Entwicklerdokumentation VHDL

Gruppe 25

Jan Waidner

06.07.2016

Gesamtprojekt

Das Teilprojekt Logarithmierer gliedert sich in das Gesamtprojekt Pegelanzeige ein, in dem es die großen Eingangswerte auf eine sinnvolle interne Verteilung mappt.

Dies geschieht bei einer fallenden Taktflanke von SCLK und gesetztem Flag-Bit (wenn also der Serial/Parallel Converter die analogen Audiosignale eingelesen und per 18 Bit Werte an den Logarithmierer weitergegeben hat).

Der Logarithmierer leitet dann einen positiven logarithmierten Wert an den Peakdetector weiter. Dieser verarbeitet die Signale weiter, bis Daten schlussendlich bei dem Display Driver/Multiplexer ankommen, der sie auf einem externes Display anzeigt.

Eingänge

- LEFT (18 Bit, vorzeichenbehaftet)
- RIGHT (18 Bit, vorzeichenbehaftet)
- FLAG (1 Bit, Boolean)
- SCLK (1 Bit, Boolean)

Ausgänge

- LOG_L (4 Bit, vorzeichenfrei)
- LOG_R (4 Bit, vorzeichenfrei)

Dateien/Programmlogik

Alle Programmteile benötigen die Unterbibliotheken ieee.std_logc_1164.all und ieee.numeric_std.all.

Der Code ist schon gut dokumentiert, soll hier aber noch einmal genau analysiert werden.

logarithmierer.vhdl

Implementierung des Hauptprogrammes. Es gelten die Ein und Ausgänge die oben definiert sind.

Das Programm ist durch einen process mit der sensitivity list (SCLK) realisiert.

Der process fährt weiter fort, wenn sich SCLK auf der fallenden Taktflanke (ungewöhnlich, jedoch genau zu diesem Zeitpunkt stellt der Serial/Parallel Converter die Daten zur Verfügung) befindet und das FLAG Bit gesetzt ist.

Dann wird LOG_L die Ausgabe der Methode mapInputToOutput(LEFT) und dann LOG_R die Ausgabe der Methode mapInputToOutput(RIGHT) zugewießen.

Durch den Aufruf einer Methode konnten Codeduplicate für jeweils die linke und rechte Seite vermieden werden.

helpfunc.vhdl (Methode mapInputToOutput(signed(17 downto 0)))

Implementierung der Methode mapInputToOutput(signed(17 downto 0))

Als Parameter ist die Variable inputVal als signed(17 downto 0) definiert, die Rückgabe ist ein std_logic_vector. Als interne Variable definieren wir absValue mit einem Wertebereich von 0 bis 131072.

Der erste Schritt besteht darin, die Eingangsvariable auf ihren Wert zu überprüfen.

Falls der Wert über 0 liegt, können wir inputVal direkt zu einem Integer wandeln und absValue zuweisen.

Falls der Wert unter 0 liegt, wandeln wir inputVal zuerst in einen Integer und invertieren ihn, bevor das Ergebnis absValue zugewiesen wird.

Die Verwendung von einem Integer ist nötig, da nur mit einem Integer eine case Abfrage auf einem Bereich gemacht werden kann.

Die case Abfrage ist auch gleich der nächste Schritt, der hier gleich rechts ersichtlich ist.

Anhand einer switch-case Bereichsabfrage, wird der richtige Wert zurück gegeben.

```
case absValue is
    when 0 to 29246         => return "0000"; -- -30db
    when 29247 to 32322     => return "0001"; -- -28db
    when 32323 to 35721     => return "0010"; -- -26db
    when 35722 to 39478     => return "0011"; -- -24db
    when 39479 to 43630     => return "0100"; -- -22db
    when 43631 to 48219     => return "0101"; -- -20db
    when 48220 to 53290     => return "0110"; -- -18db
    when 53291 to 58894     => return "0111"; -- -16db
    when 58895 to 65088     => return "1000"; -- -14db
    when 65089 to 71934     => return "1001"; -- -12db
    when 71935 to 79499     => return "1010"; -- -10db
    when 79500 to 87860     => return "1011"; -- -08db
    when 87861 to 97101     => return "1100"; -- -06db
    when 97102 to 107313    => return "1101"; -- -04db
    when 107314 to 118599   => return "1110"; -- -02db
    when 118600 to 131072   => return "1111"; --  00db
end case;
```

logarithmiererTB.vhdl

Implementierung der Testbench für das Hauptprogramm.

Als Ausgangangssignale sind LEFT_IN und RIGHT_IN als signed(17 down to 0) und FLAG und SCLK (std_logic (1 Bit)) definiert.

Als erstes wird in der Testbench ein definierter Anfangszustand von SCLK = 1, FLAG = 0, LEFT_IN = 0 und RIGHT_IN = 0 gesetzt.

Nach einer kurzen Wartezeit beginnt der erste Teil des Tests.

In einer Schleife von -131072 bis 131071 werden jeweils der aktuelle Schleifenzähler dem linken (LEFT_IN) und dem rechten (RIGHT_IN) Kanal zugewiesen und anschließend die Ausgabe per FLAG = 1 und SCLK = 0 aktiviert.

Danach wird immer wieder SCLK = 1 und FLAG = 0 gesetzt um den Eingangszustand wiederherzustellen und den Logarithmierer auf den nächsten Test vorzubereiten.

Nach der Schleife folgt der zweite Teil des Tests.

Die Eingänge LEFT_IN und RIGHT_IN werden auf 130171 gesetzt.

Wir versuchen falsche Ausgaben des Logarithmieres zu provozieren in dem wir zuerst SCLK auf 0 und Flag auf 0 setzen und danach nur FLAG auf 1 setzen aber SCLK ohne Veränderung belassen. Bei beiden Testfällen soll keine Ausgabe erzeugt werden.

make.cmd

Die Datei make.cmd erledigt die Kompilieraufgaben für uns und wird im Abschnitt Kompiliervorgang näher behandelt.

vcdOut.vcd

Ausgabe des Testlaufs, der in gtkWave betrachtet werden kann.

Im root Verzeichnis und in der Anwenderdokumentation finden Sie auch Screenshots, die eine typische Ausgabe eines Tests abbilden.

work-obj93.cf

Beim compilieren automatisch erzeugt. Bildet den Namespace für die Ausführung des Programms ab.

ghdl.exe

Binary Datei von ghdl zum kompilieren der vhdl Dateien.

ghdl kann teilweise nicht ganz einfach in der Bedienung sein und gibt nicht immer sinnvolle Fehlermeldungen aus. Daher haben wir uns dazu entschieden, alle zum Kompilieren nötigen Dateien mit zu liefern.

Kompiliervorgang

Hier soll der Kompiliervorgang, der durch die Datei make.cmd ausgeführt wird, analysiert werden.

Grundsätzlich wird eine vollständige GHDL Installation benötigt. Um Inferenzen zu vermeiden, liefern wir mit unserer Implementierung gleich alle nötigen Dateien zum kompilieren mit.

1. Analyse

   Mit dem Befehl

   ghdl.exe -a helpfunc.vhdl logarithmierer.vhdl logarithmierertb.vhdl

   wird die Analyse und Syntaxprüfung ausgeführt.

   Bei erfolgreiche Prüfung wird die Datei work-obj93.cf erstellt, welche den Namensraum für das VHDL Programm abbildet und wichtig ist, falls Methoden verwendet werden sollen.

   Kritisch bei der Analyse ist, dass alle vhdl Dateien die zum Projekt gehören, gleichzeitig analysiert werden müssen.

   Es können beliebig viele *datei.vhdl* Dateien an die Parameterliste von ghdl.exe angehängt werden.

   Falls einzeln Analysiert wird, wird jeweils die Datei work-obj93.cf neuerstellt und die alten Einträge daraus gelöscht, was in einer Fehlermeldung resultiert, falls auf eben jene alten Teile zugegriffen werden soll.

2. Ausarbeitung (elobartion)

   Mit dem Befehl

   ghdl.exe -e logarithmiererTB

   werden die ausführbaren Dateien für GHDL erstellt.

   Zu beachten ist, dass hier der Name der auszuführenden Entity, nicht der Dateienname mitgegeben werden muss.

3. Ausführung

   Mit dem Befehl

   ghdl.exe -r logarithmiererTB --vcd=vcdOut.vcd

   wird die Entity mit dem zutreffenden Name ausgeführt und die Ausgabe in eine vcd Datei abgelegt.

   Die Ausführung kann bei großen Tests durchaus etwas länger dauern (in unserem Fall mit ca 260 000 Testfällen ca. 20 Sekunden) und auch größere Ausgaben produzieren (in unserem Fall ca. 40 MB).

Erweiterung

Veränderung der Anzahl der Eingangs/Ausgangspaare

Durch den modularen Aufbau ist das hinzufügen eines Eingang/Ausgang-Paares kein Problem und schnell erledigt.

Im process in der Datei logarithmierer.vhdl wird einfach ein weiteres Mal die Methode mapInputToOutput mit dem entsprechenden Parameter aufgerufen und der Rückgabewert an einen Ausgang weiter geleitet.

Einen Eintrag zu entfernen ist ebenfalls schnell passiert.

Variablen und Portdefinitionen müssen natürlich ebenfalls entsprechend geändert werden.

Veränderung des Wertebereichs des Eingangs

Solch eine Änderung erfordert die Neuberechnung der look-up(switch-case) Tabelle in der Datei helpfunc.vhdl.

Die aktuelle Tabelle wurde mit Hilfe der Webseite wolframalpha.com und der gegebenen Formel 20 log (x/131072) = *db* errechnet, wobei bei *db*, der gewollte dezibel Wert, eingetragen wird (neben der slook-up Tabelle stehen jeweils die entsprechenden Werte).

Die Formel wird dann nach x aufgelöst und approximiert. So erhält man die Wertebereiche der look-up Tabelle.

Variablen und Portdefinitionen müssen natürlich ebenfalls entsprechend geändert werden.

Veränderung des Wertebereichs des Ausgangs

Hier müssen nur die entsprechenden Rückgabewerte der look-up Tabelle in der Datei helpfunc.vhdl verändert werden.

Variablen und Portdefinitionen müssen natürlich ebenfalls entsprechend geändert werden.

Änderung an dem Algorithmus zum Mapping

Durch die Modularität, muss hier nur die Methode mapInputToOutput in der Datei helpfunc.vhdl verändert werden.

Das Hauptprogramm aus der Datei logarithmierer.vhdl kann unverändert bleiben.