Praktikum – Rechnerarchitektur
VHDL – Projekt
Spezifikation

G53: Adrian Kögl
Daniel Osipishin
Marc Sinner

Inhaltsverzeichnis

Aufgabenverteilung

Adrian Kögl	Projektleiter
Daniel Osipishin	Verantwortlicher Dokumentation
Marc Sinner	Verantwortlicher Vortrag

Abgabetermine

Spezifikation	13.05.2018
Implementierung	17.06.2018
Ausarbeitung	08.07.2018
Vortrag	23.07 - 03.08.2018

Generell: Fertigstellung bis zum Mittwoch vor Abgabe fertig haben, um bei Rückfragen Betreuer bei der Sprechstunde zu fragen

Einzelaufgaben

Marc Sinner	Zeit	Daniel Osipishin	Zeit	Adrian Kögl	Zeit
Ausarbeitung Spezifikation (eine Erst-version und eine weitere Nachbesserung)	1x5h 1x3h	Vorbereitung des Vortrags (Powerpoint, Vorzeigeversion, etc.)	10 h	Anwender-dokumentation 4h Entwickler- Dokumentation 6h Ausarbeitung der Dokumentation (eine Erst-version und zwei weitere Nachbesserungen)	1x10h 2x 2h
		Präsentation des Vortrags vorbereiten	3h		
Implementierung	5h	Implementierung	5h	Implementierung	5h
		Testing	4h		

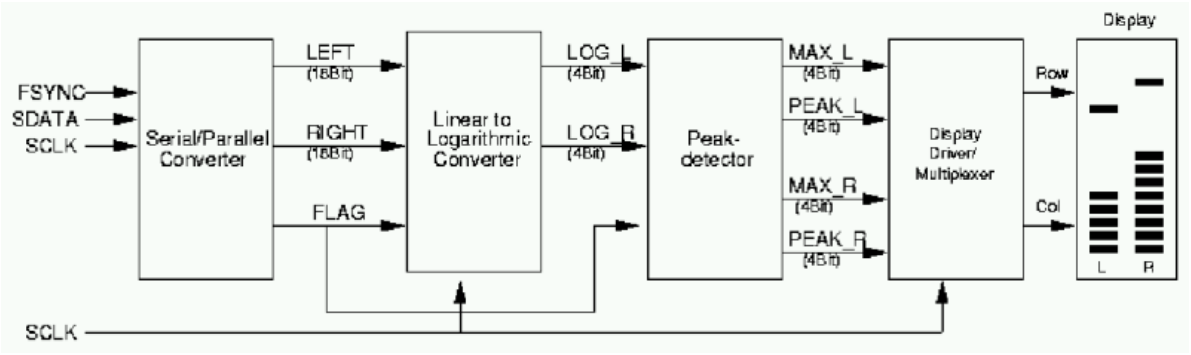
Aufgabenbeschreibung

Für eine digitale Audio-Pegelanzeige welche aus digitalen 18-Bit Audiodaten eine Pegelanzeige erstellt soll ein VHDL Programm für den Seriell/Parallel Konverter implementiert werden. Dieser Konverter ist die erste Komponente der Schaltung und bekommt als Eingang einen seriellen Datenstrom welchen es unter Berücksichtigung zweier Takte (FSYNC und SCLK) in zwei 18-Bit Werte umwandeln soll. Diese beiden Werte welche den linken und rechten Kanal repräsentieren werden dann an die nächste Komponente in der Schaltung weitergeleitet. Dafür soll zudem nach jedem 18-Bit Paar ein Flag für die nächste Komponente in der Schaltung gesetzt werden.

Ist-Analyse

Von vornherein sind die anderen Einheiten der Schaltungen vorhanden. Der Logarithmisierer, der direkt mit der von uns zu implementierenden Komponente verbunden ist, das 18-Bit Paar entgegennimmt und in die Dezibel Einheit umwandelt, sowie die weiteren Komponenten Peek Detektor, der die (temporären) Höchstwerte der digitalen Audiodaten bestimmt und der Display/Driver Multiplexer der die berechneten Werte auf einem Display projiziert.

Zudem haben wir bereits Prototyp VHDL Programme entwickelt (siehe Lösungsansätze) und diese auf dem Compiler Intel FPGA Model Sim getestet, jedoch aber noch nicht zum kompilieren bekommen. Für VHDL stehen zwar Bibliotheken zur Verfügung, jedoch werden vermutlich keine verwendet, da diese nicht notwendig sind.



Komponenten der Digital-Audio Pegelanzeigen Schaltung

Soll-Analyse

Es wird eine Entity entwickelt die als Eingänge die Signale FSYNC und SCLK als Takte, sowie den Datenstrom SDATA bekommt. Am Ausgang der Schaltung ist ein Flag, dass die nächste Komponente der Schaltung über die vorhanden Werte informiert, sowie die beiden Ausgänge LEFT und RIGHT. Dafür soll eine Architecture entwickelt werden, die auf die beiden Takte reagiert und die benötigten Ausgänge LEFT und RIGHT erstellt. Zudem sollte ein effizientes Verfahren gefunden werden, die 18-Bit Variablen (Ausgänge) zu füllen.

```
ENTITY sp_converter IS
    PORT (
        fsync, sclk, sdata : IN STD_LOGIC;
        flag : OUT STD_LOGIC;
        leftS, rightS : OUT SIGNED(17 DOWNTO 0)
    );
END sp_converter;
```

VHDL Entity

Erfolgskriterium

Es soll der serielle Datenstrom (Input) erfolgreich in zwei 18-Bit Variablen (Output) gespeichert werden mit Berücksichtigung der Takte. Der dafür verwendete Lösungsansatz sollte dies effizient realisieren.

Jedoch ist es bei diesem Projekt schwer eine objektive Aussage über die Effizienz des VHDL Programms zu erstellen, da VDHL nur eine beschreibende Programmiersprache ist (*Very High Speed Integrated Circuit Hardware Description Language*) und somit nur eine Schaltung beschreibt. Jedoch kann man bestimmen mit welchen Schaltungskomponenten die implementierten Methoden gebaut werden und daraus schlussfolgern wie günstig oder teuer diese sind und damit Aussagen über die Effizienz treffen.

Lösungsansätze

Es gibt im Prinzip nur eine Möglichkeit die Implementierung vom Prinzip zu variieren und zwar bei der Art der Speicherung der beiden 18 Bit Werte. Es werden im Folgenden zwei Varianten beschrieben. Dazu werden die bereits erstellen Prototypen für beide VHDL Architectures der Entity sp_converter (siehe Ist-Analyse) erklärt. Der komplette Code für die VHDL Prototyp Implementation ist auf SVN.

#1: Shift-Methode

Die Architecture shift_memo von der Entity sp_converter verwendet die internen Signale leftMemo und rightMemo welche vorzeichenbehafteten Zahlen mit Werten zwischen

-131072 und 131071 sind (siehe Aufgabenblatt) was mit 18 Bit realisiert werden kann bzw. den Dateninput speichert. Zudem wird ein Zähler (counter) verwendet, der bis 18 (5 Bit) hochzählt um 18 sdata bits zu betrachten und leftMemo bzw. rightMemo zu füllen.

```
ARCHITECTURE shift_memo OF sp_converter IS
    SIGNAL counter : UNSIGNED(4 DOWNTO 0);
    SIGNAL leftMemo : SIGNED(17 DOWNTO 0);
    SIGNAL rightMemo : SIGNED(17 DOWNTO 0);
```

Shift_memo architecture

Wenn sich FSYNC verändert, dann wird ab dem Zeitpunkt von LEFT auf RIGHT oder von RIGHT auf LEFT gewechselt, weshalb der Wert von counter auf null, sowie die Signale von LEFT oder RIGHT wieder auf null gesetzt werden.

```
PROCESS(fsync)
BEGIN
    counter <= "00000";

    IF (fsync = '1') THEN
        leftMemo = "000000000000000000";
    ELSE
        rightMemo = "000000000000000000";
    END IF;
END PROCESS;
```

FSYNC Prozess

Wenn der Takt SCLK auf 0 ist und noch nicht zuende gezählt wurde sollen die Daten ausgelesen werden. Hierbei tritt eine Fallunterscheidung auf ob die LEFT Variable oder die RIGHT Variable gefüllt werden muss. Dies erkennt man an dem FSYNC Takt (FSYNC == 1: Bearbeitung der LEFT Variable, und vice versa).

Hierbei werden die 18 Bit großen Standart Logic Vektoren (leftMemo, rightMemo) verwendet und der neu zu hinzufügende Bit rechts an den Vektor drangehängt, was letztendlich wie ein „Shift left“ funktioniert. Dies ist ein besonders einfaches Verfahren und füllt den Datensatz besonders schnell mit weiteren Bits.

```
PROCESS(sclk)
BEGIN
    IF (sclk = '0') THEN
        IF (counter /= 18) THEN
            IF (fsync = '1') THEN
                leftMemo <= leftMemo(16 DOWNTO 0) & sdata;
            ELSE
                rightMemo <= rightMemo(16 DOWNTO 0) & sdata;
            END IF;

            counter <= counter + 1;
```

SCLK Prozess

Bei der Flag-Setzung wird der Counter sowie der FSYNC Takt berücksichtigt. Wird momentan der RIGHT wert betrachtet (FSYNC == 0), da dieser nach LEFT kommt und ist Counter 18, bzw. wurden alle Werte für RIGHT gefüllt, so wird das Flag gesetzt (Flag <=1 sonst 0).

```
flag <= '1' WHEN (counter = 18 AND fsync = '0') ELSE '0';
```

Flag Setzung der architecture

#2: Bit-Adressierung („Slot-Methode“)

Da die Umsetzung der Prozesse in der Architecture slot_memo sich vom Prinzip her nicht groß von dem Shift-Lösungsansatz unterscheidet, werden nur die veränderten Maßnahmen beschrieben

Die Architecture verwendet statt eines 5 Bit counters einen anderen VHDL Datentyp namens NATURAL welcher (wahrscheinlich) mehr Speicher benötigt als ein 5 Bit Standart-Logic Vector counter (siehe Architecture Shift-Methode). Dieser muss jedoch verwendet werden, da der Compiler keinen Standart Logic zur Adressierung von Bits eines anderen Vektors zulässt.

```
ARCHITECTURE slot_memo OF sp_converter IS
    SIGNAL counter : NATURAL;

    SIGNAL leftMemo : SIGNED(17 DOWNTO 0);
    SIGNAL rightMemo : SIGNED(17 DOWNTO 0);
```

Slot-Memo Architecture

Bei diesem FSYNC Prozess wird counter mit 18 initialisiert und zählt dann bei jedem SCKL Takt runter um die entsprechenden Speicherzellen die Big Endian angeordnet sind, zu adressieren.

```
PROCESS(fsync)
BEGIN
    counter <= 18;

    IF (fsync = '1') THEN
        leftMemo = "000000000000000000";
    ELSE
        rightMemo = "000000000000000000";
    END IF;
END PROCESS;
```

FSYNC Prozess

Genauso wie bei dem Shift-Lösungsansatz wird hier entsprechend der FSYNC und SCLK Takte die Variablen geändert. Jedoch wird, was der Hauptunterschied zur SHIFT Methode ist, das hinzuzufügende aktuelle Bit *sdata* an die Stelle im Vektor *leftMemo* oder *rightMemo* an die Stelle die von counter-1 adressiert wird, hinzugefügt.

```
PROCESS(sclk)
BEGIN
    IF (sclk = '0') THEN
        IF (counter /= 0) THEN
            IF (fsync = '1') THEN
                leftMemo(counter - 1) <= sdata;
            ELSE
                rightMemo(counter - 1) <= sdata;
            END IF;

            counter <= counter - 1;
```

Die Flag Setzung zur Benachrichtigung der nächsten Komponente, dass die Daten bereit zur Weiterverarbeitung sind, ist identisch.

Verwendeter Lösungsansatz

Da sich beide Lösungsansätze nicht groß unterscheiden, sondern nur zwei verschiedene Möglichkeiten beschreiben die bearbeitenden Daten zu speichern, ist es schwer zwischen beiden Implementierungen zu unterscheiden. Zudem ist wie bereits im Punkt Erfolgskriterien beschrieben, VHDL nur eine beschreibende Sprache, weshalb sich die Effizienz der beiden Implementierungen nur schwer vergleichen lässt (gleiches gilt für Weiterentwicklung und Wartung).

Betrachtet man jedoch rein hypothetisch die praktische Realisierung der Komponenten in physische Bauteile dann lässt sich die Shift Funktionalität vermutlich billiger produzieren. Beim „Shifften“ hätte man aneinandergereihte D-Flip-Flops und bei dem adressieren des Vektors bräuchte man einen Multiplexer, welcher teurer bzw. ineffizienter wäre.

Zudem braucht man bei der Implementierung der Slot-Methode bei dem von uns verwendeten Compiler den NATURAL Datentyp welcher vermutlich mehr Speicher benötigt als der minimalgehaltene 5 Bit counter in der Shift-Methode.

Deshalb entscheiden wir uns für den ersten Lösungsvorschlag, die Shift-Methode. Trotzdem lassen wir die Möglichkeit bestehen, falls sich unsere Annahme, dass es effizienter ist nicht bestätigt, den anderen Lösungsvorschlag zu verwenden.