# Odtwarzacz WAV – dokumentacja

*Mikrokontroler LPC1769*

*Grupa 6-2, IS FTIMS/4, 2023*

*Marcin Mazur 242467 – lider grupy*
*Piotr Płeska 242499*
*Igor Bobrukiewicz 242361*
*Maciej Perdaszek 242490*

**Udział w pracy poszczególnych osób:**
Igor Bobrukiewicz – 40%
Marcin Mazur – 20%
Piotr Płeska – 20%
Maciej Perdaszek – 20%

| Lp. | Funkcjonalność | Urządzenia | Osoba |
|---|---|---|---|
| 1. | Odtwarzanie dźwięku | DAC, Timer, Interrupt | Marcin Mazur |
| 2. | Przycisk odtwarzania | GPIO | Maciej Perdaszek |
| 3. | Przeskakiwanie utworów | ACC, I2C | Maciej Perdaszek |
| 4. | Zmiana głośności | ADC, wzmacniacz | Piotr Płeska |
| 5. | Wyświetlanie postępu odtwarzanego utworu | I2C, PCA | Piotr Płeska |
| 6. | Wyświetlanie numeru utworu na ekranie | SPI, OLED | Igor Bobrukiewicz |
| 7. | Odczyt plików WAV z karty | SPI, MMC | Igor Bobrukiewicz |

# Spis treści

# Instrukcja użytkownika

## Wymagania

Odtwarzacz umożliwia odtwarzania plików dźwiękowych zapisanych w formacie WAV, które odczytywane są z karty SD o systemie plików FAT.

Specyfikacja akceptowalnego pliku dźwiękowego:

- Format: .wav,
- Nazwa: do 8 znaków
- Próbkowanie: 8000 Hz
- Rozdzielczość bitowa: 8 bit
- Dźwięk: mono

## Interfejs użytkownika

Interakcja z użytkownikiem jest zrealizowana za pomocą **przycisku** pełniącego funkcje *START, PAUZA, STOP*, **akcelerometru** służącego do *zmieniania plików*, oraz **potencjometru** umożliwiającego zmianę głośności odtwarzanego pliku. Nazwa i numer wybranego pliku wyświetlają się na **ekranie OLED**. Postęp odtwarzania utworu można śledzić obserwując **diody LED**. Pliki odczytywane są z umieszczonej w interfejsie **karty SD**.

### Przycisk wielofunkcyjny

Pojedyncze naciśnięcie przycisku **SW3** powoduje rozpoczęcie, wstrzymanie, bądź wznowienie odtwarzania dźwięku.

Dłuższe przytrzymanie przycisku powoduje zatrzymanie odtwarzania dźwięku i umożliwia zmianę odtwarzanego pliku.

### Potencjometr

Umożliwia zmianę głośności odtwarzanego dźwięku podczas jego odtwarzania.

### Akcelerometr

Gdy nie jest odtwarzana żadna piosenka, możliwe jest zmienianie odtwarzanego pliku. Przechylenie urządzenia w prawo zmienia odtwarzany plik na kolejny w kolejce, przechylenie urządzenia w lewo zmienia odtwarzany plik na poprzedni w kolejce.

**Ekran OLED**

Na ekranie wyświetla się numer i nazwa aktualnie wybranego pliku.

**Pasek LED**

Na pasku złożonym z 8 diod LED można na bieżąco obserwować postęp odtwarzanego utworu. Wszystkie diody LED zapalone oznaczają, że odtwarzanie utworu dobiegło końca.

**Interfejs SD/MMC**

Pliki dźwiękowe WAV odczytywane są z karty SD w formacie FAT umieszczonej w interfejsie SD/MMC.

## Opis algorytmu

Po wczytaniu programu następuje inicjalizacja wszystkich używanych urządzeń, oraz wyświetlenie napisu powitalnego na ekranie OLED.

Następnie wykrywana jest karta SD, a po pomyślnym wykryciu następuje odczyt jej zawartości. Zapamiętywane są wyłącznie dane dotyczące plików o rozszerzeniu .wav. Ponadto nazwy plików zapamiętywane są w przyjaznej użytkownikowi formie – bez rozszerzenia. Możliwy jest odczyt do 20 plików muzycznych.

Po operacjach wstępnych program działa bez końca, oczekując sygnałów od użytkownika.
W 100 milisekundowych interwałach sprawdzana jest za pomocą żyroskopu aktualna pozycja w przestrzeni urządzenia, a w wypadku wystarczającego przechylenia urządzenia – następuje zmiana pliku. Jego numer i nazwa wyświetlane są na ekranie OLED, plik jest wstępnie wczytywany i obliczane są wszystkie wymagane wielkości do jego poprawnego odtworzenia.

Pojedyncze naciśnięcie przycisku **SW3** powoduje rozpoczęcie odtwarzania danego pliku. Kolejne bajty odtwarzane są w przerwaniach. Kolejne fragmenty plików wczytywane są do dwóch buforów.

Wraz z postępem odtwarzania zapalane są kolejne diody LED.

Ponowne wciśnięcie przycisku wstrzyma odtwarzanie, a przytrzymanie – zatrzyma i pozwoli na przejście do kolejnej piosenki.

# Funkcjonalności

## 1. Odtwarzanie dźwięku

### DAC

Odtwarzanie dźwięku zrealizowane jest w głównej mierze dzięki **DAC** – przetwornikowi analogowo-cyfrowemu. Umożliwia on 10 bitową konwersję wartości cyfrowych na analogowe wyjścia napięciowe, dzięki czemu urządzenia cyfrowe mogą uzyskać połączenie z systemami analogowymi, takimi jak głośniki, słuchawki, monitory i inny sprzęt audio lub wideo.

#### Opis pinów D/A

| PIN | Typ | Opis |
|---|---|---|
| AOUT | Analog Output | Po ustawieniu rejestru DACR wartość wyjściowa napięcia jest równa VALUE ×((VREFP - VREFN)/1024)+VREFN. |
| VREFP,VREFN | Referencja Napięciowa | Piny te zapewniają poziom napięcia odniesienia dla przetworników ADC i DAC. |
| VDDA,VSSA | Zasilanie i Uziemienie | Te wartości powinny być takie same jak Vdd(napięcie przykładane do drenu tranzystora [Uziemienie]) i Vss(napięcie przykładane do źródła transyztora[Zasilanie]) |

Wstępna konfiguracja DAC odbywa się poprzez ustawienie *pinu P0.26* na *AOUT* do działania jako pin wyjściowy *DAC* poprzez ustawienie rejestru *PINSEL*.

```
PINSEL_CFG_Type PinCfg;
PinCfg.Funcnum = 2;
PinCfg.OpenDrain = 0;
PinCfg.Pinmode = 0;
PinCfg.Pinnum = 26;
PinCfg.Portnum = 0;
PINSEL_ConfigPin(&PinCfg);
```

Kolejnym krokiem jest ustawienie maksymalnej wartości prądu płynącego przez przetwornik na *750 uA* poprzez ustawienie *bitu BIAS* na *0*.

```
void DAC_Init(LPC_DAC_TypeDef *DACx)
{
	CHECK_PARAM(PARAM_DACx(DACx));
	/* Set default clock divider for DAC */
```

```
	// CLKPWR_SetPCLKDiv (CLKPWR_PCLKSEL_DAC, CLKPWR_PCLKSEL_CCLK_DIV_4);
	//Set maximum current output
	DAC_SetBias(LPC_DAC,DAC_MAX_CURRENT_700uA);
```

```
typedef enum
{
	DAC_MAX_CURRENT_700uA = 0,		/*!< The settling time of the DAC is 1 us
max and the maximum current is 700 uA */
	DAC_MAX_CURRENT_350uA			/*!< The settling time of the DAC is 2.5
us and the maximum current is 350 uA */
} DAC_CURRENT_OPT;
```

```
void DAC_SetBias (LPC_DAC_TypeDef *DACx,uint32_t bias)
{
	CHECK_PARAM(PARAM_DAC_CURRENT_OPT(bias));
	DACx->DACR &=~DAC_BIAS_EN;
	if (bias  == DAC_MAX_CURRENT_350uA)
	{
		DACx->DACR |= DAC_BIAS_EN;
	}
```

### Rejestry DAC

| Nazwa | Opis | Dostęp | Reset | Adres |
|---|---|---|---|---|
| DACR | Przechowuje wartość cyfrową która będzie konwertowana na wartość analogową. | R/W | 0 | 0x4008 C000 |
| DACCTRL | Rejestr kontrolny. Kontroluje DMA i operacje zegara. | R/W | 0 | 0x4008 C004 |
| DACCNTVAL | Zawiera wartość przeładowania dla DMA/Przerwania zegara. | R/W | 0 | 0x4008 C008 |

W celu konwersji danych cyfrowych na sygnał analogowy piszemy wartość amplitudy generowanego tonu do bitów *od 15 do 6* włącznie w *rejestrze DACR*.

### Opis rejestru DACR

| Bity | Symbol | Wartość | Opis | Reset |
|---|---|---|---|---|
| 0-5 | - | | Bity zarezerwowane dla przyszłych DAC o większej zozdzielczości. | N/A |
| 6-15 | VALUE | | Po ustawieniu rejestru DACR wartość wyjściowa napięcia jest równa VALUE ×((VREFP - VREFN)/1024)+VREFN. | 0 |
| 16 | BIAS | 0 | Ustala próbę DAC na 1 us oraz maksymalny prąd do 700uA. Pozwala to na osiągnięcie częstotliwości 1MHz. | 0 |
| | | 1 | Ustala próbę DAC na 2.5 us oraz maksymalny prąd do 350uA. Pozwala to na osiągnięcie częstotliwości 400kHz. | |

```
void DAC_UpdateValue (LPC_DAC_TypeDef *DACx,uint32_t dac_value)
{
	uint32_t tmp;
	CHECK_PARAM(PARAM_DACx(DACx));
	tmp = DACx->DACR & DAC_BIAS_EN;
	tmp |= DAC_VALUE(dac_value);
	// Update value
	DACx->DACR = tmp;
}
```

## Przerwania

Przerwania są podstawowym mechanizmem występującym w każdym procesorze. Ogólna zasada działania jest taka, że w pamięci procesora przechowywany jest wektor przerwań - czyli tablica odwzorowująca numery przerwań na adresy funkcji, zwanych procedurami obsługi przerwań.

Przerwanie może zostać wygenerowane na skutek pewnych zdarzeń w urządzeniu. Przetwarzany wątek programu zostaje przerwany, po czym następuje skok do procedury obsługi danego przerwania, której to adres jest uprzednio odczytywany ze wspomnianego wcześniej wektora przerwań.

Programista może napisać swoje własne procedury obsługi przerwań i umieścić ich adresy w tym wektorze. W przypadku wykorzystywanego układu istnieje także możliwość nadawać priorytety poszczególnym przerwaniom. Przerwania mogą być także generowane programowo.

Przerwanie oczekujące(Pending Interrupt) to po prostu przerwanie, które wystąpiło, jest włączone, ale nie przeszło przez proces nadawania priorytetów przerwań, aby jego program obsługi został wykonany.

#### Opis przykładowych rodzajów przerwań

| ID Przerwania | Numer Wyjątku | Przesunięcie Wektora | Funkcjonalność | Flagi |
|---|---|---|---|---|
| 0 | 16 | 0x40 | WDT | Watchdog Interupt |
| 1 | 17 | 0x44 | Timer 0 | Match(MR0,MR1)<br>Capture(CR0,CR1) |
| 2 | 18 | 0x48 | Timer 1 | Match(MR0,MR1,MR2)<br>Capture(CR0,CR1) |

Aby przerwanie było generowane, musi zostać ono odblokowane w dwóch miejscach:

- *rejestry kontrolne danego urządzenia peryferyjnego odpowiedzialnego za generowanie przerwania*
- *rejestry systemowego kontrolera przerwań (NVIC)*

W przypadku omawianego programu wykorzystano jedynie z przerwania *Timera 1*, choć możliwości mikrokontrolera pozwalały na użycie aż 32 rodzajów przerwań.

### Rejestry NVIC(Nested Vector Interrupt Controller)

| Nazwa | Opis | Dostęp | Reset | Adres |
|---|---|---|---|---|
| ISER0,ISER1 | Interrupt Set Enable Register. Rejestry odpowiedzialne za włączenie przerwania. | RW | 0 | ISER0 - 0xE000 E100<br>ISER1 - 0xE000 E104 |
| ICER0,ICER1 | Interrupt Clear Enable Register. Rejestry odpowiedzialne za wyłączanie przerwania. | RW | 0 | ICER0 - 0xE000 E180<br>ICER1 - 0xE000 E184 |
| ISPR0,ISPR1 | Interrupt Set Pending Register. Zmienia stan przerwania na oczekiwany. | RW | 0 | ISPR0 - 0xE000 E200<br>ISPR1 - 0xE000 E204 |
| ICPR0,ICPR1 | Interrupt Clear Pending Register. Zmienia stan przerwania na nie oczekiwany. | RW | 0 | ICPR0 - 0xE000 E280<br>ICPR1 - 0xE000 E284 |
| IABR0,IABR1 | Interrupt Active Register. Pozwala stwierdzić czy przerwanie jest aktywne. | RO | 0 | IABR0 - 0xE000 E300<br>IABR1 - 0xE000 E304 |
| IPR0-IPR8 | Interrupt Priority Register. Rejestry pozwalające ustawienie priorytetu przerwania | RW | 0 | IPR0 - 0xE000 E400<br>IPR1 - 0xE000 E404<br>IPR2 - 0xE000 E408<br>IPR3 - 0xE000 E40C<br>IPR4 - 0xE000 E414<br>IPR5 - 0xE000 E418<br>IPR6 - 0xE000 E41C<br>IPR7 - 0xE000 E420<br>IPR8 - 0xE000 EF00 |

Aby odblokować wykorzystywane przerwania w *NVIC'u*, należy ustawić wartość *bitu 2* rejestru *ISER0*, aby włączyć przerwania *Timera 1*.

```
TIMER1_IRQn = 2,         /*!< Timer1 Interrupt

static __INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
  NVIC->ISER[((uint32_t)(IRQn) >> 5)] = (1 << ((uint32_t)(IRQn) & 0x1F)); /*
enable interrupt */
}
```

Aby wyłączyć przerwanie należy ustawić wartość *bitu 2 rejestru ICER*.

```
TIMER1_IRQn = 2,         /*!< Timer1 Interrupt

static __INLINE void NVIC_DisableIRQ(IRQn_Type IRQn)
{
  NVIC->ICER[((uint32_t)(IRQn) >> 5)] = (1 << ((uint32_t)(IRQn) & 0x1F)); /*
disable interrupt */
}
/** @addtogroup CMSIS_CM3_NVIC CMSIS CM3 NVIC
  memory mapped structure for Nested Vectored Interrupt Controller (NVIC)
  @{
 */
```

```
typedef struct
{
  __IO uint32_t ISER[8];                     /*!< Offset: 0x000  Interrupt Set
Enable Register           */
       uint32_t RESERVED0[24];
  __IO uint32_t ICER[8];                     /*!< Offset: 0x080  Interrupt Clear
Enable Register          */
       uint32_t RSERVED1[24];
  __IO uint32_t ISPR[8];                     /*!< Offset: 0x100  Interrupt Set
Pending Register          */
       uint32_t RESERVED2[24];
  __IO uint32_t ICPR[8];                     /*!< Offset: 0x180  Interrupt Clear
Pending Register         */
       uint32_t RESERVED3[24];
  __IO uint32_t IABR[8];                     /*!< Offset: 0x200  Interrupt Active
bit Register           */
       uint32_t RESERVED4[56];
  __IO uint8_t  IP[240];                     /*!< Offset: 0x300  Interrupt
Priority Register (8Bit wide) */
       uint32_t RESERVED5[644];
  __O  uint32_t STIR;                        /*!< Offset: 0xE00  Software Trigger
Interrupt Register     */
}  NVIC_Type;
}
```

## Timer

Timer jest podstawowym urządzeniem peryferyjnym mikrokontrolera, które służy do zliczania impulsów zegara (w rejestrze *Timer Counter - TC*).

Każdy Timer posiada 32-bitowy preskaler, który zlicza impulsy zegara do pewnej wartości (*rejestr Prescale Register - PR*) co pozwala nam na zmniejszenie czasu do odpowiadającej nam wartości. Dopiero kiedy osiągnie właściwą wartość, 32-bitowy licznik Timera zostaje zwiększony.

Ponadto Timer zawiera rejestry *MR0/1/2/3* które są ciągle porównywane z wartością *rejestru TC*. Kiedy w *rejestrze TC* znajduje się to samo co w *rejestrze MR* podejmowana jest czynność.

W zależności od konfiguracji zapisanej w rejestrze *Match Control Register (MCR)*:

- zostaje wygenerowane przerwanie,
- wartość TC może zostać zresetowana (lub cały timer może zostać zatrzymany),
- stan na wybranych pinach procesora (MAT0/3) może ulec zmianie.

### Rejestry Timera

| Nazwa | Opis | Dostęp | Reset | Adres |
|---|---|---|---|---|
| IR | Interrupt Register. IR może zostać zapisany w celu skasowania przerwania. IR można odczytać, aby zidentyfikować, które z ośmiu możliwych źródeł przerwań jest w toku. | RW | 0 | T0IR - 0x4000 4000<br>T1IR - 0x4000 8000<br>T2IR - 0x4009 0000<br>T3IR - 0x4009 8000 |
| TCR | Timer Control Regisetr. TCR służy do sterowania funkcjami licznika czasu. Licznik czasowy można wyłączyć lub zresetować za pomocą TCR. | RW | 0 | T0TCR - 0x4000 4004<br>T1TCR - 0x4000 8004<br>T2TCR - 0x4009 0004<br>T3TCR - 0x4009 8004 |

| | | | | |
|---|---|---|---|---|
| TC | Timer Counter. 32-bitowy TC jest zwiększany co PR+1 cykli PCLK. TC jest kontrolowany przez TCR. | RW | 0 | T0TC - 0x4000 4008<br>T1TC - 0x4000 8008<br>T2TC - 0x4009 0008<br>T3TC - 0x4009 8008 |
| PR | Prescale Register.Gdy wartość PC jest równa tej wartości, następny tick zwiększa TC i zeruje PC. | RW | 0 | T0PC - 0x4000 400C<br>T1PC - 0x4000 800C<br>T2PC - 0x4009 000C<br>T3PC - 0x4009 800C |
| PC | Prescale Counter. 32-bitowy PC jest licznikiem, który jest zwiększany do wartości zapisanej w PR. Gdy wartość w PR zostanie osiągnięta, TC jest zwiększany, a PC jest zerowany. Rejestr PC jest widoczny i kontrolowany przez interfejs magistrali. | RW | 0 | T0PC - 0x4000 4010<br>T1PC - 0x4000 8010<br>T2PC - 0x4009 0010<br>T3PC - 0x4009 8010 |
| MCR | Match Control Register. MRC służy do kontrolowania, czy generowane jest przerwanie i czy TC jest zerowany, gdy wystąpi Mach. | RW | 0 | T0MCR - 0x4000 4014<br>T1MCR - 0x4000 8014<br>T2MCR - 0x4009 0014<br>T3MCR - 0x4009 8014 |
| MR0-3 | Match Register 0-3. MR0 może być aktywowany przez MCR, aby zresetować TC, zatrzymać zarówno TC, jak i PC, i/lub wygenerować przerwanie za każdym razem, gdy MR0 odpowiada TC. | RW | 0 | T0MR0 - 0x4000 4018<br>T1MR0 - 0x4000 8018<br>T2MR0 - 0x4009 0018<br>T3MR0 - 0x4009 8018 |

W przypadku opisywanego urządzenia wykorzystany został wyłącznie *Timer 1*, służący do generowania przerwań z częstotliwością wyznaczaną w procesie obliczeniowym podczas wczytywania pliku dźwiękowego (domyślnie 8 kHz).

```
static void init_Timer (int delay)
{
	TIM_TIMERCFG_Type Config;
	TIM_MATCHCFG_Type Match_Cfg;

	Config.PrescaleOption = TIM_PRESCALE_USVAL;
	Config.PrescaleValue = 1;
	CLKPWR_SetPCLKDiv (CLKPWR_PCLKSEL_TIMER1, CLKPWR_PCLKSEL_CCLK_DIV_1);
	TIM_Init (LPC_TIM1, TIM_TIMER_MODE, &Config);

	Match_Cfg.ExtMatchOutputType = TIM_EXTMATCH_NOTHING;
	Match_Cfg.IntOnMatch = TRUE;
	Match_Cfg.ResetOnMatch = TRUE;
	Match_Cfg.StopOnMatch = FALSE;
```

```
	Match_Cfg.MatchChannel = 0;

	Match_Cfg.MatchValue = delay;(wartość 128)

	TIM_ConfigMatch (LPC_TIM1, &Match_Cfg);

	TIM_Cmd (LPC_TIM1, ENABLE);



	NVIC_EnableIRQ (TIMER1_IRQn);

}
```

*Timer 1* zostaje zasilony poprzez ustawienie *bitu 2 (PCTIM1) rejestru PCONP*.

```
void TIM_Init(LPC_TIM_TypeDef *TIMx, TIM_MODE_OPT TimerCounterMode, void
*TIM_ConfigStruct){
…
else if (TIMx== LPC_TIM1)
	{
		CLKPWR_ConfigPPWR (CLKPWR_PCONP_PCTIM1, ENABLE);
		//PCLK_Timer1 = CCLK/4
		CLKPWR_SetPCLKDiv (CLKPWR_PCLKSEL_TIMER1,
CLKPWR_PCLKSEL_CCLK_DIV_4);
…
}
	}
```

Wartości *licznika timera* i *licznika preskalera* są synchronicznie resetowane przez kolejno: ustawienie i wyzerowanie *bitu 1 (Counter Reset) rejestru TCR*.

```
TIMx->TCR |= (1<<1); //Reset Counter
```

Wartość preskalera jest ustawiona na 0 poprzez wpisanie wartości 0 do *rejestru PC*, oznacza to, że z każdym tickiem zegara taktującego timer wartość *licznika TC* ulegnie zwiększeniu o 1.

```
TIMx->TC =0;

TIMx->PC =0;
```

Do *rejestru MR0* zostaje wpisana *wartość 128*, co oznacza, że zdarzenia *Compare Match 1* będzie zachodzić z częstotliwością ok. *8 kHz*. Timer zostaje skonfigurowany aby w przypadku wystąpienia zdarzenia Comare Match 1 zresetować swój licznik oraz wygenerować przerwanie, w tym celu zostają ustawione *bity 3 i 4 rejestru MCR*.

```
void TIM_ConfigMatch(LPC_TIM_TypeDef *TIMx, TIM_MATCHCFG_Type
*TIM_MatchConfigStruct)
{

	CHECK_PARAM(PARAM_TIMx(TIMx));
	CHECK_PARAM(PARAM_TIM_EXTMATCH_OPT(TIM_MatchConfigStruct-
>ExtMatchOutputType));

	switch(TIM_MatchConfigStruct->MatchChannel)
	{
```

```
	case 0:
		TIMx->MR0 = TIM_MatchConfigStruct->MatchValue;
		break;
	case 1:
		TIMx->MR1 = TIM_MatchConfigStruct->MatchValue;
		break;
	case 2:
		TIMx->MR2 = TIM_MatchConfigStruct->MatchValue;
		break;
	case 3:
		TIMx->MR3 = TIM_MatchConfigStruct->MatchValue;
		break;
	default:
		//Error match value
		//Error loop
		while(1);
	}
	//interrupt on MRn
	TIMx->MCR &=~TIM_MCR_CHANNEL_MASKBIT(TIM_MatchConfigStruct->MatchChannel);

	if (TIM_MatchConfigStruct->IntOnMatch)
		TIMx->MCR |= TIM_INT_ON_MATCH(TIM_MatchConfigStruct->MatchChannel);

	//reset on MRn
	if (TIM_MatchConfigStruct->ResetOnMatch)
		TIMx->MCR |= TIM_RESET_ON_MATCH(TIM_MatchConfigStruct-
>MatchChannel);

	//stop on MRn
	if (TIM_MatchConfigStruct->StopOnMatch)
		TIMx->MCR |= TIM_STOP_ON_MATCH(TIM_MatchConfigStruct->MatchChannel);

	// match output type

	TIMx->EMR    &= ~TIM_EM_MASK(TIM_MatchConfigStruct->MatchChannel);
	TIMx->EMR   |= TIM_EM_SET(TIM_MatchConfigStruct-
>MatchChannel,TIM_MatchConfigStruct->ExtMatchOutputType);
```

Poprzez ustawienie *bitu 0 rejestru TCR* timer zostaje uruchomiony.

```
void TIM_Cmd(LPC_TIM_TypeDef *TIMx, FunctionalState NewState)
{
	CHECK_PARAM(PARAM_TIMx(TIMx));
	if (NewState == ENABLE)
	{
		TIMx->TCR    |=  TIM_ENABLE;
	}
	else
	{
		TIMx->TCR &= ~TIM_ENABLE;
	}
}
```

## 2. Przycisk odtwarzania

### GPIO

*GPIO (General Purpose Input/Output)* to ogólny termin używany w kontekście elektroniki i programowania, oznaczający uniwersalne wejścia/wyjścia. GPIO to interfejs, który umożliwia komunikację między mikrokontrolerem lub innym układem cyfrowym, a światem zewnętrznym.

Rejestry GPIO:

| **Rejestr** | **Opis** |
|---|---|
| FIODIR | (GPIO Port Direction Control Register): Ten rejestr odpowiada za kontrolę kierunku pinów GPIO. Każdy bit w tym rejestrze odpowiada jednemu pinowi GPIO. Ustawienie bitu na wartość 1 oznacza, że pin jest skonfigurowany jako wyjście, a ustawienie bitu na wartość 0 oznacza, że pin jest skonfigurowany jako wejście. |
| FIOMASK | (GPIO Mask Register): Ten rejestr umożliwia zapisywanie, ustawianie, czyszczenie i odczytywanie do portu za sprawą zapisu do<br><br>FIOPIN, FIOSET i FIOCLR oraz odczytu FIOPIN. Pozwala to na manipulację jednym lub kilkoma pinami GPIO jednocześnie, zamiast operować na każdym pinie osobno. Ustawienie bitu w FIOMASK na wartość 1 powoduje, że odpowiadający mu bit w FIOPIN, FIOSET i FIOCLR jest ignorowany. |
| FIOPIN | (GPIO Pin Value Register): Rejestr ten przechowuje aktualne wartości logiczne pinów GPIO. Każdy bit w tym rejestrze odpowiada jednemu pinowi GPIO. |
| FIOSET | (GPIO Output Set Register): Ten rejestr służy do ustawiania wartości logicznych na pinach GPIO, które są skonfigurowane jako wyjścia. Ustawienie bitu w FIOSET na wartość 1 spowoduje ustawienie odpowiadającego mu bitu w FIOPIN na 1, co oznacza stan wysoki. Można zmieniać tylko bity aktywowane przez 0 w FIOMASK. |
| FIOCLR | (GPIO Output Clear Register): Ten rejestr służy do czyszczenia wartości logicznych na pinach GPIO, które są skonfigurowane jako wyjścia. Ustawienie bitu w FIOCLR na wartość 1 spowoduje ustawienie odpowiadającego mu bitu w FIOPIN na 0, co oznacza stan niski. |

Przycisk **SW3** jest podłączony do pinu **P0.4.**

```
btn = ((GPIO_ReadValue(0) >> 4) & 0x01);

uint32_t GPIO_ReadValue(uint8_t portNum)
{
	LPC_GPIO_TypeDef *pGPIO = GPIO_GetPointer(portNum);

	if (pGPIO != NULL) {
		return pGPIO->FIOPIN;
	}

	return (0);
}
```

Wywołujemy funkcje *GPIO_ReadValue()* dla *portu 0.* Poprzez odczyt *rejestru FIOPIN* zwracamy jego stan, a następnie poprzez przesunięcie bitowe oraz maskę sprawdzamy czy *pin 4* jest w stanie wysokim, czy niskim.

## 3. Przeskakiwanie utworów

### I2C

I2C (Inter-Integrated Circuit) – to szeregowa, dwukierunkowa magistrala, która wykorzystuje dwie linie do komunikacji: *linię danych (**SDA – Serial Data Line**)* oraz *linię zegara (**SCL – Serial Clock Line**).* Linia *SDA* jest dwukierunkowa i służy do przesyłania informacji, takich jak dane, adresy lub sygnały sterujące, natomiast linia *SCL* używana jest do synchronizacji transmisji danych.

Aby zainicjować komunikacje niezbędne jest skonfigurowanie odpowiednich pinów:

- **SDA – P0.10**
- **SDL – P0.11**

Po konfiguracji komunikacja odbywa się następujący sposób:

1. Inicjalizacja: Wysyłanie sygnału START na linii danych (SDA) i ustawienie sygnału zegara (SCL) na stan wysoki.

2. Wysłanie adresu urządzenia docelowego: Wysłanie 7- lub 10-bitowego adresu urządzenia docelowego.

3. Sygnał ACK/NACK: Odbiór sygnału potwierdzającego ACK lub NACK od urządzenia docelowego.

4. Wysyłanie danych: Wysyłanie danych na linii danych (SDA) do urządzenia docelowego.

5. Sygnał ACK/NACK dla danych: Odbiór sygnału ACK lub NACK od urządzenia docelowego po wysłaniu każdego bajtu danych.

6. Odczyt danych: Przełączenie linii danych (SDA) na tryb odbioru danych i odbiór danych od urządzenia docelowego.

7. Sygnał STOP: Wysłanie sygnału STOP na linii danych (SDA) do zakończenia komunikacji.

Inicjalizacja I2C w programie realizowana jest poprzez funkcję *static void init_i2c(void)*:

```
static void init_i2c(void)
{
	PINSEL_CFG_Type PinCfg;

	/* Initialize I2C2 pin connect */
	PinCfg.Funcnum = 2;
	PinCfg.Pinnum = 10;
	PinCfg.Portnum = 0;
	PINSEL_ConfigPin(&PinCfg);
	PinCfg.Pinnum = 11;
	PINSEL_ConfigPin(&PinCfg);

	// Initialize I2C peripheral
	I2C_Init(LPC_I2C2, 100000);

	/* Enable I2C1 operation */
	I2C_Cmd(LPC_I2C2, ENABLE);
}
```

Funkcja *PINSEL_ConfigPin()* jest używana do konfiguracji pinów mikrokontrolera, które będą używane do komunikacji. *Piny 10 i 11* na *porcie 0* są konfigurowane jako *funkcja 2,* co oznacza, że są przypisane do *interfejsu I2C2*.

Funkcja *I2C_Init()* jest wywoływana, aby zainicjować *interfejs I2C.* Przekazuje się do niej wskaźnik na strukturę reprezentującą dany interfejs I2*C* (w tym przypadku *LPC_I2C2*) oraz częstotliwość zegara - *100 kHz*.

Funkcja *I2C_Cmd()* jest używana do włączenia działania interfejsu *I2C*. Przekazuje się do niej wskaźnik na strukturę reprezentującą dany interfejs *I2C* (w tym przypadku *LPC_I2C2*) oraz wartość *ENABLE*, aby włączyć interfejs.

## Rejestry I2C:

| **Rejestr** | **Opis** |
|---|---|
| I2CONSET | Rejestr kontrolny - służy do ustawiania flag i bitów kontrolnych, które sterują działaniem interfejsu $I^2C$. |
| I2STAT | Rejestr stanu - zawiera informacje o bieżącym stanie interfejsu. Służy do monitorowania postępu operacji. |

| | |
|---|---|
| I2DAT | Rejestr danych - służy do przechowywania danych, które są wysyłane lub odbierane za pośrednictwem interfejsu $I^2C$. Może zawierać zarówno dane, jak i adresy urządzeń. |
| MMCTRL | Rejestr kontroli multimastera - służy do konfiguracji trybu multimastera w przypadku, gdy na magistrali $I^2C$ działa więcej niż jedno urządzenie o zdolnościach mastera. |
| I2SCLH | Rejestr ustawień czasowych - służy do konfigurowania najwyższej wartości cyklu zegara. Określa prędkość transmisji danych na magistrali. |
| I2SCLL | Rejestr ustawień czasowych - służą do konfigurowania najniższej wartości cyklu zegara. Określa prędkość transmisji danych na magistrali. |
| I2CONCLR | Rejestr kontrolny - służy do czyszczenia ustawień i flag kontrolnych interfejsu $I^2C$. |
| I2ADR0<br>I2ADR1<br>I2ADR2<br>I2ADR3 | Rejestry adresów - są wykorzystywane do ustawiania adresu docelowego dla transmisji lub odbioru danych. Można skonfigurować do czterech różnych adresów, co umożliwia komunikację z wieloma urządzeniami na tej samej magistrali $I^2C$. |
| I2DATA_ BUFFER | Rejestr bufora danych - jest wykorzystywany do przechowywania bufora danych wejściowych lub wyjściowych podczas transmisji. |
| I2MASK0<br>I2MASK1<br>I2MASK2<br>I2MASK3 | Rejestry maski - służą do ustawiania masek bitowych, które definiują, które bity adresu są ignorowane podczas transmisji. Pozwalają na filtrowanie adresów, które są akceptowane lub odrzucane przez urządzenie $I^2C$. |

Rejestr *I2CONSET* jest 8 bitowy, poszczególne bity zawierają następujące informacje:

| **Bit** | **Opis** |
|---|---|
| 0 | zarezerwowany |
| 1 | zarezerwowany |
| 2 | AA (Assert Acknowledge) - powoduje wysłanie potwierdzenia podczas odbierania danych. |
| 3 | SI ($I^2C$ Interrupt) – generuje przerwanie $I^2C$ |

| | |
|---|---|
| 4 | STO - wysłanie sygnału stop, kończącego transmisję |
| 5 | STA - wysłanie sygnału start, rozpoczynającego transmisję |
| 6 | I2EN ($I^2C$ Interface Enable) - włącza interfejs $I^2C$ |
| 7 | zarezerwowany |

## Akcelerometr

Akcelerometr to urządzenie pomiarowe stosowane do mierzenia przyspieszenia, czyli zmiany prędkości obiektu w czasie. Podłączony jest do magistrali $I^2C$ za pomocą pinów *P0.10* oraz *P0.11,* co umożliwia wymianę danych. Po skonfigurowaniu tych pinów, które zostały szczegółowiej opisane w podrozdziale $I^2C$ następuje inicjalizacja akcelerometru:

```
void acc_init (void)
{

    /* set to measurement mode by default */

    setModeControl( (ACC_MCTL_MODE(ACC_MODE_MEASURE)
            | ACC_MCTL_GLVL(ACC_RANGE_2G) ));

}
```

Ustawiany jest tryb pomiaru (*ACC_MODE_MEASURE)* oraz zakres pomiarowy akcelerometru na wartość 2G ((*ACC_RANGE_2G*)) poprzez funkcje *setModeControl()* . Wartości te można w razie potrzeby zmienić.

**Tryby pomiaru:**

```
typedef enum
{
    ACC_MODE_STANDBY,
    ACC_MODE_MEASURE,
    ACC_MODE_LEVEL, /* level detection */
    ACC_MODE_PULSE, /* pulse detection */
} acc_mode_t;
```

**Zakresy pomiaru:**

```
typedef enum
{
    ACC_RANGE_8G,
    ACC_RANGE_2G,
    ACC_RANGE_4G,
} acc_range_t;
```

Akcelerometr daje możliwość wyboru spośród trzech zakresów precyzji dla pomiarów. Wartość *rejestru MODE* determinuje zmianę wewnętrznego przyrostu akcelerometru, umożliwiając mu pracę z odpowiednią wrażliwością.
Wrażliwość może być dostosowana w trakcie działania poprzez modyfikację *dwóch bitów GLVL* znajdujących się w *rejestrze MODE*. Poniżej przedstawiono możliwe wartości tych rejestrów.

| **GLVL [1:0]** | **g-Range** | **Wrażliwość** |
|---|---|---|
| 00 | 8g | 16 LSB/g |
| 01 | 2g | 64 LSB/g |
| 10 | 4g | 32 LSB/g |

| **MODE [1:0]** | **Funkcja** |
|---|---|
| 00 | Standby Mode |
| 01 | Measurement Mode |
| 10 | Level Detection Mode |
| 11 | Pulse Detection Mode |

Do odczytu wartości przyspieszeń akcelerometru służy funkcja *void acc_read (int8_t *x, int8_t *y, int8_t *z)*:

```
void acc_read (int8_t *x, int8_t *y, int8_t *z)
{
    uint8_t buf[1];

    /* wait for ready flag */
    while ((getStatus() & ACC_STATUS_DRDY) == 0);

    /*
     * Have experienced problems reading all registers
     * at once. Change to reading them one-by-one.
     */
    buf[0] = ACC_ADDR_XOUT8;
    I2CWrite(ACC_I2C_ADDR, buf, 1);
    I2CRead(ACC_I2C_ADDR, buf, 1);

    *x = (int8_t)buf[0];

    buf[0] = ACC_ADDR_YOUT8;
    I2CWrite(ACC_I2C_ADDR, buf, 1);
    I2CRead(ACC_I2C_ADDR, buf, 1);

    *y = (int8_t)buf[0];

    buf[0] = ACC_ADDR_ZOUT8;
```

```
    I2CWrite(ACC_I2C_ADDR, buf, 1);
    I2CRead(ACC_I2C_ADDR, buf, 1);

    *z = (int8_t)buf[0];
}
```

Następnie poprzez funkcje *I2CWrite()* oraz *I2CRead()*, które wykonują operacje zapisu oraz odczytu poprzez magistrale $I^2C$, wysyłany jest adres akcelerometru - *0x1D* oraz odczytane wartości przyspieszenia.

| **Rejestr** | **Adres rejestru** |
|---|---|
| ACC_ADDR_XOUT8 | 0x06 |
| ACC_ADDR_YOUT8 | 0x07 |
| ACC_ADDR_ZOUT8 | 0x08 |

Adresy te są używane w funkcji do określenia, które dane mają być odczytywane z akcelerometru.

## 4. Zmiana głośności

### ADC

ADC (*Analog-to-Digital Converter)* – to przetwornik służący do przekształcania sygnałów analogowych na sygnały cyfrowe.

Aby go zainicjować, niezbędne jest skonfigurowanie odpowiedniego pinu:

- *P0.23*

Inicjacja ADC w programie realizowana jest poprzez funkcję *static void init_adc(void)*:

```
static void init_adc(void)
{
	PINSEL_CFG_Type PinCfg;

	/*
	 * Init ADC pin connect
	 * AD0.0 on P0.23
	 */
	PinCfg.Funcnum = 1;
	PinCfg.OpenDrain = 0;
	PinCfg.Pinmode = 0;
	PinCfg.Pinnum = 23;
	PinCfg.Portnum = 0;
	PINSEL_ConfigPin(&PinCfg);

	/* Configuration for ADC :
	 *	Frequency at 0.2Mhz
	 *  ADC channel 0, no Interrupt
	 */
	ADC_Init(LPC_ADC, 200000);
	ADC_IntConfig(LPC_ADC,ADC_CHANNEL_0,DISABLE);
	ADC_ChannelCmd(LPC_ADC,ADC_CHANNEL_0,ENABLE);

}
```

Funkcja PINSEL_ConfigPin*()* jest wywoływana, aby skonfigurować pin przekazany przez parametr.

Funkcja *ADC_Init()* jest wywoływana, aby zainicjować przetwornik ADC. Przekazuje się do niej wskaźnik na strukturę reprezentującą dany przetwornik, oraz częstotliwość zegara – *0.2 Mhz*. Ustawia ona:

- bit PCADC
- zegar dla ADC
- częstotliwość zegara

Funkcja *ADC_IntConfig()* jest wywoływana, aby skonfigurować przerwania ADC. Przekazuje się do niej wskaźnik na strukturę reprezentującą dany przetwornik, kanał przerwania (w naszym przypadku kanał 0) oraz flagę generującą przerwanie (w naszym przypadku jest ona ustawiona na „DISABLE").

Funkcja *ADC_ChannelCmd()* jest wywoływana, aby włączyć kanał ADC. Przekazuje się do niej wskaźnik na strukturę reprezentującą dany przetwornik, numer kanału, oraz flagę (w naszym przypadku „ENABLE").

Po konfiguracji musimy także ustawić tryb startowy korzystając z funkcji *ADC_StartCmd().*

```
ADC_StartCmd(LPC_ADC,ADC_START_NOW);
```

Przekazuje się do niej wskaźnik na strukturę reprezentującą dany przetwornik oraz tryb startowy (w naszym przypadku „ADC_START_NOW"). Do wyboru mamy następujące tryby:

- *ADC_START_OPT*
- *ADC_START_CONTINUOUS*
- *ADC_START_NOW*
- *ADC_START_ON_EINT0*
- *ADC_START_ON_CAP01*
- *ADC_START_ON_MAT01*
- *ADC_START_ON_MAT03*
- *ADC_START_ON_MAT10*
- *ADC_START_ON_MAT11*

Do pozyskania danych używamy funkcji *ADC_ChannelGetData().*

```
trim = ADC_ChannelGetData(LPC_ADC,ADC_CHANNEL_0);
```

Przekazuje się do niej wskaźnik na strukturę reprezentującą dany przetwornik oraz numer kanału.

**Rejestry ADC:**

| **Rejestr** | **Opis** |
|---|---|
| ADCR | Rejestr kontrolny – pozwala wybrać tryb pracy przed rozpoczęciem konwersji A/D. |
| ADGDR | Rejestr danych – zawiera bit DONE przetwornika ADC oraz wynik ostatniej konwersji A/D. |
| ADINTEN | Rejestr przerwań – pozwala włączyć lub wyłączyć przerwanie. |
| ADDR0 | Rejestr danych kanału 0 - zawiera wynik ostatniej konwersji zakończonej na kanale 0. |
| ADDR1 | Rejestr danych kanału 1 - zawiera wynik ostatniej konwersji zakończonej na kanale 1. |
| ADDR2 | Rejestr danych kanału 2 - zawiera wynik ostatniej konwersji zakończonej na kanale 2. |
| ADDR3 | Rejestr danych kanału 3 - zawiera wynik ostatniej konwersji zakończonej na kanale 3. |
| ADDR4 | Rejestr danych kanału 4 - zawiera wynik ostatniej konwersji zakończonej na kanale 4. |
| ADDR5 | Rejestr danych kanału 5 - zawiera wynik ostatniej konwersji zakończonej na kanale 5. |
| ADDR6 | Rejestr danych kanału 6 - zawiera wynik ostatniej konwersji zakończonej na kanale 6. |
| ADDR7 | Rejestr danych kanału 7 - zawiera wynik ostatniej konwersji zakończonej na kanale 7. |
| ADDSTAT | Rejestr stanu – zawiera flagi DONE oraz OVERRUN dla wszystkich kanałów A/D, jak również flagę A/D interrupt/DMA. |
| ADTRM | Jest rejestrem przycinania ADC. |

## Wzmacniacz analogowy LM4811

Wzmacniacz analogowy LM4811 to podwójny wzmacniacz mocy audio. LM4811 posiada cyfrową regulację głośności, która ustawia wzmocnienie wzmacniacza w zakresie od +12dB do -33dB w 16 dyskretnych krokach za pomocą interfejsu dwuprzewodowego.

W celu przystąpienia do zmieniania głośności konfigurowane są odpowiednie piny:

```
GPIO_SetDir(0, 1<<27, 1);
GPIO_SetDir(0, 1<<28, 1);
GPIO_SetDir(2, 1<<13, 1);
```

```
GPIO_ClearValue(0, 1<<27); //LM4811-clock
GPIO_ClearValue(0, 1<<28); //LM4811-up/dn
GPIO_ClearValue(2, 1<<13); //LM4811-shutdn
```

Funkcja *GPIO_SetDir()* jest wywoływana, aby ustawić kierunek dla portu GPIO. Przekazuje się do niej wartość numeru portu (powinna być w zakresie od 0 do 4) oraz wartość zawierającą wszystkie bity do ustawienia kierunku, w zakresie od 0 do 0xFFFFFFFF.

Funkcja *GPIO_ClearValue()* jest wywoływana, aby początkowo wyczyścić wartości bitów, które mają kierunek wyjścia na porcie GPIO. Przekazuje się do niej wartość numeru portu (powinna być w zakresie od 0 do 4) oraz wartość zawierającą zawierająca wszystkie bity na GPIO do wyczyszczenia, w zakresie od 0 do 0xFFFFFFFF.

Wzmocnienie LM4811 jest kontrolowane przez sygnały przyłożone do wejść *CLOCK* i wejścia *UP/DN*. Zewnętrzny zegar jest wymagany do zasilania pinu *CLOCK*. Przy każdym narastającym impulsie sygnału zegarowego, wzmocnienie będzie się zwiększać lub zmniejszać o krok wielkości 3 dB w zależności od poziomu napięcia przyłożonego do pinów *UP/DN*. Po włączeniu zasilania urządzenia, wzmocnienie jest ustawione na domyślną wartość 0 dB.

## 5. Wyświetlanie postępu odtwarzanego utworu

### PCA9532

PCA9532 to 16-bitowy ekspander wejść/wyjść magistrali $I^2C$ i SMBus (magistrala zarządzania systemem) zoptymalizowany do ściemniania diod LED w 256 dyskretnych krokach dla mieszania kolorów RGB i podświetlania.

Do ustawienia stanów diod LED (włączone lub wyłączone) wywoływana jest funkcja *pca9532_setLeds(),* która przyjmuje następujące parametry:

- *Diody LED, które powinny być włączone. Ta maska ma priorytet nad drugim parametrem.*
- *Diody LED, które powinny być wyłączone.*

PCA9532 zawiera wewnętrzny oscylator z dwoma programowalnymi przez użytkownika częstotliwościami migania i cyklami pracy sprzężonymi z *wyjściem PWM*.
Jasność diody LED jest kontrolowana przez ustawienie częstotliwości migania na tyle wysokiej (> 100 Hz), aby miganie nie było widoczne, a następnie używając cyklu pracy, aby zmienić czas, przez jaki dioda LED jest włączona, a tym samym średni prąd przepływający przez diodę LED.

Tylko jedno polecenie z magistrali jest wymagane do włączenia, wyłączenia - *BLINK RATE 1* lub *BLINK RATE 2*.
W oparciu o zaprogramowaną częstotliwość i cykl pracy - *BLINK RATE 1 i BLINK RATE 2* powodują, że diody LED świecą z różną jasnością lub migają.

**Definicja rejestru kontrolnego:**

| **B3** | **B2** | **B1** | **B0** | **Nazwa rejestru** | **Funkcja rejestru** |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | INPUT0 | Rejestr wejścia |
| 0 | 0 | 0 | 1 | INPUT1 | Rejestr wejścia |
| 0 | 0 | 1 | 0 | PSC0 | Dzielnik częstotliwości |
| 0 | 0 | 1 | 1 | PWM0 | Rejestr PWM |
| 0 | 1 | 0 | 0 | PSC1 | Dzielnik częstotliwości |
| 0 | 1 | 0 | 1 | PWM1 | Rejestr PWM |
| 0 | 1 | 1 | 0 | LS0 | Selektor LED 0-3 |
| 0 | 1 | 1 | 1 | LS1 | Selektor LED 4-7 |
| 1 | 0 | 0 | 0 | LS2 | Selektor LED 8-11 |
| 1 | 0 | 0 | 1 | LS3 | Selektor LED 12-15 |

**Rejestr INPUT0:**

| | LED 7 | LED 6 | LED 5 | LED 4 | LED 3 | LED 2 | LED 1 | LED 0 |
|---|---|---|---|---|---|---|---|---|
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| domyślnie | X | X | X | X | X | X | X | X |

**Rejestr INPUT1:**

| | LED 15 | LED 14 | LED 13 | LED 12 | LED 11 | LED 10 | LED 9 | LED 8 |
|---|---|---|---|---|---|---|---|---|
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| domyślnie | X | X | X | X | X | X | X | X |

*Rejestr INPUT0* oraz *INPUT1* odzwierciedla stan pinów urządzenia (*wejścia 0 do 7 oraz 8 do 15*). Domyślna wartość "X" jest określana przez zewnętrznie zastosowaną logikę.

**Rejestr PSC0 - służy do programowania okresu wyjścia PWM:**

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| domyślnie | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Rejestr PWM0 - określa cykl pracy BLINK0:**

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| domyślnie | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Rejestr PSC1 – służy do programowania okresu wyjścia PWM:**

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| domyślnie | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Rejestr PWM1 – określa cykl pracy BLINK1:**

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| domyślnie | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Rejestr LS0:**

| | LED 3 | | LED 2 | | LED 1 | | LED 0 | |
|---|---|---|---|---|---|---|---|---|
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| domyślnie | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Rejestr LS1:**

| | LED 7 | | LED 6 | | LED 5 | | LED 4 | |
|---|---|---|---|---|---|---|---|---|
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| domyślnie | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Rejestr LS2:**

| | LED 11 | | LED 10 | | LED 9 | | LED 8 | |
|---|---|---|---|---|---|---|---|---|
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| domyślnie | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Rejestr LS3:**

| | LED 15 | | LED 14 | | LED 13 | | LED 12 | |
|---|---|---|---|---|---|---|---|---|
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| domyślnie | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Rejestry LSX określają źródło danych LED:

- 00 = Wyjście ustawiona jest na Hi-Z (dioda LED wyłączona)
- 01 = Wyjście ustawione jest na LOW (dioda LED włączona)
- 10 = Wyjście miga z częstotliwością PWM0
- 11 = Wyjście miga z częstotliwością PWM1

## 6. Wyświetlanie numeru utworu na ekranie

### SPI

Serial Peripheral Interface – magistrala szeregowa, umożliwiająca synchroniczną i dwukierunkową transmisję danych. Transmisja jest rozpoczynana przez jedno urządzenie nadrzędne.

Piny magistrali SPI:

- SCK (Serial Clock) – sygnału taktującego *(PIO2_7)*
- MISO (Master In Slave Out) – wyjścia danych z układu podrzędnego *(PIO0_8)*
- MOSI (Master Out Slave In) – wejścia danych układu podrzędnego *(PIO0_9)*
- SSEL (Slave Select) – wyboru slave

Przez standardowy interfejs SPI transmisja realizowana jest po jednym bajcie, gdzie najbardziej znaczący bit wysyłany jest jako pierwszy.

W przypadku mikrokontrolera *LPC 1769* magistrala SPI jest podłączona między innymi do *interfejsu SD/MMC* oraz wyświetlacza *OLED*, choć wyświetlacz OLED może zostać alternatywnie podłączony również do interfejsu I2C.

Konfiguracja pinów i inicjalizacja interfejsu SPI wykonywana jest na początku działania programu:

```
static void init_ssp(void)

{
	SSP_CFG_Type SSP_ConfigStruct;
	PINSEL_CFG_Type PinCfg;

	/*
	 * Initialize SPI pin connect
	 * P0.7 - SCK;
	 * P0.8 - MISO
	 * P0.9 - MOSI
	 * P2.2 - SSEL - used as GPIO
	 */
	PinCfg.Funcnum = 2;
	PinCfg.OpenDrain = 0;
	PinCfg.Pinmode = 0;
	PinCfg.Portnum = 0;
	PinCfg.Pinnum = 7;
	PINSEL_ConfigPin(&PinCfg);
	PinCfg.Pinnum = 8;
	PINSEL_ConfigPin(&PinCfg);
	PinCfg.Pinnum = 9;
	PINSEL_ConfigPin(&PinCfg);
	PinCfg.Funcnum = 0;
	PinCfg.Portnum = 2;
	PinCfg.Pinnum = 2;
	PINSEL_ConfigPin(&PinCfg);

	SSP_ConfigStructInit(&SSP_ConfigStruct);
```

```
	// Initialize SSP peripheral with parameter given in structure above
	SSP_Init(LPC_SSP1, &SSP_ConfigStruct);

	// Enable SSP peripheral
	SSP_Cmd(LPC_SSP1, ENABLE);
}

void SSP_Init(LPC_SSP_TypeDef *SSPx, SSP_CFG_Type *SSP_ConfigStruct)
{
	uint32_t tmp;

	CHECK_PARAM(PARAM_SSPx(SSPx));

	if(SSPx == LPC_SSP0) {
		/* Set up clock and power for SSP0 module */
		CLKPWR_ConfigPPWR (CLKPWR_PCONP_PCSSP0, ENABLE);
	} else if(SSPx == LPC_SSP1) {
		/* Set up clock and power for SSP1 module */
		CLKPWR_ConfigPPWR (CLKPWR_PCONP_PCSSP1, ENABLE);
	} else {
		return;
	}

	/* Configure SSP, interrupt is disable, LoopBack mode is disable,
	 * SSP is disable, Slave output is disable as default
	 */
	tmp = ((SSP_ConfigStruct->CPHA) | (SSP_ConfigStruct->CPOL) \
		| (SSP_ConfigStruct->FrameFormat) | (SSP_ConfigStruct->Databit))
		& SSP_CR0_BITMASK;
	// write back to SSP control register
	SSPx->CR0 = tmp;

	tmp = SSP_ConfigStruct->Mode & SSP_CR1_BITMASK;
	// Write back to CR1
	SSPx->CR1 = tmp;

	// Set clock rate for SSP peripheral
	setSSPclock(SSPx, SSP_ConfigStruct->ClockRate);
}
```

SPI zawiera 5 rejestrów:

| **Rejestr** | **Opis** |
|---|---|
| S0SPCR | Rejestr kontrolny używany do konfiguracji SPI |
| S0SPSR | Rejestr statusu SPI |
| S0SPDR | Rejestr danych zawierający otrzymane dane, bądź dane do transmisji |
| S0SPCCR | Rejestr licznika zegara używany do kontrolowania częstotliwości SCK |
| S0SPINT | Rejestr flagi przerwania |

## OLED

Ekran OLED1 używany w mikrokontrolerze ma rozdzielczość 96x64 piksele, a zasilany jest napięciem o wysokości ok. 11V kontrolowanym przez *PIO1_10*.
Do sterowania ekranem wykorzystywany jest sterownik SSD1305.
Inicjalizacja urządzenia następuje na początku działania programu:

```
void oled_init (void)
{
    int i = 0;

    //GPIO_SetDir(PORT0, 0, 1);
    GPIO_SetDir(2, (1<<1), 1);
    GPIO_SetDir(2, (1<<7), 1);
    GPIO_SetDir(0, (1<<6), 1);

    /* make sure power is off */
    GPIO_ClearValue( 2, (1<<1) );

#ifdef OLED_USE_I2C
    GPIO_ClearValue( 2, (1<<7)); // D/C#
    GPIO_ClearValue( 0, (1<<6)); // CS#
#else
    OLED_CS_OFF();
#endif

    runInitSequence();

    memset(shadowFB, 0, SHADOW_FB_SIZE);

    /* small delay before turning on power */
    for (i = 0; i < 0xffff; i++);

     /* power on */
    GPIO_SetValue( 2, (1<<1) );
}
```

Zapis i odczyt danych ekranu zrealizowany jest z użyciem magistrali SPI.

```
static void
writeData(uint8_t data)
{
(…)
    SSP_DATA_SETUP_Type xferConfig;
    OLED_DATA();
    OLED_CS_ON();

      xferConfig.tx_data = &data;
      xferConfig.rx_data = NULL;
      xferConfig.length  = 1;
    SSP_ReadWrite(LPC_SSP1, &xferConfig, SSP_TRANSFER_POLLING);
    OLED_CS_OFF();
#endif
```

Ekran posiada dostępne trzy różne tryby adresacji pamięci:

- Strony (używana w programie)
- Horyzontalna
- Wertykalna

W trybie stronicowym po każdym odczycie/zapisie do pamięci RAM ekranu, wskaźnik adresu kolumny jest automatycznie zwiększany o 1. Wskaźnik adresu kolumny po dotarciu do adresu ostatniej kolumny jest resetowany, a wskaźnik adresu strony pozostaje bez zmian. Przejście na kolejną stronę musi być zainicjowane ręcznie.
Operacje na pamięci RAM ekranu wykonywane są poprzez przesłanie konkretnych instrukcji procesora i wartości z wykorzystaniem magistrali SPI:

```
void oled_putPixel(uint8_t x, uint8_t y, oled_color_t color) {

    uint8_t page;
    uint16_t add;
    uint8_t lAddr;
    uint8_t hAddr;
    uint8_t mask;
    uint32_t shadowPos = 0;

    if (x > OLED_DISPLAY_WIDTH) {
        return;
    }
    if (y > OLED_DISPLAY_HEIGHT) {
        return;
    }

    /* page address */
         if(y < 8)  page = 0xB0;
    else if(y < 16) page = 0xB1;
    else if(y < 24) page = 0xB2;
    else if(y < 32) page = 0xB3;
    else if(y < 40) page = 0xB4;
    else if(y < 48) page = 0xB5;
    else if(y < 56) page = 0xB6;
    else            page = 0xB7;

    add = x + X_OFFSET;
    lAddr = 0x0F & add;             // Low address
    hAddr = 0x10 | (add >> 4);      // High address

    // Calculate mask from rows basically do a y%8 and remainder is bit position
    add = y>>3;                     // Divide by 8
    add <<= 3;                      // Multiply by 8
    add = y - add;                  // Calculate bit position
    mask = 1 << add;                // Left shift 1 by bit position

    setAddress(page, lAddr, hAddr); // Set the address (sets the page,
                                    // lower and higher column address pointers)

    shadowPos = (page-0xB0)*OLED_DISPLAY_WIDTH+x;

    if(color > 0)
        shadowFB[shadowPos] |= mask;
```

```
    else
        shadowFB[shadowPos] &= ~mask;


    writeData(shadowFB[shadowPos]);
}
```

Poniższy fragment kodu odtwarzacza odpowiada za wyświetlanie nazw odtwarzanych plików:

```
(…)
if(chosenFileIndex != prevChosenFileIndex){
                        oled_clearScreen(OLED_COLOR_WHITE);
                        char name[strlen(fileNames[chosenFileIndex]+3)];
                        sprintf(name, "%i. %s", chosenFileIndex+1,
fileNames[chosenFileIndex]);
                        printf(name);
                        oled_putString(1,1,name, OLED_COLOR_BLACK,
OLED_COLOR_WHITE);
(…)
```

## 7. Wyświetlanie numeru utworu na ekranie

### MMC

Interfejs karty pamięci SD (Secure Digital) /MMC (Multi Media Card) umożliwia odczyt danych do pamięci RAM mikrokontrolera, oraz zapis danych na kartę. Do przesyłu danych wykorzystywany jest interfejs SPI.

Po zainicjalizowaniu wszystkich potrzebnych urządzeń peryferyjnych, następuje inicjalizacja interfejsu SD/MMC i próba odczytu danych z karty:

```
(…)
DSTATUS stat;
    BYTE res;
    DIR dir;

    printf("MMC: Initializing\r\n");

    SysTick_Config(SystemCoreClock / 100);

    Timer0_Wait(500);

    stat = disk_initialize(0);
    if (stat & STA_NOINIT)
     printf("MMC: not initialized\r\n");

    if (stat & STA_NODISK)
     printf("MMC: No Disk\r\n");

    if (stat != 0)
        return 1;

    printf("MMC: Initialized\r\n");
    printf("Reading FLASH");

    res = f_mount(0, &Fatfs[0]);
    if(res != FR_OK) {
       printf("Failed to mount");
       return 1;
    }

    res = f_opendir(&dir, "/");
    if(res) {
       printf("Failed to open dir /");
       return 1;
    }
(…)

DSTATUS disk_initialize (
       BYTE drv            /* Physical drive nmuber (0) */
)
{
       BYTE n, cmd, ty, ocr[4];

       GPIO_SetDir(2, 1<<2, 1);  /* CS */
       GPIO_SetDir(2, 1<<11, 0); /* Card Detect */
```

```
	if (drv) return STA_NOINIT;			/* Supports only single drive */
	if (Stat & STA_NODISK) return Stat;	/* No card in the socket */

	power_on();							/* Force socket power on */
	FCLK_SLOW();
	for (n = 10; n; n--) rcvr_spi(); /* 80 dummy clocks */

	ty = 0;
	if (send_cmd(CMD0, 0) == 1) {			/* Enter Idle state */
		Timer1 = 100;
		if (send_cmd(CMD8, 0x1AA) == 1) {	/* SDHC */
			for (n = 0; n < 4; n++) ocr[n] = rcvr_spi();
			if (ocr[2] == 0x01 && ocr[3] == 0xAA) {
				while (Timer1 && send_cmd(ACMD41, 1UL << 30));
				if (Timer1 && send_cmd(CMD58, 0) == 0) {
					for (n = 0; n < 4; n++) ocr[n] = rcvr_spi();
					ty = (ocr[0] & 0x40) ? CT_SD2 | CT_BLOCK : CT_SD2;
				}
			}
		} else {
			if (send_cmd(ACMD41, 0) <= 1)	{
				ty = CT_SD1; cmd = ACMD41;/* SDv1 */
			} else {
				ty = CT_MMC; cmd = CMD1;  /* MMCv3 */
			}
			while (Timer1 && send_cmd(cmd, 0));
			if (!Timer1 || send_cmd(CMD16, 512) != 0)
				ty = 0;
		}
	}
	CardType = ty;
	deselect();

	if (ty) {			/* Initialization succeded */
		Stat &= ~STA_NOINIT;		/* Clear STA_NOINIT */
		FCLK_FAST();
	} else {			/* Initialization failed */
		power_off();
	}

	return Stat;
}
```

Kolejne bajty plików wczytywane są do dwóch naprzemiennie napełnianych i opróżnianych buforów:

```
if(playingBuffer == 2){
				if(emptyBuffer1){
					bufferNum++;
					f_read(&file, buffer1, sizeof buffer1, &br);
					emptyBuffer1 = FALSE;
				}
			}
			else if(playingBuffer == 1){
				if(emptyBuffer2){
					bufferNum++;
					f_read(&file, buffer2, sizeof buffer2, &br);
					emptyBuffer2 = FALSE;
				}
```

## Analiza FMEA/FMECA

| **Ryzyko** | **Prawdopodobieństwo** | **Znaczenie** | **Samo wykrywalność** | **Reakcja** |
|---|---|---|---|---|
| Mikrokontroler | 0.02 | Krytyczne | Brak | Brak |
| Akcelerometr | 0.02 | Znaczące | Brak | Brak |
| Przycisk SW3 | 0.2 | Znaczące | Brak | Brak |
| Pokrętło R105 | 0.2 | Znaczące | Brak | Brak |
| Wyświetlacz OLED | 0.1 | Niegroźne | Brak | Brak |
| Brak karty SD | 0.3 | Krytyczne | Przy uruchomieniu programu sprawdzana jest obecność karty | Wyświetlanie informacji w programie |
| Ledy U12 | 0.1 | Niegroźne | Brak | Brak |
| $I^2C$ | 0.02 | Znaczące | Brak | Brak |
| SPI | 0.02 | Znaczące | Brak | Brak |
| DAC | 0.02 | Znaczące | Brak | Brak |