FH-OÖ Hagenberg/ESD
Hardware / Software Codesign, SS 2020
Florian Eibensteiner © 2020 (R 4864)

1. Übung: Modelling with SystemC – The Cordic-Algorithm

Name(n):

1 SIN und COS in Hardware

Um die Performance eines Systems zu steigern und die Software zu entlasten, soll die Berechnung von Winkelfunktionen (sin und cos) in Hardware ausgelagert werden, indem der Cordic-Algorithmus (*Coordinate Rotation Digital Computer*) implementiert wird. Der Algorithmus soll in der Zielplattform als Peripherie eines Prozessors in einem SoC eingesetzt werden. Um möglichst schnell einen ersten Entwurf zu erhalten, soll zuerst der Cordic-Algorithmus in SystemC modelliert werden und in einem weiteren Schritt das Systemmodell um eine CPU erweitert werden.

1.1 Cordic-Algorithmus

Der Cordic-Algorithmus ist ein sehr effizienter Weg zur Berechnung von beispielsweise trigonometrischen, logarithmischen oder exponentiellen Funktionen. Der Algorithmus könnte auch für Multiplikation und Division verwendet werden, wobei er jedoch in diesem Fall Standard-Multiplizier- und -Dividiereinheiten unterlegen ist [Ris04].

Implementieren Sie ein SystemC-Modul, welches basierend auf dem Cordic-Algorithmus den Sinus bzw. Kosinus für einen gegebenen Winkel berechnet. Gehen Sie dabei schrittweise vor, indem Sie zuerst ein allgemeines Modell erstellen, welches mithilfe der Funktionen *sin()* und *cos()* aus der *C++*-Bibliothek die gewünschten Werte berechnet.

Verfeinern Sie das Modell im nächsten Schritt, indem Sie den Cordic-Algorithmus, wie in der Übung vorgestellt (Circular Rotating-Mode), auf abstrakter Ebene umsetzen. Sie können Ihre Implementierung an die Umsetzung in MATLAB (siehe *cordic.m*) anlehnen. Für die Berechnung der nächsten Schrittweite können Sie auf die Funktion *atan()* zurückgreifen. Da es sich um ein abstraktes Modell ohne Timing handelt, soll **kein** Takt- oder Resetsignal eingebaut werden.

Wählen Sie die Anzahl der Iterationen mit 16. Welche Genauigkeit kann erreicht werden? Ist eine
□ Erhöhung der Iterationen unter Berücksichtigung unten stehender Schnittstelle sinnvoll?

Der Cordic Algorithmus konvergiert nur bei Winkel zwischen ca. $90°$ und $-90°$ ($\varphi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$) [Ris04]. Daher müssen alle Winkel die nicht in diesen Quadranten liegen, entsprechend umgerechnet werden. Die Umrechnung soll nicht in Hardware erfolgen sondern in Software durch den Treiber gemacht werden.

Um die Implementierung weiter zu vereinfachen und um bei gleichbleibender Bitbreite die Ge-

nauigkeit zu erhöhen, muss der Cordic-Core nur Winkel zwischen $0°$ und $90°$ berechnen können. Dadurch müssen nur noch positive Winkel berechnet werden und die Datenschnittstelle kann wie folgt spezifiziert werden:

- iPhi: 1.21 Darstellung (WL = 22, IWL = 1)
- oX: 0.16 Darstellung (WL = 16, IWL = 0)
- oY: 0.16 Darstellung (WL = 16, IWL = 0)

1.1.1 Schnittstelle

Damit im nächsten Schritt bereits mit der Entwicklung der Firmware, diese läuft im CPU-Modul, begonnen werden kann, soll die Schnittstelle des Cordic-Cores register- und bitgenau implementiert werden. Die Schnittstelle der Cordic-IP soll wie folgt implementiert werden:

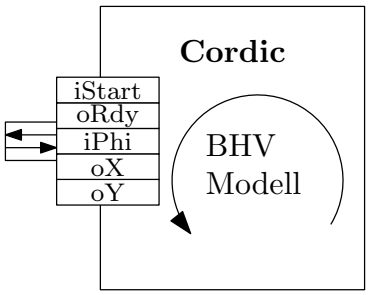
Name	Typ	Beschreibung
iStart	boolean	Start-Flag startet die Berechnung bei einer steigenden Flanke
oRdy	boolean	Ready-Flag signalisiert das Ende der Berechnung mit einer steigenden Flanke
iPhi	sc_ufixed	Eingabewinkel
oX	sc_ufixed	Ausgabewinkel (cos)
oY	sc_ufixed	Ausgabewinkel (sin)

Note: Vergessen Sie nicht zum Einbinden der Festkommatypen aus SystemC das entsprechende *Define* zu setzen (beachte mehrfach ein von `systemc.h`).

```
1 ...
2 #define SC_INCLUDE_FX // -> or by compiler switch
3 #include <systemc.h>
4 ...
```

Die Kapselung der Schnittstelle in ein entsprechendes Registerfile, welches über einen 32-bit Bus von der Firmware aus gelesen und geschrieben werden kann, folgt später.

Die Berechnung wird gestartet, sobald das *iStart*-Flag gesetzt wird. Nach Beendigung der Berechnung wird an den Ausgängen *oX* und *oY* das Ergebnis angelegt und das *oRdy*-Flag gesetzt. Diese Leitung könnte später in Hardware beispielsweise als Interruptleitung genutzt werden.



Die richtige Funktionsweise der Flags soll in einem Trace-File nachgewiesen werden. Da ohne Timing gearbeitet wird müssen für die Erzeugung und Erkennung der Flanken Deltazyklen stattfinden. Damit diese im Trace-File ausgewertet werden können, muss folgender Parameter gesetzt werden:

```
...
sc_trace_file *my_trace_file;
my_trace_file = sc_create_vcd_trace_file("cordic_trace");
my_trace_file->delta_cycles(true);
...
```

□ Die richtige Funktionsweise Ihrer Implementierung soll ausgiebig und sinnvoll mit entsprechenden Testfällen getestet werden. Die Testfälle und das generierte Trace-File sind Teil der Abgabe und müssen bei einer Übungspräsentation gezeigt werden.

Literatur

[Ris04] Thomas Risse. CORDIC-Algorithmen verbinden Mathematik, Computer-Architektur und Anwendungen. *Global J. of Engn. Educ.*, 8, No.3, 2004.