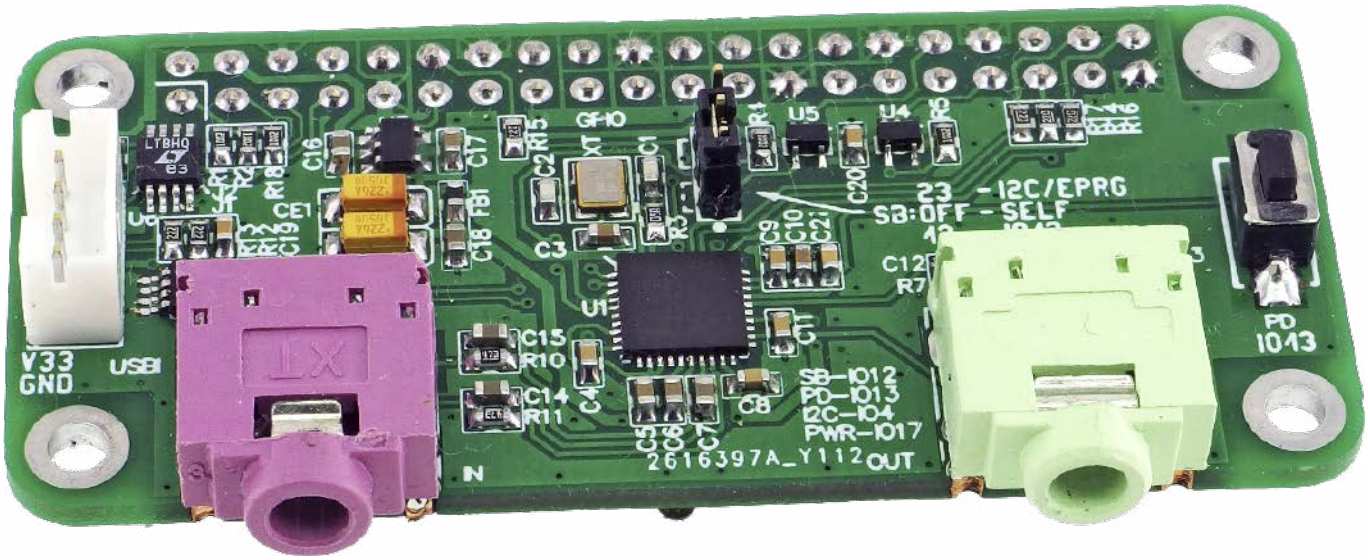


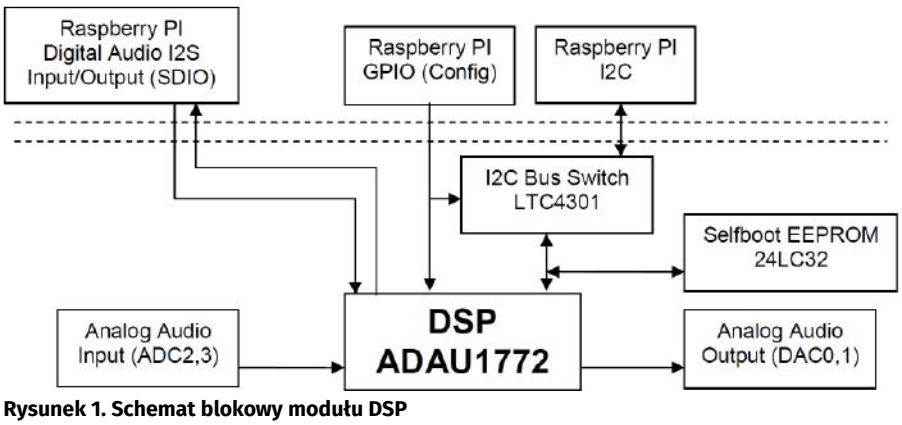
Moduł DSP Audio do Raspberry Pi

Połączenie Raspberry Pi i procesora DSP umożliwia tworzenie wbudowanych aplikacji multimedialnych o funkcjach nieosiągalnych przy zastosowaniu zwykłego DAC.

Moduł DSP zbudowany jest w oparciu na nowoczesnym układzie DSP firmy Analog Devices typu ADAU1772. Procesor jest konfigurowany i programowany za pomocą graficznego środowiska Sigma Studio, gdzie aplikacje tworzy się poprzez łączenie gotowych bloków funkcjonalnych. Docelowe parametry toru DSP mogą być zmieniane procesorem nadrzędnym, czyli bezpośrednio z Raspberry Pi.

Schemat blokowy modułu został pokazany na **rysunku 1**. Projekt zakładał osiągnięcie możliwie największej uniwersalności. Procesor DSP typu ADAU1772, którego budowę wewnętrzną pokazuje **rysunek 2**, jest odpowiedzialny za obróbkę stereofonicznego sygnału audio w torze analogowym, z wykorzystaniem wbudowanych przetworników ADC i DAC audio. Jest połączony z cyfrowym interfejsem audio I²S płytki Raspberry Pi, co dodatkowo umożliwia obróbkę sygnału z pomocą komputerka lub pracę w roli przetwornika ADC/DAC audio z wbudowanym procesorem DSP.

Elastyczna konfiguracja modułu przewiduje pracę procesora ADAU1772 w trybie Selfboot, gdzie aplikacja i ustawienia pobierane są z pamięci EEPROM każdorazowo po włączeniu zasilania i może zostać modyfikowana na bieżąco poprzez magistralę I²C. Drugim trybem pracy jest ładowanie parametrów i przygotowanego algorytmu bezpośrednio do pamięci DSP z pominięciem EEPROM przez aplikację z Raspberry Pi.



Rysunek 1. Schemat blokowy modułu DSP

Budowa i działanie

Układ procesora U1 jest taktowany rezonatorem XT o częstotliwości 12,288 MHz zapewniając wzorcowy sygnał zegarowy zgodny z typowymi

Dodatkowe materiały do pobrania ze strony www.media.avt.pl

W ofercie AVT* AVT----

Podstawowe parametry:

- zawiera procesor DSP typu ADAU1772 z wbudowanymi przetwornikami ADC i DAC audio,
- procesor pracuje w trybie Selfboot, czyli aplikacja i ustawienia pobierane są z pamięci EEPROM przy każdym uruchomieniu modułu,
- docelowe parametry toru DSP mogą być zmieniane procesorem nadrzędnym, czyli bezpośrednio z Raspberry Pi.

Projekty pokrewne na www.media.avt.pl:

AVT-5858	Płytka bazowa dla Raspberry Pi Pico (EP 5/2021)
AVT-5851	Dwukanałowy port szeregowy dla Raspberry (EP 3/2021)
AVT-5847	Interfejs wyświetlacza TFT-RGB dla RPi Zero (EP 3/2021)
----	Sterownik 18 LED dla Pi Zero (EP 2/2021)
AVT-5837	Moduł do pomiaru napięcia i prądu z interfejsem I²C (EP 1/2021)
AVT-5811	Odtwarzacz audio z Raspberry Pi (EP 10-12/2020)

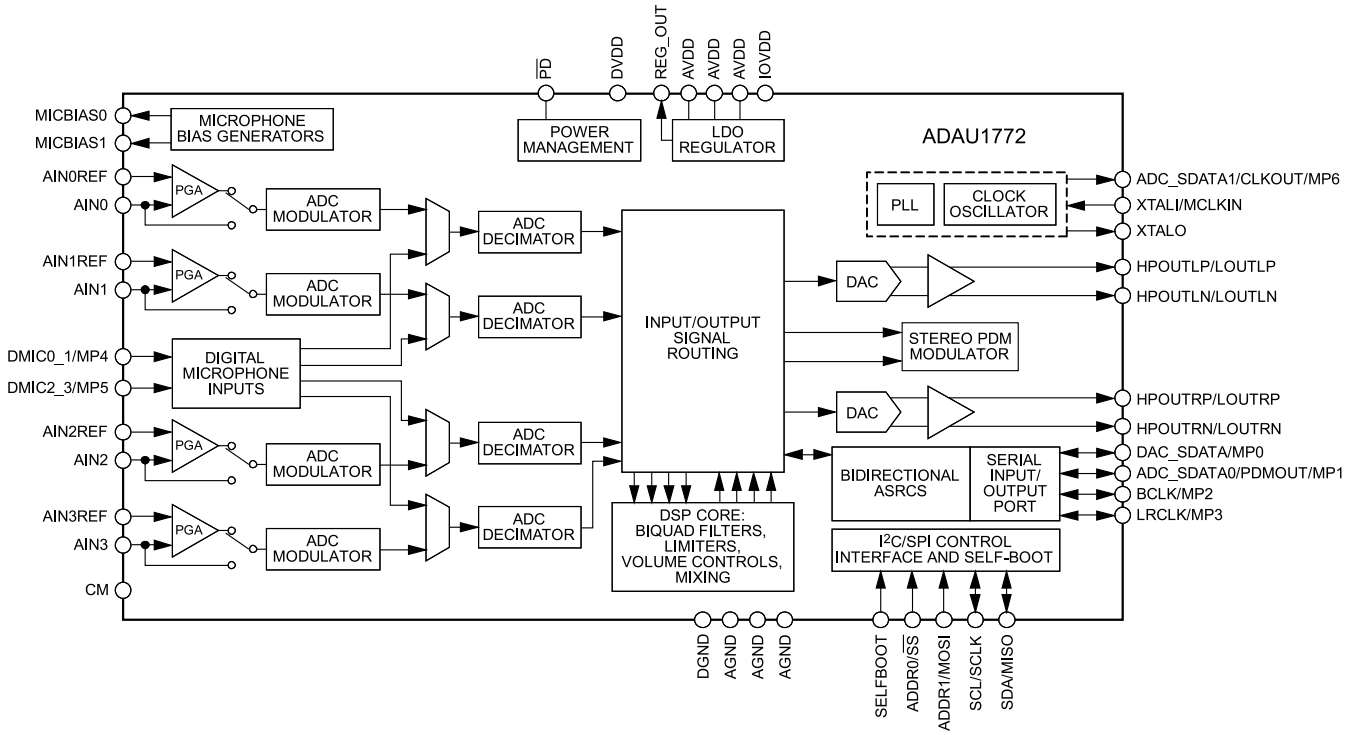
*** Uwaga!** **Elektroniczne zestawy do samodzielnego montażu. Wymagana umiejętność lutowania!**
Podstawową wersją zestawu jest wersja [**B**] nazywana potocznie KIT-em (z ang. zestaw). Zestaw w wersji [**B**] zawiera elementy elektroniczne (w tym [**UK**] – jeśli występuje w projekcie), które należy samodzielnie wlutować w dołączoną płytkę drukowaną (PCB). Wykaz elementów znajduje się w dokumentacji, która jest podlinkowana w opisie kitu.
Mając na uwadze różne potrzeby naszych klientów, oferujemy dodatkowe wersje:

- wersja [**C**] – zmontowany, uruchomiony i przetestowany zestaw [**B**] (elementy wlutowane w płytkę PCB)
- wersja [**A**] – płytka drukowana bez elementów i dokumentacji

Kity w których występuje układ scalony wymagający zaprogramowania, mają następujące dodatkowe wersje:

- wersja [**A+**] – płytka drukowana [**A**] + zaprogramowany układ [**UK**] i dokumentacja
- wersja [**UK**] – zaprogramowany układ

Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik pdf! Podczas składania zamówienia upewnij się, którą wersję zamawiasz!
http://sklep.avt.pl. W przypadku braku dostępności na *http://sklep.avt.pl*, osoby zainteresowane zakupem płytek drukowanych (PCB) prosimy o kontakt via e-mail: kity@avt.pl.



Rysunek 2. Schemat blokowy układu ADAU1772

częstotliwościami próbkowania audio. Wejściowy analogowy sygnał audio z wejścia IN jest doprowadzony do przetworników ADC2 i ADC3 układu. Analogowy sygnał wyjściowy z przetworników DAC0 i DAC1 bloku DSP doprowadzony jest do wyjścia OUT. Oba gniazda są standardzie mini jack stereo 3,5 mm.

Sygnał cyfrowego audio I²S ze złącza GPIO Raspberry Pi jest doprowadzony do portów szeregowych SDI/SDO procesora ADAU1772. W przypadku, gdy częstotliwość próbkowania obrabianego sygnału cyfrowego nie jest zgodna z ustalonym próbkowaniem *fs* w DSP, należy użyć bloków konwerterów częstotliwości próbkowania ASRC wbudowanych w ADAU1772.

Wykaz elementów:
Rezystory: (SMD0603, 1%)
R1, R2, R4, R5, R18: 10 kΩ
R3, R7, R9: 100 Ω
R6, R8, R10, R11: 47 kΩ
R12, R13: 2,2 kΩ
R14…R17: 22 Ω

Kondensatory:
C1, C2: 22 pF SMD0603
C3, C10: 2,2 µF SMD0603
C4, C8, C9, C11, C19, C20, C22: 0,1 µF SMD0603
C5…C7, C12…C18: 10 µF SMD0603
C21: 1 nF SMD0603
CE1, CE2: 47 µF SMD3216 tantalowy 10 V

Półprzewodniki:
U1: ADAU1772BCPZ (LFCSP40)
U2: 24LC32A (MSOP8)
U3: MCP1812AT-033 (SOT-23-5)
U4, U5: 74LVC1G06DCK (SC70-5)
U6: LTC4301CMS8 (MSOP8)

Pozostałe:
XT: rezonator kwarcowy 12,288M-SR CFPX-180 SMD
FB1: koralik ferrytowy BLM18AG121SN1D SMD603
GPIO: złącze żeńskie IDC40
USBI: złącze PH4 2 mm proste
IN, OUT: gniazdo mini jack stereo FC68133
SB: lista SIP3 2 mm + zwora
J1, J2: zwora PCB, opis w tekście
PD: mikroprzełącznik

Procesor jest zasilany napięciem V33 (3,3 V) uzyskiwanym z linii zasilającej V50 (+5 V) modułu Raspberry Pi oraz generowanym wewnętrzne napięciem DVDD (1,1/1,2 V) zasilającym rdzeń DSP. Napięcie V33 i dodatkowo filtrowane zasilanie części analogowej DSP V33A dostarczane jest przez stabilizator LDO U3 typu MCP1812AT-033. Układ U3 wyposażony jest w funkcję rozładowania podłączonych pojemności wyjściowych, co eliminuje problemy z prawidłowym restartem U1. Napięcie wyjściowe stabilizatora kluczowane jest sygnałem PWR17 z lini GPIO17 Raspberry Pi. Podanie stanu wysokiego włączy stabilizator U3 i zasilanie DSP. Sygnał PDE13 z linii GPIO13 umożliwia sterowanie wyprowadzeniem !PD układu U1 odpowiedzialnym za wprowadzenie procesora w tryb obniżonego poboru mocy. Tryb PD może też zostać wyzwolony przyciskiem PD.

Inwerter U4 z otwartym kolektorem sumuje sygnały z GPIO13 i przycisku PB. Ustawienie linii GPIO13 w stan wysoki wprowadza U1 w tryb obniżonego poboru mocy niezależnie od stanu przycisku PD.

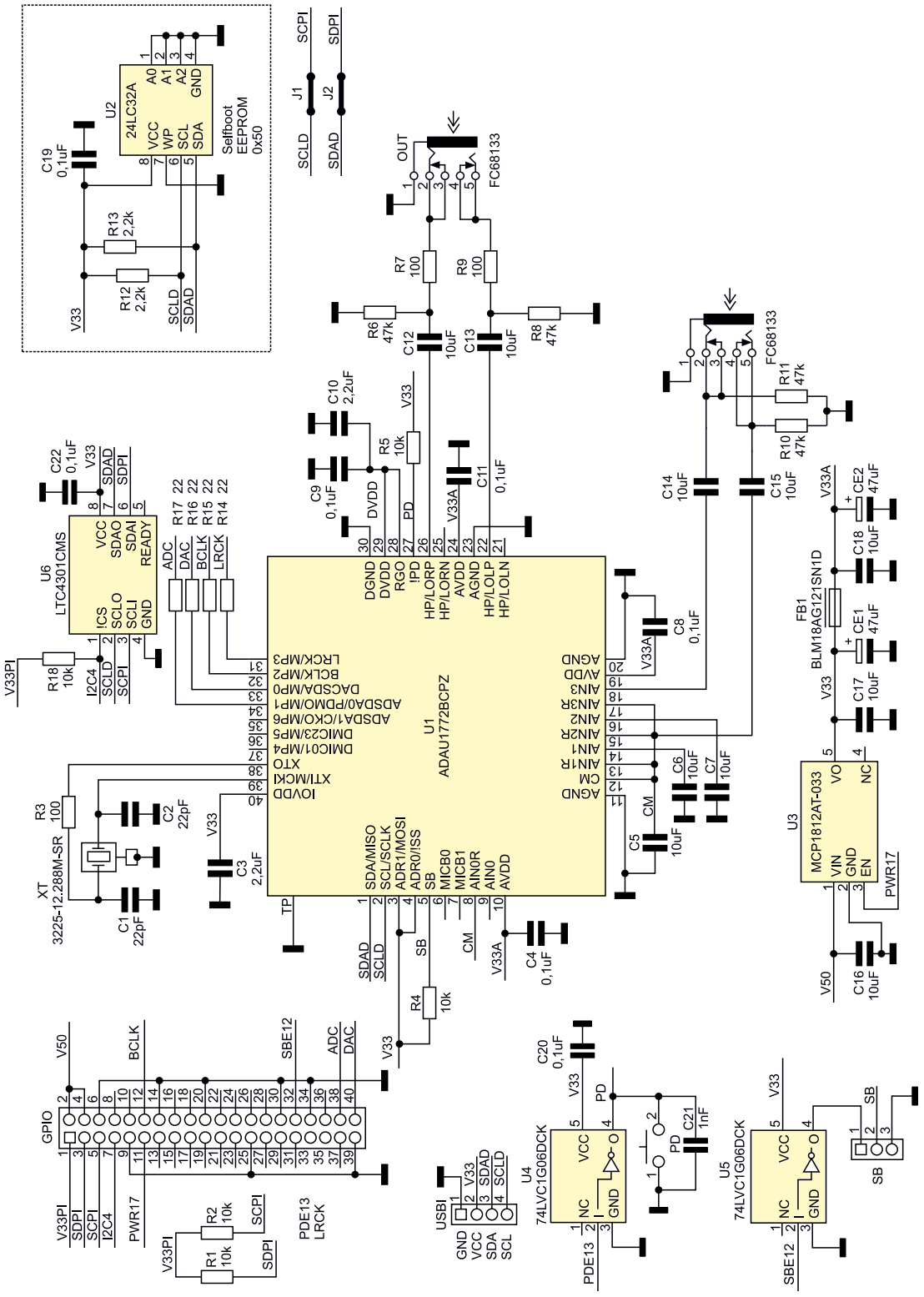
Tryb pracy DSP jest wybierany stanem wyprowadzenia 27 układu U1. Tryb Selfboot, w którym procesor pracuje samodzielnie, a aplikacja jest ładowana z pamięci EEPROM po włączeniu zasilania U2, aktywowany jest stanem wysokim sygnału SB. Stan niski podczas włączenia zasilania konfiguruje układ do pracy z ładowaniem aplikacji z I²C (lub z programatora USBi). Do wyboru trybu pracy służy zwora SB, która rozwarta aktywuje Selfboot, ze zwartymi wyprowadzeniami 2–3 tryb I²C, a ze zwartymi 1–2 przekazuje sterowanie trybem na wyprowadzenie GPIO12 (sygnał PGE12). Stan niski GPIO12 aktywuje tryb Selfboot. Zachowanie odpowiednich sekwencji sygnałów PWR17, PDE13, SBE12 umożliwia wprowadzenie ADAU1772 w wymagany tryb pracy.

Ostatnim, ale bardzo istotnym elementem jest układ U6 typu LTC4301CMS. Jest to sterowany bufor magistrali I²C separujący magistrale Raspberry Pi i ADAU1772 podczas ładowania programu z pamięci EEPROM. W przypadku bezpośredniego połączenia na magistrali dwóch układów master bez arbitrażu nastąpi zablokowanie magistrali. Wymuszony programowo niski stan sygnału I2C4 z GPIO4 aktywuje bufory i załączy magistralę I²C, gdy ADAU1772 zakończy cykl Selfboot. Połączenie realizowane jest dopiero po wykryciu sekwencji STOP lub IDLE magistrali, aby nie zakłócać jej działania. Rozłączenie magistral umożliwia także programowanie pamieci EEPROM lub konfigurację ADAU1772 poprzez złącze USBi zewnętrznym programatorem USBi.

Montaż i uruchomienie

Moduł jest zmontowany na niewielkiej dwustronnej płytce drukowanej zgodnej ze standardem Raspberry Pi Zero. Jej schemat wraz z rozmieszczeniem elementów został pokazany na **rysunku 4**. Sposób montażu jest klasyczny i nie wymaga dokładnego opisu, gotowa płytka wygląda tak jak na fotografii tytułowej.

Prawidłowo zmontowany moduł nie wymaga uruchamiania i po konfiguracji sprzętowo-programowej może być użyty w docelowej aplikacji. Przykładowym projektem postaram się ułatwić rozpoczęcie eksperymentów z DSP. Projekt będzie obsługiwał



Rysunek 3. Schemat ideowy modułu DSP

miksowanie sygnału ze źródła analogowego z wejścia IN z sygnałem cyfrowym I²S pochodzącym z Raspberry Pi.

Do oprogramowania procesora DSP potrzebne będzie środowisko Sigma Studio i programator USBi, a do konfiguracji komputerek Raspberry Pi z aktywnym I²C i I²S DAC. W systemie Pi należy skonfigurować DAC, zgodny z PCM5102. Na wszelki wypadek należy zaktualizować system poleceniami:

```
sudo rpi-update
sync
sudo reboot
```

Następnie należy:

- usunąć z pliku */etc/modprobe.d/raspi-blacklist.conf* linie:

```
blacklist i2c-bcm2708
blacklist snd-soc-pcm512x
blacklist snd-soc-wm8804
```

- usunąć driver w pliku */etc/modules:*

```
snd_bcm2835
```

- skonfigurować DAC, zgodny z PCM5102, dodając w pliku */boot/config.txt*

```
dtoverlay=hifiberry-dac
```

- skonfigurować ALSA, tworząc plik */etc/asound.conf* z zawartością:

```
pcm.!default {
    type hw card 0
}
ctl.!default {
    type hw card 0
}
```

Następnie należy zrestartować PI i po uruchomieniu sprawdzić poprawność konfiguracji poleceniem:

```
sudo aplay -l
```

DAC powinien pojawić się na liście dostępnych urządzeń odtwarzających:

```
**** List of PLAYBACK Hardware
Devices ****
card 0: sndrpihifiberry [snd_
rpi_hifiberry_dac], device 0:
HifiBerry DAC HiFi pcm5102a-
hifi-0 []
Subdevices: 1/1
Subdevice #0: subdevice #0
```

Do odtwarzania plików muzycznych, z racji uruchamiania w konsoli, używany będzie program aplay. Należy też przygotować kilka plików testowych audio 44,1 kHz w formacie **.wav*, najlepiej z muzyką, której słuchanie nie sprawia nam przykrości. Potrzebne będzie też źródło sygnału analogowego np. wyjście audio z PC oraz wzmacniacz lub głośniki PC do odsłuchu efektów.

Przed przygotowaniem projektu w Sigma Studio, sprawdzimy poprawność sterowania DSP z poziomu Raspberry Pi:

- konfigurujemy sterowanie zasilaniem GPIO17 jako wyjście:

```
echo 17 > /sys/class/gpio/export
echo out > /sys/class/gpio/
gpio17/direction
```

- sprawdzamy kluczowanie zasilania poleceniami:

```
echo 1 > /sys/class/gpio/
gpio17/value
echo 0 > /sys/class/gpio/
gpio17/value
```

Ustawienie stanu 1 załączy, a stanu 0 wyłączy +3,3 V dla DSP;

- następnie sprawdzamy sterowanie GPIO13 sygnałem PD, ustawiając go jako wyjście:

```
echo 13 > /sys/class/gpio/
export
echo out > /sys/class/gpio/
gpio13/direction
```

- sprawdzamy sterowanie PD poleceniami:

```
echo 1 > /sys/class/gpio/
gpio13/value
echo 0 > /sys/class/gpio/
gpio13/value
```

Ustawienie stanu 1 wprowadzi DSP w stan obniżonego poboru mocy, a stan 0 aktywuje DSP;

- następnie w położeniu zwory SB 1-2 sprawdzamy poprawność sterowania trybem pracy DSP, ustawiając GPIO12 jako wyjście:

```
echo 12 > /sys/class/gpio/
export
echo out > /sys/class/gpio/
gpio12/direction
```

- sprawdzamy sterowanie sygnałem SB poleceniami:

```
echo 1 > /sys/class/gpio/
gpio12/value
echo 0 > /sys/class/gpio/
gpio12/value
```

- ustawiamy stany:

```
echo 1 > /sys/class/
gpio/gpio17/value
echo 0 > /sys/class/
gpio/gpio13/value
echo 0 > /sys/class/
gpio/gpio12/value
```

Teraz sprawdzamy działanie bufora LTC4301 poleceniem:

```
i2cdetect -y 1
```

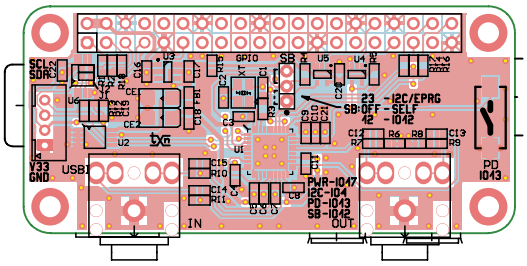
Na magistrali nie powinien pojawić się adres 0x3F DSP lub 0x50 pamięci EEPROM. Aktywując bufor poleceniami:

```
echo 4 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio4/
direction
echo 0 > /sys/class/gpio/gpio4/
value
```

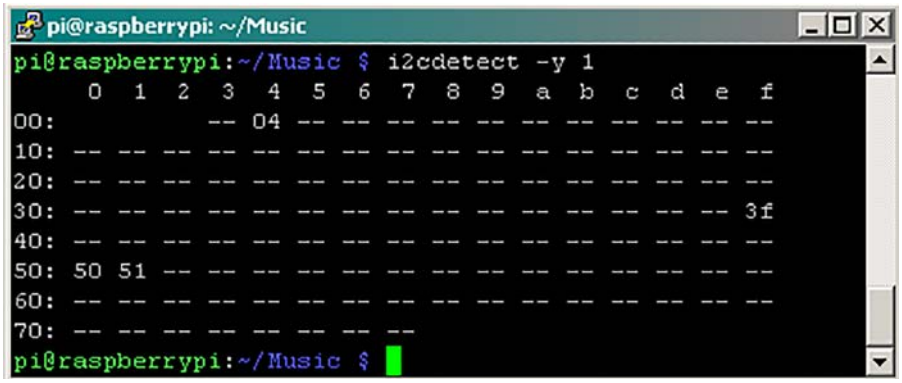
i ponawiając polecenie:

```
i2cdetect-y 1
```

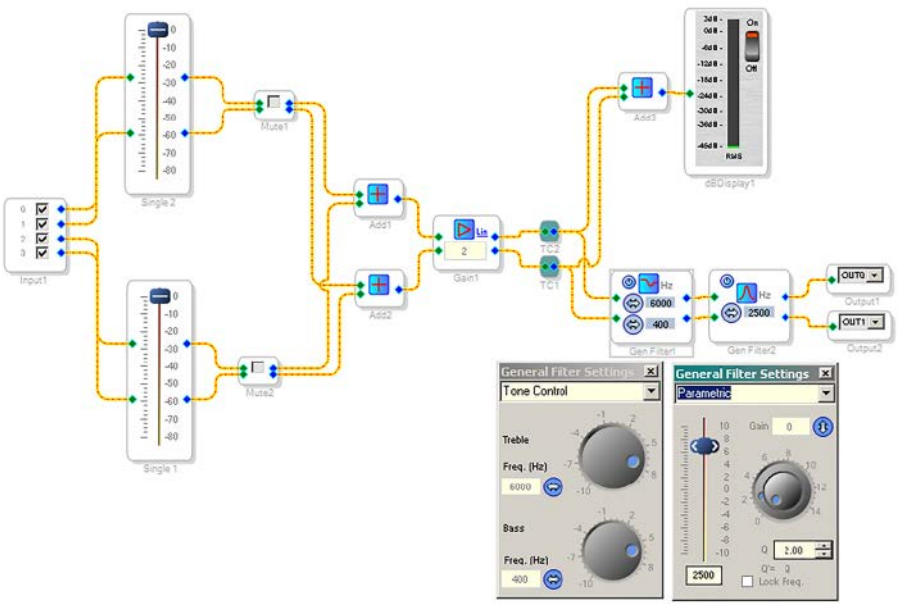
powinniśmy wykryć wszystkie układy modułu, co pokazano na **rysunku 5**. Pod adresem 0x3F widoczny jest ADAU1772, 0x50 pamięć EEPROM 24LC32, a 0x51 zajmuje programator USBi. UWAGA: pod adres 0x51 nie należy niczego zapisywać, bo uszkodzimy firmware programatora USBi.



Rysunek 4. Schemat płytki PCB wraz z rozmieszczeniem elementów



Rysunek 5. Detekcja układów RaspbPI_Zero_DSP



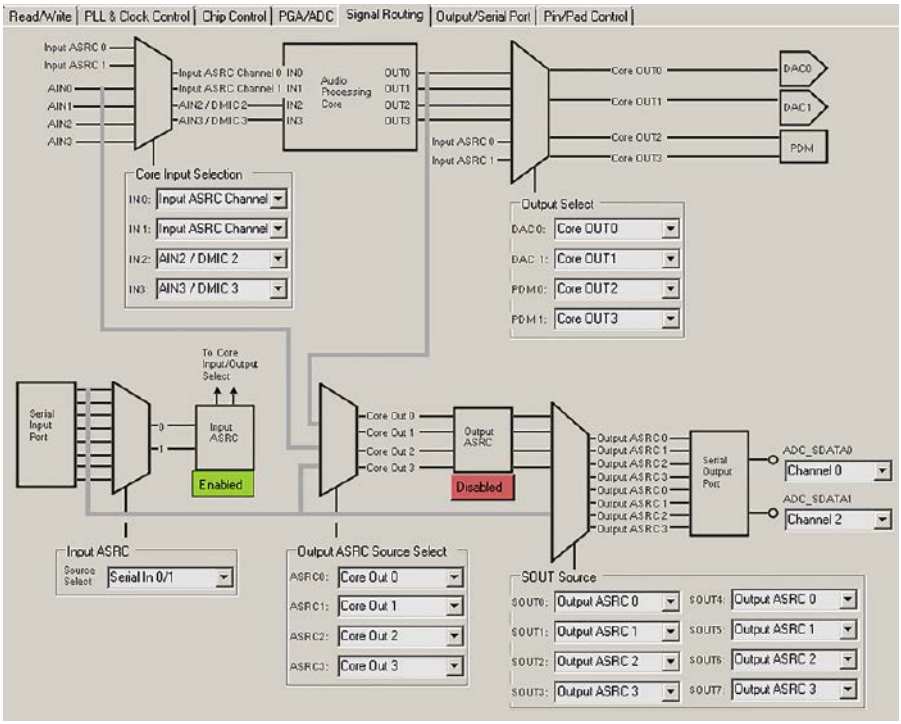
Rysunek 6. Aplikacja testowa DSP

W tym momencie można optymistycznie przyjąć, że sprzęt jest sprawny i należy zabrać się do oprogramowania, zakładając zworę SB w położenie 2–3 i odłączając I²C od procesora poleceniem:

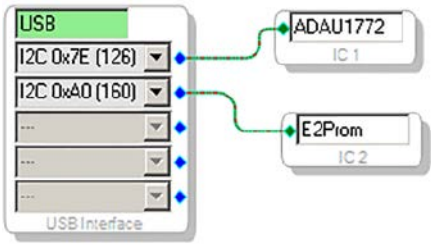
```
echo 1 > /sys/class/gpio/gpio4/
value
```

Aplikacja testowa

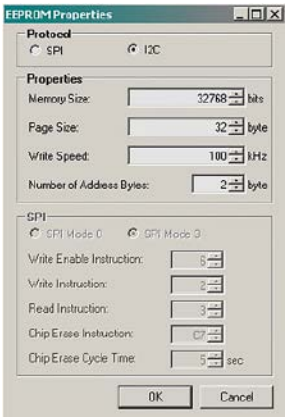
Schemat aplikacji testowej *ADAU1772_DSP-DAC.dspproj* przygotowanej w Sigma Studio został pokazany na **rysunku 6**. Źródła tego projektu są dostępne w materiałach dodatkowych. Stereofoniczne sygnały z bloku wejść cyfrowych (wejście 0 i 1) i analogowych (wejście 2 i 3) z bloku Input1 doprowadzone



Rysunek 7. Konfiguracja sygnałów DSP



Rysunek 8. Konfiguracja USBi



Rysunek 9. Konfiguracja EEPROM

są do regulatorów poziomu Single1/2. Następnie poprzez bloki wyciszenia Mute1/2 są sumowane w blokach Add1/2 i wzmocnione blokiem Gain1, skąd doprowadzone są do bloku miernika poziomu dBDisplay1 i na blok parametryzowanych filtrów Gen Filter1/2, a następnie do wyjść analogowych OUT0/1. Poziom mierzony przez blok dbDisplay1 jest przechowywany w rejestrze DBREG0 i może zostać użyty w aplikacji sterującej np. do sprawdzenia obecności sygnału audio.

Pierwszy z filtrów realizuje funkcję regulatora barwy dźwięku, drugi regulatora parametrycznego. Schemat konfiguracji związanej z przepływem obrabianych sygnałów pokazano na **rysunku 7**. Sygnały z wejściowego interfejsu cyfrowego I²S (Serial In 0/1) poddane są konwersji częstotliwości próbkowania ASRC i razem z sygnałami z ADC2/3 doprowadzone są do rdzenia DSP pracującego z ustawioną częstotliwością próbkowania fs i po obróbce wyprowadzone są na przetworniki DAC0 i DAC1 z niezmienionym fs.

W pozostałych zakładkach skonfigurowane są bloki ADC/PGA/DAC oraz PLL i zasilanie układu. Aby wgrać aplikację do DSP, konieczne jest skonfigurowanie USBi i systemu zgodnie z **rysunkiem 8**. Z sygnałów programatora dostępnych na kablu USBi typu IDC10 łączymy z gniazdem USBi modułu tylko magistralę I²C: SDA, SCL oraz masę programatora, zachowując możliwie krótkie przewody.

Konfigurujemy system: procesor ADAU1772 z ustawionym adresem 0x7E (0x3F) i pamięć EEPROM z adresem 0xA0 (0x50). Należy też ustawić typ i parametry pamięci zgodnie z **rysunkiem 9**, (prawy klawisz na ikonie IC2). W zakładce IC2 WinE2PromLoader należy pamięć wyczyścić (Clear E2Pro) i zaprogramować jej zawartość skompilowanym projektem, wracając do zakładki Config i wybierając opcję Write Latest Compilation to E2Prom (prawy klawisz na ikonie IC1).

Po założeniu zwory SB w położenie 2–3 (sterowanie z GPIO) restartujemy DSP sekwencją sygnałów PD/PWR/SB/PWR/PD poleceniami:

```
echo 1 > /sys/class/gpio/gpio13/value
echo 0 > /sys/class/gpio/gpio17/value
echo 0 > /sys/class/gpio/gpio12/value
echo 1 > /sys/class/gpio/gpio17/value
echo 0 > /sys/class/gpio/gpio13/value
```

Jeżeli do wejścia analogowego mamy podłączony sygnał audio, do wyjścia głośnikowego powinien docierać słyszalny sygnał testowy. Teraz możemy sprawdzić miksowanie sygnału z I²S Raspberry Pi, odtwarzając przygotowany plik testowy *01.wav*:

```
aplay 01.wav
```

Zostanie on zmiksowany z sygnałem testowym z wejścia IN. Jeżeli wstępne testy przebiegły pomyślnie, procesor jest zaprogramowany i działający, pozostaje przetestowanie komunikacji i sterowania DSP przez I²C poleceniem:

```
echo 0 > /sys/class/gpio/gpio4/value
```

Załączmy bufor, dla pewności sprawdzamy obecność układów na magistrali:

```
i2cdetect-y 1
```

Wracając do Sigma Studio, po wykonaniu kompilacji i ładowania projektu, opcją Link Compile Download (klawisz F7) aktywowana jest ikona Export System Files, dzięki której możemy wyeksportować pliki konfiguracyjne niezbędne do stworzenia aplikacji sterującej DSP. Pliki zapisane są w formacie tekstowym i zawierają nagłówki zgodne z C/C++ określające parametry projektu i używane rejestry DSP. Warto zapoznać się z zawartością wygenerowanych plików, jest to niezbędne do utworzenia aplikacji sterującej DSP.

Za pomocą Sigma Studio możemy też wygenerować zawartość EEPROM, którą można wykorzystać do samodzielnego programowania, czy to skryptem czy zewnętrznym programatorem (*Upload E2Prom To File* w zakładce *IC 2–WinE2PromLoader*). Podczas eksportu zostanie wygenerowanych kilka typów plików:

- **.params* – zawiera wszystkie nazwy parametrów aplikacji DSP, adresy i wartości do zapisania w pamięci;
- **.hex* – zawiera wartości bajtów pamięci parametrów w formacie szesnastkowym, dane te są również odzwierciedlone w polu Parameter Data pliku **.params;*
- **.h* – które definiują nazwy, adresy i wartości wszystkich parametrów aplikacji DSP;
- **_REG.h* – plik nagłówkowy zawiera definicje wszystkich rejestrów sprzętowych wraz z wartościami początkowymi.

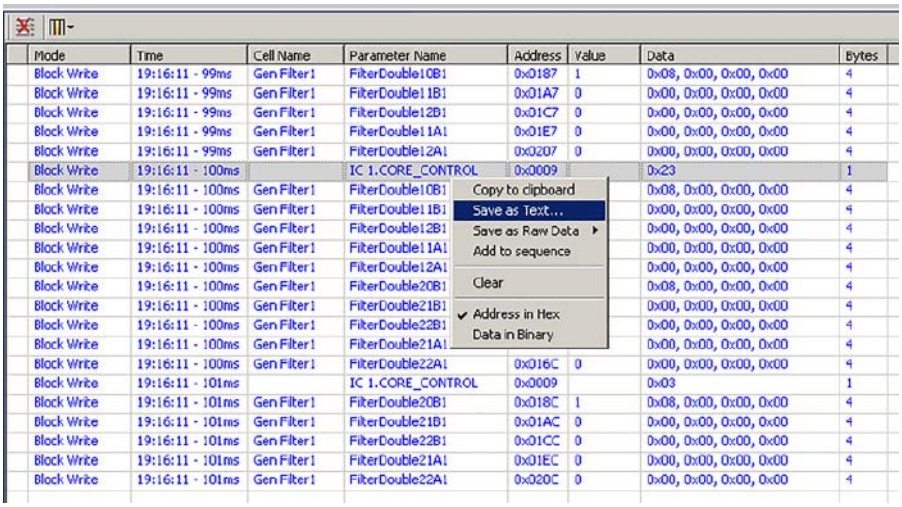
Oczywiście po każdej zmianie algorytmu konieczne jest ponowne generowanie kompletu plików konfiguracyjnych. Aby przyspieszyć tworzenie aplikacji, gdy konfiguracja DSP jest już określona i sprawdzona,

a zależy nam tylko na szybkim wyznaczeniu zmienianych parametrów bloków funkcjonalnych, należy użyć okienka Capture (**rysunek 10)**. Każdorazowo po zmianie parametru Sigma Studio zapisuje rejestry DSP odpowiednio przeliczonymi wartościami. Dla przykładu, fragment parametrów zmiany poziomu tonów wysokich bloku Gen Filter1, która zapisana jest w pliku **.params*, wygląda następująco:

```
Cell Name          = Gen Filter1
Parameter Name     =
FilterDouble10B1
Parameter Address = 391
Parameter Value    =
-1.20945749431849
Parameter Data :
0xF6 ,  0x53 ,  0x07 ,  0xF3 ,
```

Dla przyspieszenia pracy zmienione parametry mogą być skopiowane do schowka lub zapisane po kliknięciu prawym klawiszem w obszarze okienka Capture. Po przygotowaniu zestawu nastaw dla wszystkich bloków można przygotować oprogramowanie sterujące DSP.

Testowe sterowanie procesorem można przeprowadzić, korzystając z narzędzi i2c-tools. Adresy rejestrów sprzętowych i realizowane funkcje są ustalone i nie podlegają zmianie niezależnie od aplikacji DSP. Dla próby zmienimy poziom sygnału DAC, w tym celu w pliku *ADAU1772_DSPDAC_IC_1_REG.h*, w którym znajdują się definicje



Rysunek 10. Zmiana parametrów

rejestrów DSP, szukamy sekcji DAC0_VOLUME, DAC1_VOLUME odpowiedzialnej za poziom sygnału wyjściowego DAC (pełną mapę ponad 70 rejestrów DSP możemy też oczywiście sprawdzić w dokumentacji ADAU1772):

```
/* DAC0_VOLUME  - Registers (IC
1) */
#define REG_DAC0_VOLUME_IC_1_
ADDR                  0x2F
#define REG_DAC0_VOLUME_IC_1_
BYTE                  1
#define REG_DAC0_VOLUME_IC_1_
VALUE                 0x2C

/* DAC1_VOLUME  - Registers (IC
1) */
#define REG_DAC1_VOLUME_IC_1_
ADDR                  0x30
#define REG_DAC1_VOLUME_IC_1_
BYTE                  1
#define REG_DAC1_VOLUME_IC_1_
VALUE                 0x2C
```

Jak widać, rejestry odpowiedzialne za poziom wyjściowy DAC znajdują się pod adresem 0x2F/0x30, a w rzeczywistości pod adresem 16-bitowym, 0x002F/0x0030, a poziom sygnału określony jest zapisanym

Listing 1. Zmiana poziomu sygnału dla bloku Single 1 na poziom 0 dB wymaga pokazanych sekwencji zapisów blokowych (skrypt *vol0dB.sh*)

```
#!/bin/bash
echo 'ADAU1772 vol 0dB'
i2cset -y 1 0x3F 0x01 0x84 0x08 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0xA4 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0xC4 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0xE4 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x02 0x04 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x00 0x09 0x23 i
i2cset -y 1 0x3F 0x00 0xE4 0x08 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x04 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x24 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x44 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x64 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x00 0xE9 0x08 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x09 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x29 0x00 0x00 0x00 0x00 i
```

Listing 2. Zmiana poziomu sygnału dla bloku Single 1 na poziom -20 dB wymaga pokazanych sekwencji zapisów blokowych (*skrypt vol20dB.sh*)

```
#!/bin/bash
echo 'ADAu1772 vol -20dB'
i2cset -y 1 0x3F 0x01 0x84 0x02 0x87 0xA2 0x6C i
i2cset -y 1 0x3F 0x01 0xA4 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0xC4 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0xE4 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x02 0x04 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x00 0x09 0x23 i
i2cset -y 1 0x3F 0x00 0xE4 0x02 0x87 0xA2 0x6C i
i2cset -y 1 0x3F 0x01 0x04 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x24 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x44 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x64 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x00 0xE9 0x02 0x87 0xA2 0x6C i
i2cset -y 1 0x3F 0x01 0x09 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x29 0x00 0x00 0x00 0x00 i
```

Listing 3. Zmiana ustawień bloku Gen Filter 1 pomijająca jego działanie – bypass (skrypt *toneoff.sh*)

```
#!/bin/bash
echo 'ADAU1772 tone reg off'
i2cset -y 1 0x3F 0x01 0x87 0x08 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0xA7 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0xC7 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0xE7 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x02 0x07 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x00 0x09 0x23 i
i2cset -y 1 0x3F 0x00 0xE7 0x08 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x07 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x27 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x47 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x67 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x00 0xEC 0x08 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x0C 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x2C 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x4C 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0x6C 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x00 0x09 0x03 i
i2cset -y 1 0x3F 0x01 0x8C 0x08 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0xAC 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0xCC 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x01 0xEC 0x00 0x00 0x00 0x00 i
i2cset -y 1 0x3F 0x02 0x0C 0x00 0x00 0x00 0x00 i
```

Listing 4. Przykładowe podbicie tonów niskich i wysokich (skrypt *toneoff.sh*)

```
#!/bin/bash
echo 'ADAU1772 tone reg set'
i2cset -y 1 0x3F 0x01 0x87 0xF1 0xC5 0x61 0xA8 i
i2cset -y 1 0x3F 0x01 0xA7 0x17 0x23 0x0C 0xC6 i
i2cset -y 1 0x3F 0x01 0xC7 0xF7 0x92 0x74 0xA1 i
i2cset -y 1 0x3F 0x01 0xE7 0x09 0x02 0x1A 0x0D i
i2cset -y 1 0x3F 0x02 0x07 0xFE 0xC7 0x01 0xB5 i
i2cset -y 1 0x3F 0x00 0x09 0x23 i
i2cset -y 1 0x3F 0x00 0xE7 0xF1 0xC5 0x61 0xA8 i
i2cset -y 1 0x3F 0x01 0x07 0x17 0x23 0x0C 0xC6 i
i2cset -y 1 0x3F 0x01 0x27 0xF7 0x92 0x74 0xA1 i
i2cset -y 1 0x3F 0x01 0x47 0x09 0x02 0x1A 0x0D i
i2cset -y 1 0x3F 0x01 0x67 0xFE 0xC7 0x01 0xB5 i
i2cset -y 1 0x3F 0x00 0xEC 0xF1 0xC5 0x61 0xA8 i
i2cset -y 1 0x3F 0x01 0x0C 0x17 0x23 0x0C 0xC6 i
i2cset -y 1 0x3F 0x01 0x2C 0xF7 0x92 0x74 0xA1 i
i2cset -y 1 0x3F 0x01 0x4C 0x09 0x02 0x1A 0x0D i
i2cset -y 1 0x3F 0x01 0x6C 0xFE 0xC7 0x01 0xB5 i
i2cset -y 1 0x3F 0x00 0x09 0x03 i
i2cset -y 1 0x3F 0x01 0x8C 0xF1 0xC5 0x61 0xA8 i
i2cset -y 1 0x3F 0x01 0xAC 0x17 0x23 0x0C 0xC6 i
i2cset -y 1 0x3F 0x01 0xCC 0xF7 0x92 0x74 0xA1 i
i2cset -y 1 0x3F 0x01 0xEC 0x09 0x02 0x1A 0x0D i
i2cset -y 1 0x3F 0x02 0x0C 0xFE 0xC7 0x01 0xB5 i
```

do nich bajtem. Wartość poziomu zmieniana jest z krokiem 0,125 dB. Wartości 0 dB odpowiada zapis 0x00, natomiast wartości –95,625 dB odpowiada zapis 0xFF. Za pomocą poleceń:

```
i2cset -y 1 0x3F 0x00 0x2F 0x10 i
i2cset -y 1 0x3F 0x00 0x30 0x10 i
```

możemy ustawić poziom sygnału DAC. W podobny sposób zmieniamy zawartość pozostałych rejestrów sprzętowych DSP, np. sprawdzamy poziom sygnału w rejestrze DBREG0:

```
i2cset -y 1 0x3F 0x00 0x0C
i2cget -y 1 0x3F
```

Odczytana wartość powinna zmieniać się w zależności od poziomu sygnału, 0 dB=0xFF. Ze względu na 16-bitowy adres konieczne jest rozbicie odczytu na dwa polecenia, pierwsze ustawia „wskaźnik adresu”, drugie odczytuje wartość rejestru.

Nieco bardziej skomplikowana jest sprawa parametryzacji realizowanych algorytmów. Tutaj też z pomocą przychodzi funkcja Capture. Przed zmianą parametru najlepiej ustawić wartość domyślną, wyczyścić zawartość okienka Capture oraz zmienić pożądany parametr, obserwując log aktywności komunikacji z DSP. Następnie, czy to poprzez schowek, czy zapis do pliku, należy wygenerować zestaw modyfikowanych rejestrów i ich zawartości oraz na ich podstawie przygotować oprogramowanie. Przykładowo, zmiana poziomu sygnału dla bloku Single 1 na poziom 0 dB wymaga sekwencji zapisów blokowych pokazanych na **listingu 1** (skrypt *vol0dB.sh*), natomiast zmiana na –20 dB wymaga sekwencji z **listingu 2** (*skrypt vol20dB.sh*). Zmiana ustawień bloku Gen Filter 1, pomijająca jego działanie – bypass, jest możliwa po wysłaniu sekwencji z **listingu 3** (skrypt *toneoff.sh*), a przykładowe podbicie tonów niskich i wysokich pokazuje **listing 4** (skrypt *toneoff.sh*). Jeżeli skrypty działają, możemy uznać układ za przetestowany i przejść do własnych aplikacji.

Podsumowanie

Na zakończenie warto wspomnieć o układzie kodeka ADAU1372, który jest uproszczoną wersją ADAU1772 pozbawioną rdzenia DSP, która jest w 100% zgodna sprzętowo i programowo. Liczba rejestrów konfiguracyjnych jest pomniejszona o rejestry rdzenia DSP, pozostałe zachowują adresacje i funkcje. W przypadku zastosowania ADAU1372 należy jedynie wyprowadzenie SB połączyć stale z masą układu. Zbędne są też układy U2 (24LC32), U5 (LVC06), U6 (LTC4301), elementy R12, R13, C19, a w miejsce U6 należy wlutować zwory J1, J2 łączące magistralę I²C z Raspberry Pi.

Dla osób niemających styczności z Sigma DSP polecam opublikowany na łamach EP (EP 4/19...EP 11/19) kurs Audio DSP, gdzie znajdują się podstawowe informacje o środowisku Sigma Studio i procesorach Sigma DSP pominięte ze względu na ograniczoną objętość artykułu. Powodzenia w aplikacjach świata cyfrowego audio.

Adam Tatuś, EP

Masz może pomysł na ciekawy artykuł lub projekt? Skonstruowałeś urządzenie, które jest godne zaprezentowania szerszej publiczności? Możesz napisać artykuł edukacyjny? Chcesz podzielić się doświadczeniem? W takim razie zapraszamy do współpracy na łamach Elektroniki dla Wszystkich. Kontakt: edw@elportal.pl