Grzegorz Ryński

Systemy wbudowane
Projekt

Do stworzenia projektu była wykorzystana płytka STM32F429ZI

Założenia projektu

Projekt zakłada stworzenie aplikacji dźwiękowej z wykorzystaniem zewnętrznego głośnika. Użytkownik za pomocą menu wyświetlanym na ekranie oraz zewnętrznym urządzeniom wejścia może zaprogramować melodię, która następnie będzie odtwarzana poprzez głośnik.

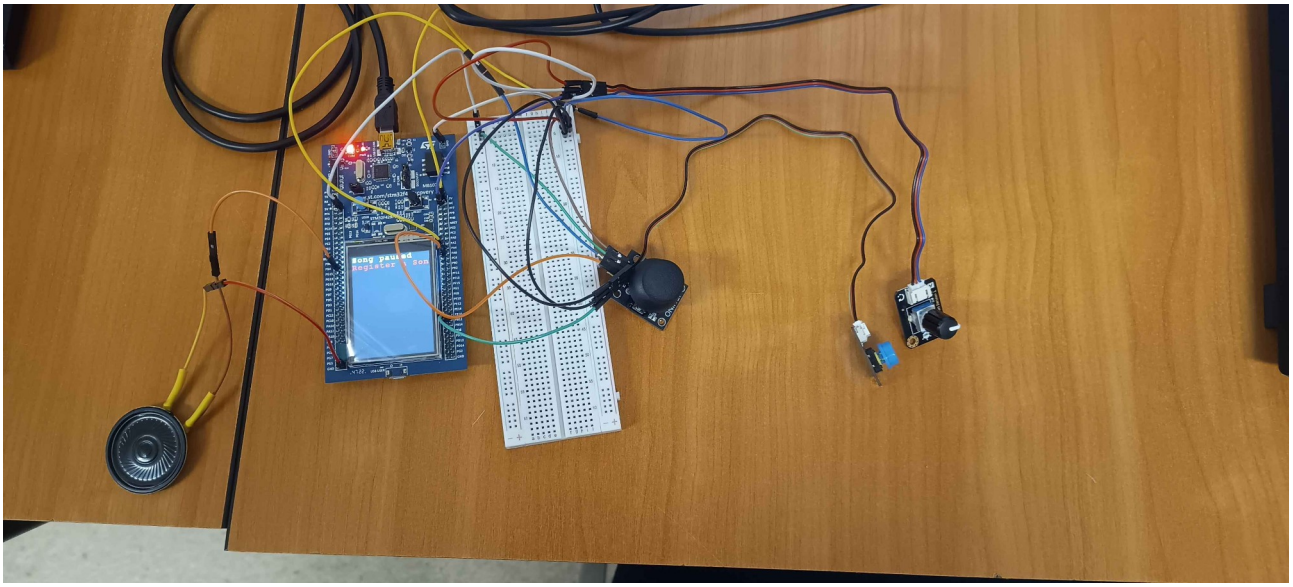
Elementy użyte w projekcie

- Wyświetlacz wbudowany w płytkę,
- Dioda wbudowana w płytkę,
- Przycisk wbudowany w płytkę,
- Przycisk zewnętrzny,
- Joystick zewnętrzny,
- Potencjometr zewnętrzny,
- Głośnik zewnętrzny

Elementy programistyczne użyte w projekcie

- Wewnętrzne zegary,
- Konwerter analogowo-cyfrowy,
- Generowanie sygnału PWN,
- Przerwanie

Zdjęcie podłączenia układu



Piny na płytce

- Sygnał wyjściowy, generowany PWN – podłączenie głośnika: PB4

- Sygnał wejściowy joysticka osi X: PA5

- Sygnał wejściowy joysticka osi Y: PA7

- Sygnał wejściowy konwertera analogowo-cyfrowego – podłączenie potencjometru – PF6

- Sygnał wejściowy przycisku zewnętrznego: PB13

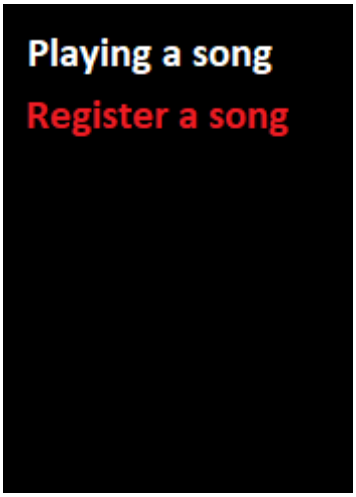
Działanie programu

Po poprawnym podłączeniu układu i uruchomieniu programu na wyświetlaczu pojawi się menu początkowe:

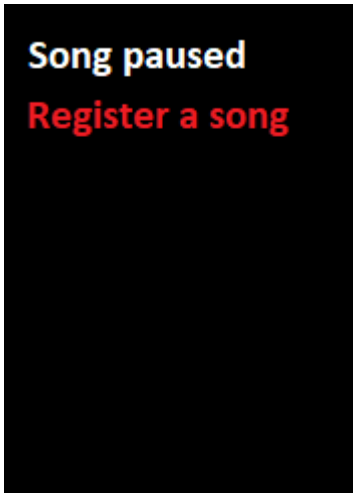
Pierwsza linijka jest informacją na temat stanu programu. Wszystkie pozostałe są opcjami do wyboru, a aktualnie wybrana opcja jest wyświetlana na czerwono. Po menu użytkownik porusza się za pomocą gałki analogowej joysticka w kierunkach poziomych i pionowych. Aby zatwierdzić wybraną opcję użytkownik naciska przycisk zewnętrzny.

Po wybraniu opcji „Record a song” prezentowane jest menu zaprogramowania melodii:

Tutaj użytkownik wybiera wysokość tonu oraz długość granej nuty, aby zatwierdzić wybraną nutę i dodać ją do granej melodii użytkownik wybiera opcję „Confirm”, a aby zakończyć rejestrowanie melodii opcję „Finish”. Wtedy prezentowane zostaje menu grania melodii poprzez głośnik:



To menu informuje użytkownika że melodia jest odtwarzana. Aby zarejestrować nową melodię trzeba wpierw nacisnąć niebieski przycisk USER na płytce, aby zatrzymać granie melodii:



W tym momencie użytkownik może wybrać opcję zarejestrowania nowej melodii i powtórzenia poprzednich kroków lub ponownego wciśnięcia przycisku na płytce, aby wznowić odtwarzanie melodii.

Dodatkową możliwością modyfikacji odtwarzanej melodii jest zmienianie prędkości jej odtwarzania za pomocą potencjometru:

- Maksymalne odchylenie – Melodia gra dwukrotnie szybciej

- Minimalne odchylenie – Melodia gra dwukrotnie wolniej

- Odchylenie częściowe – Melodia gra w tempie zwyczajnym

Elementy programu

Program wykorzystuje własne struktury: Song i Note

```
struct Note {
	int freq;
	float dur;
};
```

Struktura Note przechowuje w sobie informacje o wartości potrzebnej do ustawiania odpowiedniej częstotliwości generowania sygnału PWN do wytworzenia oczekiwanego tonu oraz o długości nuty, która ma być odtwarzana.

```
struct Song {
	struct Note notes[10000];
	int length;
};
```

Struktura Song jest strukturą melodii, która przechowuje w sobie obiekty typu strukturalnego Note oraz ilości zarejestrowanych nut, w celu uproszczenia i zoptymalizowania programu zakłada się iż melodia nie będzie składała się z większej ilości nut niż 10000.

W programie zostało zastosowanych wiele funkcji odczytujących dane z urządzeń wejścia

```
int readButton(int *val) {
	int new_val;
	if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_13) == GPIO_PIN_SET) {
		new_val = 1;
	}
	else if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_13) == GPIO_PIN_RESET) {
		new_val = 0;
	}
	if (*val != new_val) {
		*val = new_val;
		return 1;
	}
	return 0;
}
```

Funkcja readButton odczytuje wartość z zewnętrznego przycisku i przypisuje ją do zmiennej o ile jest to nowa wartość, gdy wartość odczytana z przycisku jest inna to funkcja zwraca 1, a gdy taka sama to zwraca 0.

```
int readAnalogStickx(uint32_t value, int* val) {
	int new_val;
	if (value > 3200) {
		new_val = 1;
	}
	else if (value < 3000) {
		new_val = -1;
	}
	else new_val = 0;
	if (*val!=new_val){
		*val=new_val;
		return 1;
	}
	return 0;
}
```

```
int readAnalogSticky(uint32_t value, int* val) {
	int new_val;
	if (value > 3400) {
		new_val = -1;
	}
	else if (value < 2800) {
		new_val = 1;
	}
	else new_val = 0;
	if (*val!=new_val){
		*val=new_val;
		return 1;
	}
	return 0;
}
```

Funkcje readAnalogStickx i readAnalogSticky są bardzo podobne, z tą różnicą że polaryzacja osi Y jest odwrócona, a czułość lekko zwiększona. Funkcja odczytuje wartość analogową otrzymaną od konwertera analogowo-cyfrowego i mapuje ją na wartości z zakresu {-1;0;1}, oraz rejestruje czy sygnał został zmieniony od ostatniego odczytu.

```
int readPotentiometer(int *val) {
	uint32_t value = HAL_ADC_GetValue(&hadc3);
	int new_val;
	if (value > 3200) {
		new_val = 2;
	}
	else if (value < 3200 && value > 700) {
		new_val = 1;
	}
	else if (value < 700) {
		new_val = 0;
	}
	if (*val != new_val) {
		*val = new_val;
		return 1;
	}
	return 0;
}
```

Funkcja readPotentiometer działa analogicznie jak funkcje odczytujące wartości z drążka analogowego.

```
int Action(int *option, int *valuex, int *valuey, int *valueButton, int maxoption, int
*freq_index, int *len_index) {
	if (*valuey == -1 && *option < 3 && *valuex==0) {
		*option += 1;
	}
	if (*valuey == 1 && *option > 0 && *valuex == 0) {
		*option -= 1;
	}
	if (*valuey == 0 && *option == 0 && *valuex == 1 && *freq_index<24) {
		*freq_index += 1;
	}
	if (*valuey == 0 && *option == 0 && *valuex == -1 && *freq_index>0) {
		*freq_index -= 1;
	}
	if (*valuey == 0 && *option == 1 && *valuex == 1 && *len_index < 4) {
		*len_index += 1;
	}
	if (*valuey == 0 && *option == 1 && *valuex == -1 && *len_index > 0) {
		*len_index -= 1;
	}
	if (*valuey == 0 && *option == 3 && *valuex == 0 && *valueButton == 1) {
		return 0;
	}
	if (*valuey == 0 && *option == 0 && *valuex == 0 && *valueButton == 1) {
		return 2;
	}
	if (*option > maxoption) {
		*option = maxoption;
	}
	return 1;
}
```

Funkcja Action wykonuje wszystkie działania związane z wykorzystaniem danych pobranych z urządzeń na zmiennych pomocniczych potrzebnych do wyświetlania i wybierania odpowiednich opcji z menu. Zwraca określone wartości całkowite zależnie od konfiguracji sygnałów wejściowych i odpowiednio wybranych opcji z menu.

```
void Display(int mode, int* option, int* freq, float* dur) {
	if (mode == 0) {
		BSP_LCD_DisplayStringAtLine(0, (uint8_t*)"                      ");
		BSP_LCD_DisplayStringAtLine(0, (uint8_t*)"Playing a Song");
		BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"                      ");
		BSP_LCD_SetTextColor(LCD_COLOR_RED);
		BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Record a Song");
		BSP_LCD_SetTextColor(LCD_COLOR_WHITE);
		BSP_LCD_DisplayStringAtLine(2, (uint8_t*)"                      ");
		BSP_LCD_DisplayStringAtLine(3, (uint8_t*)"                      ");
		BSP_LCD_DisplayStringAtLine(4, (uint8_t*)"                      ");
	}
	if (mode == 1) {
		BSP_LCD_SetTextColor(LCD_COLOR_WHITE);
		BSP_LCD_DisplayStringAtLine(0, (uint8_t*)"                  ");
		BSP_LCD_DisplayStringAtLine(0, (uint8_t*)"Recording song");
		BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"                  ");
		if (*option == 0) {
			BSP_LCD_SetTextColor(LCD_COLOR_RED);
		}
		if (*freq == 308) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: C1");
		}
		if (*freq == 290) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: CIS1");
		}
		if (*freq == 274) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: D1");
		}
		if (*freq == 256) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: DIS1");
		}
		if (*freq == 244) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: E1");
		}
		if (*freq == 230) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: F1");
		}
		if (*freq == 217) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: FIS1");
		}
		if (*freq == 204) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: G1");
		}
		if (*freq == 192) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: GIS1");
		}
		if (*freq == 182) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: A1");
		}
		if (*freq == 171) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: AIS1");
		}
		if (*freq == 163) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: B1");
		}
		if (*freq == 153) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: C2");
		}
		if (*freq == 144) {
			BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: CIS2");
		}
		if (*freq == 136) {
```

```
        BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: D2");
    }
    if (*freq == 128) {
        BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: DIS2");
    }
    if (*freq == 121) {
        BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: E2");
    }
    if (*freq == 114) {
        BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: F2");
    }
    if (*freq == 108) {
        BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: FIS2");
    }
    if (*freq == 102) {
        BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: G2");
    }
    if (*freq == 96) {
        BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: GIS2");
    }
    if (*freq == 91) {
        BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: A2");
    }
    if (*freq == 86) {
        BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: AIS2");
    }
    if (*freq == 81) {
        BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: B2");
    }
    if (*freq == 0) {
        BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Pitch: Pause");
    }
    BSP_LCD_SetTextColor(LCD_COLOR_WHITE);
    BSP_LCD_DisplayStringAtLine(2, (uint8_t*)"                          ");
    if (*option == 1) {
        BSP_LCD_SetTextColor(LCD_COLOR_RED);
    }
    if (*dur == (float)1.0) {
        BSP_LCD_DisplayStringAtLine(2, (uint8_t*)("Length: Full"));
    }
    if (*dur == (float)0.5) {
        BSP_LCD_DisplayStringAtLine(2, (uint8_t*)("Length: Half"));
    }
    if (*dur == (float)0.25) {
        BSP_LCD_DisplayStringAtLine(2, (uint8_t*)("Length: Quarter"));
    }
    if (*dur == (float)0.125) {
        BSP_LCD_DisplayStringAtLine(2, (uint8_t*)("Length: Eighth"));
    }
    if (*dur == (float)0.0625) {
        BSP_LCD_DisplayStringAtLine(2, (uint8_t*)("Length: Sixteenth"));
    }
    BSP_LCD_SetTextColor(LCD_COLOR_WHITE);
    BSP_LCD_DisplayStringAtLine(3, (uint8_t*)"                      ");
    if (*option == 2) {
        BSP_LCD_SetTextColor(LCD_COLOR_RED);
    }
    BSP_LCD_DisplayStringAtLine(3, (uint8_t*)"Confirm");
    BSP_LCD_SetTextColor(LCD_COLOR_WHITE);
    BSP_LCD_DisplayStringAtLine(4, (uint8_t*)"                      ");
    if (*option == 3) {
        BSP_LCD_SetTextColor(LCD_COLOR_RED);
    }
    BSP_LCD_DisplayStringAtLine(4, (uint8_t*)"Finish");
    BSP_LCD_SetTextColor(LCD_COLOR_WHITE);
```

```
	}
	if (mode == 2) {
		BSP_LCD_DisplayStringAtLine(0, (uint8_t*)"                    ");
		BSP_LCD_DisplayStringAtLine(0, (uint8_t*)"Song paused");
		BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"                    ");
		BSP_LCD_SetTextColor(LCD_COLOR_RED);
		BSP_LCD_DisplayStringAtLine(1, (uint8_t*)"Register a Song");
		BSP_LCD_SetTextColor(LCD_COLOR_WHITE);
		BSP_LCD_DisplayStringAtLine(2, (uint8_t*)"                    ");
		BSP_LCD_DisplayStringAtLine(3, (uint8_t*)"                    ");
		BSP_LCD_DisplayStringAtLine(4, (uint8_t*)"                    ");
	}
}
```

Funkcja Display jest odpowiedzialna za konfigurowanie wyświetlacza tak, aby prezentował z wykorzystaniem menu stan programu oraz wybierane z niego opcje.

```
struct Song registerSong(int *option, int *valuex, int *valuey, int *valueButton){
	struct Song nowa;
	int tmp = 0;
	nowa.length=0;
	int freq_index = 0;
	int len_index = 0;
	int freq_table[] = { 308, 290, 274, 256, 244, 230, 217, 204, 192, 182, 171, 163,
153, 144, 136, 128, 121, 114, 108, 102, 96, 91, 86, 81, 0};
	float len_table[] = { 1.0,0.5,0.25,0.125,0.0625 };
	Display(1, option, &freq_table[freq_index], &len_table[len_index]);
	while (Action(option, valuex, valuey, valueButton,3, &freq_index, &len_index) >
0 || nowa.length==0) {
		if (tmp==1) Display(1, option, &freq_table[freq_index],
&len_table[len_index]);
		if (*valueButton == 1 && *option == 2) {
			nowa.notes[nowa.length].freq = freq_table[freq_index];
			nowa.notes[nowa.length].dur = len_table[len_index];
			nowa.length += 1;
		}
		HAL_ADC_Start(&hadc1);
		HAL_ADC_Start(&hadc2);
		HAL_ADC_Start(&hadc3);
		HAL_Delay(20);
		if (
 readAnalogStickx(HAL_ADC_GetValue(&hadc1), valuex) ||
 readAnalogSticky(HAL_ADC_GetValue(&hadc2), valuey) ||
 readButton(valueButton)) tmp = 1;
		else tmp = 0;
	}
	*option=0;
	return nowa;
}
```

Funkcja registerSong jest wykorzystywana do rejestrowania melodii z wykorzystaniem menu. Funkcja odświeżania wyświetlacza jest wywoływana jedynie gdy wartość odczytana z urządzenia wejścia uległa zmianie, aby zapobiec ciągłym miganiom obrazu.

```
void playSong(struct Song song, float speed) {
	float basespeed;
	if (speed == 0) {
		basespeed = 0.5;
	}
	if (speed == 1) {
		basespeed = 1.0;
	}
	if (speed == 2) {
		basespeed = 2.0;
	}
	for (int i = 0; i < song.length; i++) {
		playNote(song.notes[i], basespeed);
	}
}
```

Funkcja playSong odpowiada za wyznaczanie szybkości odtwarzania melodii i odtwarzanie poszczególnych nut.

```
void playNote(struct Note note, float basespeed) {
       HAL_Delay(100);
       HAL_TIM_PWM_Stop(&htim3, TIM_CHANNEL_1);
       if (note.freq!=0) HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
       TIM3->ARR = note.freq;
       TIM3->CCR1 = TIM3->ARR * 0.5;
       HAL_Delay((int)1000*basespeed * note.dur);
}
```

Ostatnią funkcją jest playNote, odpowiada ona za odtworzenie pojedynczej nuty melodii w odpowiednim tonie przez określoną ilość czasu, z możliwością wyłączania generowania sygnału wejściowego w trakcie pauzy.

Funkcja przerwania

```
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_SET);
  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
  /* USER CODE BEGIN EXTI0_IRQn 1 */

  /* USER CODE END EXTI0_IRQn 1 */
}
```

Funkcja przerwania aktywowana podczas wciśnięcia niebieskiego przycisku na płytce. Odpowiada za zapalenie diody informującej o zatrzymaniu odtwarzania melodii i do samego jej zatrzymania.

```c
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC2_Init();
  MX_TIM6_Init();
  MX_TIM4_Init();
  MX_FMC_Init();
  MX_LTDC_Init();
  MX_DMA2D_Init();
  MX_I2C3_Init();
  MX_SPI5_Init();
  MX_ADC1_Init();
  MX_ADC3_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  int on=0; // Zmienna przechowująca stan przycisku włączania
  // Glosnik: PB4
  int valuex; // PA5
  int valuey; // PA7
  int option=0;
  int potval=0; // PF6
  int valueButton = 0; // PB13
  int phvi =0;
  float phvf =0;
  struct Song song;

  BSP_LCD_Init();
  BSP_LCD_LayerDefaultInit(LCD_BACKGROUND_LAYER, LCD_FRAME_BUFFER);
  BSP_LCD_Clear(LCD_COLOR_BLACK);
  BSP_LCD_SetBackColor(LCD_COLOR_BLACK);
  BSP_LCD_SetTextColor(LCD_COLOR_WHITE);
  Display(0, &option,&phvi,&phvf);
  BSP_LCD_DisplayStringAtLine(0, (uint8_t*)"                    ");
  BSP_LCD_DisplayStringAtLine(0, (uint8_t*)"No song recorded");
```

```
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1); //wartosc zegara prc: 79 counter_period:
999 pulse: 499
  while (1)
  {
	if (HAL_GPIO_ReadPin(GPIOG,  GPIO_PIN_14)==GPIO_PIN_SET) {
		if (on) {
			on = 0;
			Display(2, &option,0,0);
		}
		else {
			on = 1;
			Display(0, &option,0,0);
		}
		HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);
	}
	if (on) {
		HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
		playSong(song, potval);
	}
	else {
		HAL_TIM_PWM_Stop(&htim3, TIM_CHANNEL_1);
	}
	HAL_ADC_Start(&hadc1);
	HAL_ADC_Start(&hadc2);
	HAL_ADC_Start(&hadc3);
	readAnalogStickx(HAL_ADC_GetValue(&hadc1), &valuex);
	readAnalogSticky(HAL_ADC_GetValue(&hadc2), &valuey);
	readButton(&valueButton);
	if (Action(&option, &valuex, &valuey, &valueButton, 0,0,0) == 2) {
		song = registerSong(&option, &valuex, &valuey, &valueButton);
		on = 1;
		Display(0, &option,0,0);
	}
	readPotentiometer(&potval);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Przed wykonywaniem polecenia while w funkcji main są definiowane zmienne pomocnicze wykorzystywane w trakcie działania programu, które mają przechowywać wartości odczytywane z urządzeń wejściowych oraz informacje o stanie aplikacji. W pętli while wykorzystywane są wszystkie wcześniej zdefiniowane funkcje.

Grzegorz Ryński
Systemy wbudowane
Projekt