

Dokumentacja

Marcin Radecki

Wojciech Urbanek

Modulator dźwięków

Temat projektu.

Tematem projektu było stworzenie modulatora dźwięków na podstawie mikrokontrolera STM32F4 discovery. Za pojęciem modulator mamy na myśli odtwarzanie wcześniej nagranych dźwięków poprzez wbudowany mikrofon z inną częstotliwością wyjściową co w efekcie daje zmodulowany dźwięk.

Wykonanie i użytkowanie.

Projekt został wykonany zgodnie ze wcześniejszymi założeniami.

Do poprawnego działania oraz przetestowania będą potrzebne nam:

- mikrokontroler STM32F4 discovery
- słuchawki lub głośniki (mini jack 3,5 mm)
- komputer
- kabel micro USB

Program posiada wbudowane 3 tryby które obsługują nasz modulator. Aby przejść między trybami poruszać należy się wbudowanym przyciskiem.

Nasze tryby to:

- 1) Test wydajności mikrofonu oraz słuchawek. Ma to na celu sprawdzenie czy nasz sprzęt działa poprawnie. Mikrokontroler nagrywa wszelkie dźwięki z otoczenia następnie bez zapisywania ich wypuszcza bezpośrednio na wyjście słuchawkowe.
- 2) Wyższa częstotliwość – w trybie tym odtworzymy nasz wcześniej nagrany dźwięk z nieco większą szybkością.
- 3) Niższa częstotliwość – w trybie tym odtwarzany dźwięk jest nieco bardziej spowolniony co daje wrażenie usłyszenia głosu porywacza (niski, gruby głos).

W trybie 2 oraz 3 nagrywany dźwięk może mieć długość tylko 5 sekund gdyż wielkość pamięci wbudowanej do mikrokontrolera to 1MB. Jednak nie przeszkadza to w sprawnym przetestowaniu działania programu.

Wykorzystywane biblioteki.

- pdm_filter.h

Biblioteka ta służy do dekodowania i rekonstrukcji sygnału audio, który wytworzony jest przez mikrofon.

Uwagi końcowe:

Program w całości został napisany przy użyciu środowiska CoCoX.

Najważniejsze funkcje.

inicjowanie **WaveRecordera** – ustala on pasmo, w którym mieści się sygnał oraz częstotliwość poprzez filtr

```
uint32_t WaveRecorderInit(uint32_t AudioFreq, uint32_t BitRes, uint32_t ChnlNbr)
{
    /* Check if the interface is already initialized */
    if (AudioRecInitd)
    {
        /* No need for initialization */
        return 0;
    }
    else
    {
        /* Enable CRC module */
        RCC->AHB1ENR |= RCC_AHB1ENR_CRCEN;

        /* Filter LP & HP Init */
        Filter.LP_HZ = 8000;
        Filter.HP_HZ = 10;
        Filter.Fs = 16000;
        Filter.Out_MicChannels = 1;
        Filter.In_MicChannels = 1;

        PDM_Filter_Init((PDMFilter_InitStruct *)&Filter);

        /* Konfiguracja GPIO dla układu MP45DT02 zawierającego mikrofon wielokierunkowy */
        Inicjacja_GPIO_dla_MP45DT02();

        /* Konfiguracja przerwania */
        Konfiguracja_przerwan_NVIC();

        /* Configure the SPI */
        Konfiguracja_SPI(AudioFreq);

        /* Set the local parameters */
        AudioRecBitRes = BitRes;
        AudioRecChnlNbr = ChnlNbr;

        /* Set state of the audio recorder to initialized */
    }
```

```
AudioRecInited = 1;
```

```
/* Return 0 if all operations are OK */
```

```
return 0;
```

```
}
```

```
}
```

b)

```
void Przerywanie_od_MP45DT02(void)
```

```
{
```

```
u16 app; //otrzymywane dane
```

```
u16 volume;
```

```
/*Check if data are available in SPI Data register otrzymywane dane */
```

```
if (SPI_GetITStatus(SPI2, SPI_I2S_IT_RXNE) != RESET)
```

```
{
```

```
app = SPI_I2S_ReceiveData(SPI2);
```

```
InternalBuffer[InternalBufferSize++] = HTONS(app);
```

```
if (InternalBufferSize >= INTERNAL_BUFF_SIZE)
```

```
{
```

```
InternalBufferSize = 0;
```

```
volume = 20;
```

```
PDM_Filter_64_LSB((uint16_t *)InternalBuffer, (uint16_t *)pAudioRecBuf, volume ,  
(PDMFilter_InitStruct *)&Filter);
```

```
Data_Status = 1;
```

```
if (Switch ==1)
```

```
{
```

```
pAudioRecBuf = RecBuf;
```

```
writebuffer = RecBuf1;
```

```
Switch = 0;
```

```
}
```

```
else
```

```
{
```

```
pAudioRecBuf = RecBuf1;
```

```
writebuffer = RecBuf;
```

```
Switch = 1;
```

```
}
```

```
for(counter=0;counter<16;counter++)
```

```
{
```

```
if(Switch==0)
```

```
RecBufPCM[RecBufIter]=RecBuf1[counter];
```

```
if(Switch==1)
```

```
RecBufPCM[RecBufIter]=RecBuf[counter];
```

```
RecBufIter++;
```

```
if(RecBufIter==SIZE_GENERAL_BUF)
```

```
{
```

```
RecBufIter=0;
```

```
if(button==3 || button==2)
```

```
{
```

```
WaveRecorderStop();
```

```

        }
        if(button==3)aktyw_hi=1;
        if(button==2)aktyw_low=1;
    }
    /* Gdy bufor się zapełnia transmisja danych jest zatrzymywana i ustalany jest
    odpowiedni tryb*/
    }}}}

```

Powyższy kod służy do obsługi mikrofonu oraz konwersji danych z formatu PDM na PCM. Zapisuje on również nagrany dźwięk.

e) wysyłanie na wyjście słuchawkowe oraz zamiana częstotliwości próbki

```

if(button==1)
{

//NAGRYWANIE

if(lock==0 && RecBufIter==16)
{
    I2S_Cmd(CODEC_I2S, ENABLE);
    lock=1;
    aktyw=1;
}

if (SPI_I2S_GetFlagStatus(CODEC_I2S, SPI_I2S_FLAG_TXE) && aktyw==1)
{
    sample=RecBufPCM[sampleCounter];

    SPI_I2S_SendData(CODEC_I2S, sample);
    sampleCounter++;
}

if (sampleCounter==SIZE_GENERAL_BUF/2)
{
    GPIOD->BSRRL = GPIO_Pin_12;
}

else if (sampleCounter == SIZE_GENERAL_BUF)
{
    GPIOD->BSRRH = GPIO_Pin_12;
    sampleCounter = 0;
}

}

if(button==2 && aktyw_low==1)
{

I2S_Cmd(CODEC_I2S, DISABLE);

//ZMIANA GŁOSU

SPI_I2S_DeInit(CODEC_I2S);
I2S_InitType.I2S_AudioFreq = I2S_AudioFreq_7k;//11
I2S_InitType.I2S_MCLKOutput = I2S_MCLKOutput_Enable;
I2S_InitType.I2S_DataFormat = I2S_DataFormat_16b;
I2S_InitType.I2S_Mode = I2S_Mode_MasterTx;
I2S_InitType.I2S_Standard = I2S_Standard_Phillips;

```

```

I2S_InitType.I2S_CPOL = I2S_CPOL_Low;

I2S_Init(CODEC_I2S, &I2S_InitType);
I2S_Cmd(CODEC_I2S, ENABLE);

int i=0;
sampleCounter=0;
for(i=0;i<SIZE_GENERAL_BUF;)
{

    if (SPI_I2S_GetFlagStatus(CODEC_I2S, SPI_I2S_FLAG_TXE))
    {
        sample=RecBufPCM[sampleCounter];

        SPI_I2S_SendData(CODEC_I2S, sample);
        sampleCounter++;
        i++;
    }

    if (sampleCounter==SIZE_GENERAL_BUF/2)
    {
        GPIOD->BSRRL = GPIO_Pin_12;
    }

    else if (sampleCounter == SIZE_GENERAL_BUF)
    {
        GPIOD->BSRRH = GPIO_Pin_12;
        sampleCounter = 0;
    }
}
SPI_I2S_SendData(CODEC_I2S, 0);
aktyw_low=0;
}

if(button==3 && aktyw_hi==1)
{
    I2S_Cmd(CODEC_I2S, DISABLE);

    //ZMIANA GŁOSU

    SPI_I2S_DeInit(CODEC_I2S);
    I2S_InitType.I2S_AudioFreq = I2S_AudioFreq_5k;//11
    I2S_InitType.I2S_MCLKOutput = I2S_MCLKOutput_Enable;
    I2S_InitType.I2S_DataFormat = I2S_DataFormat_16b;
    I2S_InitType.I2S_Mode = I2S_Mode_MasterTx;
    I2S_InitType.I2S_Standard = I2S_Standard_Phillips;
    I2S_InitType.I2S_CPOL = I2S_CPOL_Low;

    I2S_Init(CODEC_I2S, &I2S_InitType);
    I2S_Cmd(CODEC_I2S, ENABLE);

    int i=0;
    sampleCounter=0;
    for(i=0;i<SIZE_GENERAL_BUF;)
    {

        if (SPI_I2S_GetFlagStatus(CODEC_I2S, SPI_I2S_FLAG_TXE))
        {
            sample=RecBufPCM[sampleCounter];

```

```
SPI_I2S_SendData(CODEC_I2S, sample);
sampleCounter++;
i++;
}

else if (sampleCounter == SIZE_GENERAL_BUF)
{
    sampleCounter = 0;
}
}
aktyw_hi=0;
}
}}
```