Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Programowanie kładów FPGA

(projekt)

Implementacja algorytmów cyfrowego przetwarzania sygnałów audio na bazie układu FPGA z interfejsem UART

Pierczyk Krzysztof

Warszawa, 7 kwietnia 2021

Spis treści

1 Wstęp

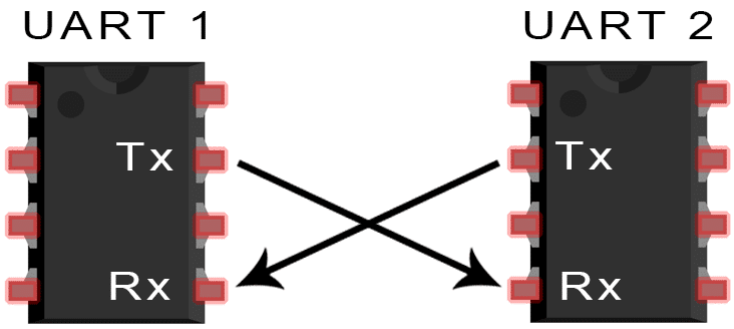
Celem projektu jest opracowanie i zaimplementowanie zestawu wybranych metod przetwarzania sygnałów audio znanych z popularnych multiefektów gitarowych. Zadaniem tego typu rozwiązań jest modyfikowanie próbkowanego dźwięku w czasie rzeczywistym w taki sposób, aby urozmaicić jego brzmienie np. poprzez modulację, przesunięcie fazowe lub wprowadzenie dodatkowych składowych. Przykładami takich efektów są m.in.

- **echo** (opóźnienie, ang. *delay*) - do sygnału dodawana jest jedna lub kilka jego kopii opóźnionych o określoną liczbę próbek; efekt ma symulować warunki panujące w halach widowiskowych
- **chorus** - działa na podobnej zasadzie co echo, jednak występujące tu opóźnienia są z reguły znacznie krótsze a dodatkowe składowe podlegają niewielkiej modulacji; daje to charakterystyczny efekt brzmienia chóralnego
- **flanger** - kolejny efekt wykorzystujący opóźnione próbki sygnału; w tym przypadku wielkość opóźnienia podlega cyklicznym zmianom, co przekłada się na pulsacyjny charakter wyjściowego dźwięku
- **overdrive** - nazwa ogółu metod prowadzących do znacznego zniekształcenia sygnału bazowego; jednym z popularnych sposobów jego implementacji jest nałożenie obustronnych ograniczeń na wyjściowe wartości przepuszczanego sygnału

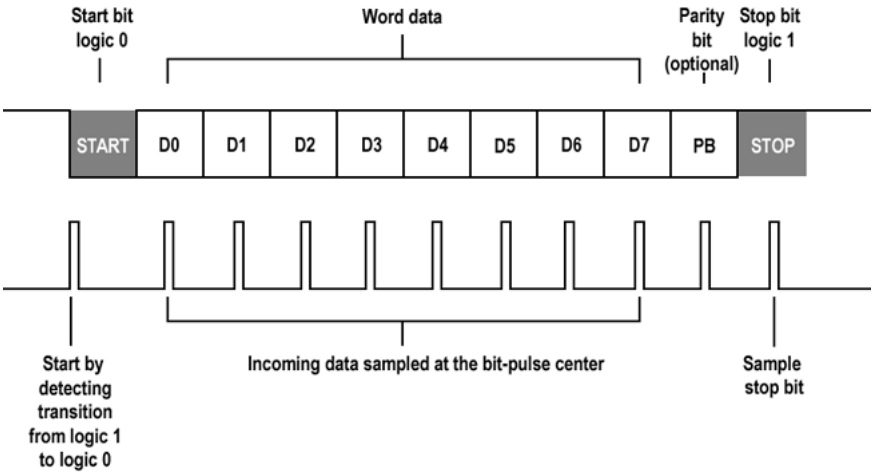
Powyższe efekty stanowią jedynie niewielki wycinek stosowanych rozwiązań, wśród których wymienić można także szeroko pojęte metody equalizacji, czy modyfikowania częstotliwości sygnału. Minimalna wersja projektu zakłada implementację scharakteryzowanych metod przetwarzania wraz z prostymi mechanizmami wprowadzania i wyprowadzania danych z urządzenia a także dostosowywania parametrów filtrów. Jako metodę komunikacji wybrano popularny (choć może w niec innych zastosowaniach) interfejs UART (ang. *universal asynchronous receiver-transmitter*). Jego prostota umożliwi przyspieszenie procesu implementacji, a co za tym idzie szybsze przejście do zasadniczej części projektu. Testowanie urządzenia obywać się będzie z pomocą prostej aplikacji w języku Python, która z pomocą wirtualnego portu szeregowego wysyłać będzie do urządzenia próbki dźwięku oraz zapisywać odebrane dane do osobnego pliku. Cyfrowy charakter UARTa pozwoli w przyszłości przejść na popularny interfejs I^2S, dzięki któremu możliwe będzie proste dołączenie do urządzenia układów przetwornikowych.

2 Analiza interfejsu komunikacyjnego

Interfejs UART jest dzisiaj dostępny w niemal wszystkich obecnych na mikrokontrolowerowych oraz w wielu układach typu SoC. Jego popularność wynika zarówno z (jak sama nazwa wskazuje) uniwersalnego charakteru jak i prostoty implementacji. UART to cyfrowe urządzenie peryferyjne umożliwiające szeregową komunikację asynchroniczną. W większości implementacji parametry komunikacji takie jak szybkość, czy format danych mogą być konfigurowane poprzez zmianę wartości odpowiednich rejestrów sterujących. Nierzadko możliwe jest też ustawienie trybu komunikacji spośród *simplex*, *duples* lub *half-duples*. Interfejsy tego typu, szczególnie w zastosowaniach przemysłowych, są często sprzęgane z konwerterami poziomów logicznych odpowiednich dla standardów RS-232 lub RS-485.



Rysunek 1: Typowa struktura komunikacji dwóch węzłów z wykorzystaniem układu UART, źródło: [1]



Rysunek 2: Struktura ramki UART, źródło: [2]

Komunikacja asynchroniczna wymaga aby wszystkie węzły nadawały i odbierały dane o z góry ustalonym formacie i długości znaku (wynikającym z szybkości transmisji). Ponadto należy wziąć pod uwagę, że zegary obecne w poszczególnych urządzeniach mogą się z czasem rozsynchronizowywać, a co za tym idzie konieczny jest mechanizm ponownej synchronizacji. W przypadku komunikacji z wykorzystaniem modułu UART mechanizm ten wynika z formatu przesyłanych danych. Typowa ramka składa się z trzech elementów: **bitu startu**, **bitów danyc** oraz **bitów stopu**. Bit startu oznacza początek nowej ramki i jest sygnalizowany stanem niskim na linii. Po nim następować może pewna liczba bitów danych - zazwyczaj 7 lub 8 - a na końcu jeden lub dwa bity stopu (sygnalizowane stanem wysokim). Bit startu odpowiada za synchronizację zegarów wykorzystywanych do próbkowania stanu linii, natomiast bity stopu definiują minimalną przerwę między kolejnymi ramkami. Fakt że każdy bit startu to ponowna okazja do zsynchronizowania zegarów sprawia, że nie muszą one pracować z dokładnie tymi samymi szybkościami. Niewielkie różnice nie powodują błędów w odbiorze danych.

Format ramki może zostać rozszerzony o element kontrolny w postaci **bitu parzystości**. Jeśli występuje, przyjmuje on wartość zależną od ilości bitów w stanie wysokim w przesyłanych danych i znajduje się przed bitami stopu. Możliwy jest bit parzystości

(ang. *even*) - ustawiony, gdy suma jest parzysta - lub nieparzystości (ang. *odd*) - ustawiany, gdy suma jest nieparzysta. Dodatkowy element pozwala wykrywać ewentualne błędy transmisji. Format ramki często oznacza się w postaci trzyznakowego identyfikatora postaci DPS, gdzie D oznacza ilość bitów danych, P - typ bitu kontrolnego (E - bit parzystości, O - bit nieparzystości, N - brak bitu kontrolnego) a S ilość bitów stopu. Typowowymi prędkościami transmisji przez UART są:

- 9600 bit/s
- 19200 bit/s
- 38400 bit/s
- ...

Jest to pewna zaszłość historyczna wynikająca z częstotliwości standardowych oscylatorów dostępnych na rynku. Moduły współcześnie implementowane w układach scalonych wzbogacone są często o dodatkowe wyprowadzenia zegarowe umożliwiające komunikację synchroniczną. Tego typu urządzenia określane są zazwyczaj mianem USART (ang. *universal synchronous asynchronous receiver-transmitter*).