacz OLED

Komunikacja przebiega za pomocą interfejsu SPI, a przesyłane dane są 8-bitowe. Aby wysłać dane należy ustawić kontroler ekranu OLED jako slave (wartość 0 na lini SSEL(port 0 bit 2). Interpretacja przesyłanych zależy od sygnału ustawionego na 7 bicie portu 2. W przypadku gdy jest on 1 przesyłane dane są interpretowane jako dane, w przeciwnym wypadku jako komendy.

Wyświetlacz składa się z 8 stron (każda to fragment ekranu o pełnej szerokości i wysokości 8 pikseli), z których każda ma 136 bajtów (maksymalna szerokość ekranu obsługiwana przez kontroler), a każdy bajt opisuje osiem pikseli o kolejnych współrzędnych y. Ustawienie koloru pojedynczego piksela polega na wysłaniu jako komenda adresu strony (`0xB0`...`0xB7`), następnie adresu bajtu `0bABCDEFGH` w postaci kolejnych bajtów `0b0000EFGH` i `0b0001ABCD`. Następnie jako dane wysyłamy pod ustawiony adres maskę, gdzie wartość 1 oznacza biały piksel a wartość 0 czarny.

4.9 Przerwania (Interrupts)

Przerwanie to sygnał dla procesora pochodzący ze sprzętu bądź oprogramowania, informujący o zajściu zdarzenia wymagającego natychmiastowego przetworzenia. Procesor, otrzymawszy taki sygnał, zawiesza wykonywanie obecnego kodu, zachowuje swój stan, i przechodzi do przetwarzania przerwania. Nasz program wykorzystuje przerwania z zegara (*timer*) w celu odtwarzania melodii niezależnie od algorytmu gry.

Korzystamy z 32-bitowego timera 1 (CT32B1) o preskalerze ustawionym na 1 mikrosekundę. Podczas inicjalizacji timera opisanej szczegółowo osobno w poświęconej mu sekcji, w celu włączenia pochodzących zeń przerwań ustawiamy bit zerowy i pierwszy w rejestrze *Match Control Register* TMR32B1 MCR (tj. wartość liczbową `0b11`), co odpowiednio oznacza, że przerwania mają być generowane przy osiągnięciu wartości MR0, oraz że timer, przy osiągnięciu tej wartości, ma być resetowany do zera. Następnie włączamy w kontrolerze przerwań (NVIC, *Nested Vectored Interrupt Controller*) obsługę przerwań z 32-bitowego timera 1 poprzez ustawienie odpowiadającego mu dwunastego bitu w rejestrze ISER1.

Przerwania użytego timera obsługuje funkcja TIMER32_1_IRQHandler, której adres jest przechowywany w zerowym bloku pamięci RAM wraz z innymi procedurami obsługi zdarzeń. Obsługa tego przerwania zaczyna się w naszym przypadku od ustawienia zerowego bitu rejestru TMR32B1 IR w celu zresetowania przerwania. W ogólnym przypadku należałoby odczytać ów rejestr, aby ustalić rodzaj zdarze-

nia, i następnie ustawić bit w zależności od tego rodzaju, jednak nasz program generuje tylko jeden rodzaj przerwań z timera, więc byłoby to zbędne. Potem następuje właściwa część obsługi służąca odegraniu melodii. Ta część ustawia również rejestr MR0 na ilość czasu daną w mikrosekundach, po której powinno zajść kolejne przerwanie, co zależy od częstotliwości obecnie granej nuty lub długości obecnej pauzy.

4.10 Głośnik (GPIO) – Melodia

Odgrywanie melodii realizowane jest w handlerze przerwania 32 bitowego timera nr 1 i kontrolowane za pomocą zmiennej `is_play_song`. Jeżeli jej wartość logiczna odpowiada prawdzie, zaczyna się przetwarzanie. Melodia jest zapisana jako ciąg znaków, gdzie każde kolejne trzy oznaczają wysokość dźwięku (zakodowana jako litery a-g i A-G), długość jego trwania (zakodowana jako znak 0-9) i długość następującej po dźwięku pauzy (zakodowana jako jeden ze znaków [+.-_]). Z ciągu odczytywana jest wysokość dźwięku, dekodowana jest jako wysokość dźwięku w hercach (między 262 a 988 Hz), długość dźwięku w milisekundach (między 0 a 1800 ms) oraz długość pauzy w milisekundach (między 0 a 30 ms). Następnie rejestr `MR0` jest ustawiany na połowę okresu dźwięku o danej częstotliwości w mikrosekundach

$$1000000[\mu s/s]/(pitch[Hz] \cdot 2),$$

licznik ustawiany jest ilość zmian sygnału w danym dźwięku

$$(duration[ms] * 1000[\mu s/ms])/half\_period[ms].$$

Następnie w związku z przestawieniem rejestru `MR0` funkcja jest wywoływana co połowę okresu i za każdym razem nastepuje zmiana sygnału przekazywanego na głośnik – w ten sposób odtwarzany jest dźwięk. Po zakończeniu odtwarzania dźwięku `MR0` zostaje ustawiony na długość pauzy w milisekundach, a przy następnym wywołaniu (po zakończeniu trwania pauzy) rejestr `MR0` zostaje ustawiony na 1000 [ms] i algorytm oczekuje na kolejny sygnał odtworzenia melodii.

4.11 Akcelerometr

Akcelerometr to urządzenie zdolne do wykrycia i pomiaru oddziałującego nań wektora przyspieszenia. W proponowanym przez nas programie służy on jako alternatywna dla joysticka metoda sterowania procesją. Gracz przechylając w odpowiednim kierunku mikrokomputer sprawia, że procesja skręca we wskazaną stronę.

Zastosowany w naszym układzie akcelerometr MMA7455L jest inicjowany przez zdefiniowanie trybu pomiaru i zakresu poprzez przypisanie (przy pomocy I^2C) do rejestru MCR (*Mode Control Register* o adresie `0x16`) wartości 1 na bit zerowy i drugi (przejście w *Measurement Mode*, tj. tryb pomiarowy przy zakresie 2 g, przekładający się na większą precyzję pomiaru [6]).

Przy odczycie najpierw sprawdzamy rejestr statusu akcelerometru o adresie `0x09`. Zerowy bit tego rejestru – oznaczony przez DRDY – informuje nas, czy dane są gotowe do odczytania. Jeżeli tak, to z rejestru o adresie `0x06` pobierany jest odczyt 8 bitów opisujących składową X wektora przyspieszenia. Następnie odczytywane są wartości z rejestrów `0x07` i `0x08` odnoszące się odpowiednio do składowych Y i Z. Warto zauważyć, że współrzędna Z służy jedynie do autodiagnostyki urządzenia. Nie bierze ona udziału w sterowaniu procesją w sposób bezpośredni.

Następnie, jeżeli odczytane wartości na współrzędnych X oraz Y przekraczają (co do modułu) wartość $\frac{10}{64}$ g, to ustawiane są odpowiednie (1/2 dla kierunków góra/dół, 3/4 dla kierunków prawo/lewo) bity zmiennej sterującej, która (po wykonaniu bitowej operacji OR ze zmienną przechowującą odczyt z joysticka) zostaje przekazana do funkcji odpowiadającej za przetwarzanie procesji.

4.12 Czujnik Światła

Czujnik światła przyłączony jest do płytki magistralą I^2C [5]. Pod adres szyną I^2C `(0x44 << 1)` przesyłane są wartości (kolejno) {`0x00`, `(1 << 7)`}. Powoduje to ustawienie wartości 1 na siódmym bicie rejestru Command Register. Bit ten odpowiada za uruchomienie stosownego przetwornika analogowo-cyfrowego w czujniku.

Ustawiamy następnie wartości 0 i 1 na (odpowiednio) bitach o numerach 2 i 3. Sprawia to, że zakres odczytu czujnika zmieni się z domyślnych 1000 luksów do 4000 luksów.

Odczyt czujnika światła znajduje się w rejestrach LSB_sensor (rejestr o adresie `0x04`) – odpowiadający za dolną połowę odczytu i MSB_sensor (rejestr o adresie `0x05`), w którym znajduje się bajt górnej połowy odczytu. Przekazywany jest on do programu przez I^2C, a konkretnie do zmiennej nazwanej `data`. Ostateczny odczyt ma wartość

```
3892 * data / (1 << 16)
```

i wyrażony jest w luksach.

W programie czujnik światła został zastosowany do zrestartowania gry w momencie, gdy odczyty natężenia oświetlenia spadną poniżej 100 luksów (np. w wy-

niku zakrycia czujnika światła palcem).

5 Analiza FMEA

Możliwa awaria	Prawdopodobieństwo	Reakcja	Istotność	Iloczyn
Uszkodzenie wyświetlacza LCD	Niewielkie 0.05	Nie	Krytyczna 10	0.5
Uszkodzenie czujnika światła	Małe 0.08	Nie	Krytyczna 10	0.8
Uszkodzenie joysticka	Średnie 0.2	Tak	Wysoka 8	1.6
Uszkodzenie akcelerometru	Średnie 0.3	Tak	Średnia 6	1.8
Uszkodzenie wyświetlacza 7segmentowego	Znikome 0.01	Nie	Zerowa 0.1	0.001
Uszkodzenie pokrętła ADC	Małe 0.15	Nie	Mała 2	0.3

5.1 Uszkodzenie wyświetlacza LCD

Możliwe uszkodzenie wyświetlacza LCD spowoduje całkowitą niegrywalność. Nawet jeżeli możliwa byłaby autodiagnostyka tego przypadku, uznano że urządzenie miałoby nikłe szanse na sensowną reakcję na wypadek tego typu awarii.

5.2 Uszkodzenie czujnika światła

Uszkodzenie czujnika światła może generować problemy dwojakiego rodzaju, z których tylko jeden ma dla nas istotne znaczenie. W przypadku gdy odczytywane wartości są zbyt wysokie (tj. czujnik nie wykrywa zmniejszenia intensywności oświetlenia) problem nie jest krytyczny dla działania układu – niemożliwe staje się wówczas jedynie zrestartowanie gry. Uszkodzenie czujnika, które sprawi, że odczytywane wartości natężenia światła będą poniżej 100 luksów uczyni Procesję całkowicie niegrywalną, z powodu nieprzerwanego restartowania gry. Niestety w tej sytuacji nie zostały podjęte żadne działania naprawcze.

5.3 Uszkodzenie akcelerometru

Uszkodzenie akcelerometru może powodować błędy w zachowaniu się procesji – mianowicie mogłoby sprawić, że kierunek poruszania się procesji zostanie ustalony i niezależnie od wykonywanych przez gracza manewrów gra nieprzerwanie próbować będzie odczytać wskazania akcelerometra i poruszać procesją zgodnie z tymi odczytami.

5.3.1 Rozpoznanie

Przy uruchomieniu zakładamy, że płytka jest ustawiona w pozycji horyzontalnej na płaskiej powierzchni i oddziałuje na nią wektor przyspieszenia grawitacyjnego $1g$ skierowany pionowo w dół, tj. o współrzędnych $(0, 0, -1)$. Następnie obliczany jest kwadrat długości obecnie odczytanego przez akcelerometr wektora przyspiesznia. Jeżeli wartość normy odczytanego wektora przyspieszenia X przy przyjmowanym zakresie akcelerometru (wynoszącym $2g$) mieści się w przedziale

$$0.78125\ g = \sqrt{\frac{2500}{4096}}\ g\ < \|X\|\ <\ \sqrt{\frac{6000}{4096}}\ g \simeq 1.210307\ g,$$

odczyty uznajemy za poprawne. W przypadku wystąpienia odczytu wykraczającego poza ten zakres, program uznaje akcelerometr za uszkodzony.

5.3.2 Reakcja

Zmienna opisująca poprawność działania akcelerometru zostaje ustawiona na wartość $BROKEN = 0$. W wyniku tego kolejne wykonania głównej pętli programu nie uwzględniają już odczytywania wartości wskazywanych przez akcelerometr. Wprawdzie uniemożliwia to dalsze sterowanie przy pomocy akcelerometru (przynajmniej do czasu ponownego uruchomienia urządzenia), aczkolwiek nadal możliwe jest granie w Procesję przy pomocy joysticka.

5.4 Uszkodzenie joysticka

Uszkodzenie manipulatora drążkowego może spowodować odczytywanie wskazań, które nie zmieniają się lub zmieniają się w sposób całkowicie losowy. W naszej analizie skupiliśmy się na pierwszym przypadku, który uznaliśmy za dużo bardziej prawdopodobny. Wystąpienie takiej awarii sprawiłoby, że procesja (niezależnie od życzenia użytkownika i sterowania akcelerometrem) poruszałaby się w jednym kierunku, skutecznie uniemożliwiając zabawę.

5.4.1 Rozpoznanie

W zmiennej *prev_state* przechowywany jest poprzedni odczyt z joysticka. Jeżeli wskazania nie zmieniają się przez ustalony czas (odpowiadający stu wykonaniom pętli), uznajemy, że nastąpiła awaria. Wyjątek od tej reguły stanowi sytuacja, gdy wskazywane jest położenie joysticka w pozycji 0 (nienaciśniętej).

5.4.2 Reakcja

Gdy rozpoznana zostaje awaria urządzenia peryferyjnego manipulatora drążkowego następuje zaprzestanie odczytywania wskazań tegoż manipulatora. Wznowienie odczytu nie następuje – aby grać dalej należy korzystać z akcelerometra. W przypadku gdy chcemy korzystać z joysticka należy zrestartować urządzenie.

5.5 Uszkodzenie wyświetlacza 7segmentowego

Uszkodzenie wyświetlacza siedmiosegmentowego ma jedynie znaczenie kosmetyczne i nie wpływa w żaden sposób na działanie całości programu. W związku z tym nie zużywamy mocy obliczeniowej na sprawdzanie poprawności jego działania ani nie reagujemy na jego ewentualne uszkodzenie.

5.6 Uszkodzenie pokrętła ADC

Uszkodzenie pokrętła nie ma dużego wpływu na rozgrywkę – sprawia jedynie, że prędkość (a zatem poziom trudności) gry zostaje ustalona i użytkownik nie będzie miał możliwości jej modyfikacji. W przypadku ustalenia zbyt dużej prędkości gra może stać się zbyt trudna, by być satysfakcjonująca.

Niestety, nie zostały uwzględnione metody autodiagnostyki dla tego rodzaju awarii.

6 Wykorzystane noty katalogowe, dokumentacja i literatura

Bibliografia

[1] *UM10375 LPC1311/13/42/43 User manual*, 21 June 2012, **Rev. 5**,

[2] *Understanding SAR ADCs: Their Architecture and Comparison with Other ADCs*, 02 October 2001, **Maxim Integrated Products, Inc.**

[3] *LPC1343 ADC programming tutorial*, 18 November 2017, **Umang Gajera**

[4] *PCA9532, 16-bit I2C-bus LED dimmer*, 22 August 2016, **Rev. 4.1, Product data sheet**

[5] *ISL29003 Data Sheet FN7464.6*, 17 November 2011, **Intersil, Mouser Electronics**

[6] *±2g/±4g/±8g Three Axis Low-g Digital Output Accelerometer*, 12/2009, **Document Number: MMA7455L Rev 10**