Dokumentation zum Semesterprojekt Programmierbare Logik: Realisierung des Videospiels "Pong"

Marc Ludwig, Matthias Springstein

30. Juli 2012

Inhaltsverzeichnis

1 Aufgabenstellung

Ziel des Semesterprojektes ist es, auf dem Entwicklungsboard DE2-70 von Terasic [1] eine Umsetzung des 1972 ertsmals von Atari veröffentlichten Spiels Pong zu implementieren.

Das 1972 von Atari veröffentlichte Pong wurde zum ersten weltweit populären Videospiel und in den 1970er-Jahren zunächst auf Geräten in Spielhallen bekannt. Es gilt als Urvater der Videospiele, obgleich schon zuvor Videospiele entwickelt worden waren.[2]

Das Spielprinzip von Pong ist simpel und ähnelt dem des Tischtennis: Ein Punkt (Ball) bewegt sich auf dem Bildschirm hin und her. Jeder der beiden Spieler steuert einen senkrechten Strich (Schläger), den er mit einem Drehknopf (Paddle) nach oben und unten verschieben kann. Lässt man den Ball am Schläger vorbei, erhält der Gegner einen Punkt.

1.1 Anforderungen

- durchgeführte Dokumentation des Projektes (Quellcode und Aufgabenbearbeitung)
  - Quellcode durch Doxygen [3] [4]
  - Aufgabenbearbeitung durch ein PDF Dokument [5] [6]
- funktionale Bedienelemente für beide Spieler
- graphische Darstellung auf einem via VGA angeschloßenen CRT/TFT Monitor
- Quellcode, basierend auf einem durch mehrere Mitarbeitende Personen realisierbarem Konzept, welcher kommentiert und dokumentiert ist

2 Dokumentation

Die Dokumentation des Semesterprojektes wird mit dem bereits erwähnten Textsatzsystem LaTeX durchgeführt und als generierte PDF Datei zur Verfügung gestellt. Eingebundene Abbildungen und Diagramme sind ebenfalls im Dokumentationsverzeichnis enthalten. Die mittels Doxygen erstellte Quellcode Dokumentation liegt als HTML-Dokument [7] vor.

3 Entwicklunghierarchie

Als Entwicklungsansatz wurde zunächst das Top-Down Modell gewählt, unter dem Aspekt, mit einem spätere Bottom-Up Modell den gewählten Ansatz zu verifizieren. Als zu bevorzugendes Entwurfsmuster für VHDL Designs ist das Register Transfer Level (im folgenden RTL) gewählt worden.

Charakteristisch für eine RTL-Beschreibung ist die Trennung von Registern und kombinatorischen Logikstufen, die jeweils das Eingangssignal der nachfolgenden Registerstufe definieren. Vgl. Abb. 1 Der Datenpfad wird somit als Pipeline aufgefasst. Entsprechend sollen beim reinen RTL-Entwurfsstil kombinatorische und getaktete Prozeße streng voneinander getrennt werden.

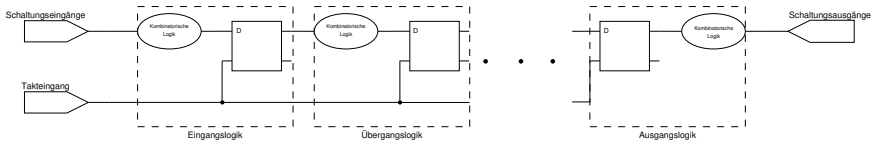

Abbildung 1: Register Transfer Level

[1] Altera Produktbeschreibung
[2] Artikel in der freien Wikipedia Enziklopädie
[3] Projektseite
[4] Artikel in der freien Wikipedia Enziklopädie
[5] Projektseite
[6] Artikel in der freien Wikipedia Enziklopädie
[7] LINK ZUR DATEI EINFÜGEN

4 Entwicklungsstrategie

Bezugnehmend auf dem Daten-Steuerpfad Modell [1], haben wir uns für folgende Methoden zur Systempartitionierung entschieden.

- Hierarchische Strukturierung des Entwurfs
- Lokalität der Module und Signale
- Reguläre Strukturen

4.1 Hierarchische Strukturierung des Entwurfs

Unter Hierarchischer Strukturierung versteht man die Partitionierung einer komplexen Entwurfsaufgabe in Teilaufgaben sowie deren weitere Gliederung in noch kleinere Einheiten, solange bis die Teilprobleme mit einfachen Lösungsansätzen zu beschreiben sind. Mit Hilfe von Komponenten (*component*) läßt sich eine hierarchische Problemlösung erreichen. Ergebnis der Synthese ist letzlich eine hierarchische Netzliste.

4.2 Lokalität der Module und Signale

Dies ist eine Forderung, daß die einzelnen Module zur Lösung der Teilprobleme keine Seiteneffekte auf benachtbarte Schaltungsteile aufweisen. Durch Blöcke (*block*) lassen sich innerhalb einer Architektur nebenläufige Anweisungen zu Funktionsblöcken zusammenfassen. Innerhalb von Blöcken können lokale Signale definiert werden, die nur innerhalb des Blocks gültig sind.

4.3 Reguläre Strukturen

Dies bedeutet die Verwendung möglichst einheitlich aufgebauter Bibliothekskomponenten. Derartig reguläre Entwürfe sind einfacher zu beschreiben und zu verifizieren. Prozeduren (*procedure*) und Funktionen (*function*), die insbesondere auch in übergeordneten Bibliotheken (*library*) oder Design-spezifischen Bibliotheken (*package*) abgelegt werden können, dienen dazu, standardisierte Lösungen für häufig auftretende Probleme zu verwenden.

Zunächst wird die Problemstellung in einzelne Funktionsblöcke gegliedert. Sollten diese noch nicht überschaubar genug sein, so sind diese weiter aufzuteilen Anschließend werden alle zwischen den Funktionsblöcken erforderlichen Schnittstellensignale mit Signalflußrichtung und Datentyp festgelegt. Jeder Funktionsblock wird als Komponente betrachtet.

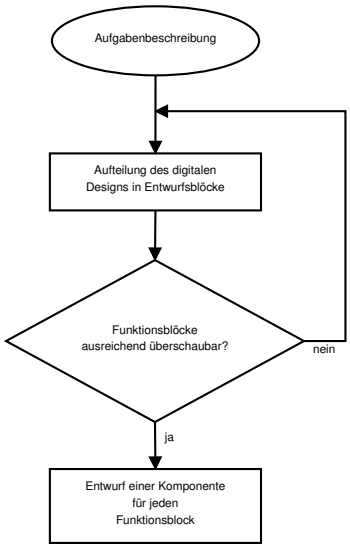

Abbildung 2: Ablaufplan zur Erstellung der Partitionierung eines digitalen Systems

5 Struktureller Entwurf mit Komponenten

In diesem Abschnitt erläutern wir nun, wie der strukturelle Entwurf digitaler Systeme auf Basis vom vorangegangen eingeführten hierarchisch aufgebauten *entity/architecture*-Paaren realisiert wird. In der übergeordneten Ebene wird dazu ein Strukturmodell formuliert, in dem diese Paare als Komponenten zu einer größeren Schaltung zusammengefaßt werden. Sie enthalten entweder die Verhaltensbeschreibungen digitaler Funktionen, oder sie bestehen wiederum aus Strukturbeschreibungen einer hierarchisch niedrigeren Stufe. [1]

Mit dem VHDL-Strukturbeschreibungsstil werden in einer Top-Entity die Signalkopplungen von mehreren Komponenten in einer Architektur erzeugt. Jede Komponente bezieht sich somit auf eine untergeordnete *entity* mit mindestens einer dazugehörigen *architecture*. In der Top-Entity sind deshalb nur die Schnittstellen der einzelnen Komponenten untereinander und zu Systemgrenzen Sichtbar und die Top-Entity stellt somit eine textuelle Formulierung eines Blockschaltbildes dar. Welches von den meisten Synthesewerkzeugen über eine Schaltplanausgabe automatisch erstellt werden kann. [2] [3] Mit diesem Ansatz wird dann ein Top-Down Entwurf verfolgt.

In der nun Vorliegenden strukturellen Architektur der Top-Entity werden die zu nutzenden Komponenten zuerst mit ihren Schnittstellen deklariert und im Architekturrumpf instanziiert. In dieser Instanz werden die Komponentenanschlüsse durch interne Koppelsignale bzw. durch Signale aus der Schnittstellenliste der Top-Entity verdrahtet. Zum leichteren Verständnis kann man sich Analog einen Hardwarebezug mit dem „Board-Socket-Chip-Modell“[4] vorstellen. Hier repräsentiert die Top-Entity eine zu modelierende Platine (Board). Die Komponentendeklaration macht den IC-Typ mit seinen Anschlüßen bekannt. Wobei eine Komponenteninstanziierung dem Einlöten eines IC-Sockels (Socket) entspricht. Mit der Konfiguration werden im letzten Schritt die IC-Sockel mit ICs bestückt, welche durch ihre speziellen *entity/architecture*-Kombinationen die Funktionalität der Platine bestimmen. [1]

6 Checkliste zum systematischen Systementwurf

Basierend auf dem Daten-Steuerpfad-Modell lässt sich Folgende Checkliste aufstellen.

1. Identifizieren der Eingangs- und Ausgangssignale des Systems auf einem Blatt Papier (Eingänge von links, Ausgänge nach rechts)
2. Analyse der Timing-Randbedingungen: In welcher zeitlichen Reihenfolge müssen Eingangs- und Ausgangssignale eintreffen bzw. generiert werden, um die einzelnen Funktionen des Systems zu realisieren?
3. Systempartitionierung des Datenpfads: Welche Komponenten sind zur Lösung von Teilaufgaben geeignet?
4. Analyse der Datenpfadkomponenten
   - 4.1 Blockdiagramm für den Datenpfad: Eine Liste mit erforderlichen lokalen Datenpfadsignalen.
   - 4.2 Welche Funktionsblöcke sind taksynchron und welche kombinatorisch?
   - 4.3 Welche synchronen Komponenten benötigen einen synchronen/asynchronen Reset/Preset?
   - 4.4 Analyse aller Datenpfadkomponenten: Welche Steuersignale sind für die verschiedenen Teilfunktionen erforderlich? Welche Statussignale werden erzeugt? Eintragung dieser in die Signalliste und ergänzen des Blockdiagramms.
   - 4.5 Ergänzung der Datenpfadkomponenten im Blockdiagramm durch eine entsprechende Abhängigkeitsnotation.
5. Ergänzen des Blockdiagramms durch einen Zustandsautomaten für den Steuerpfad. Zunächst reicht ein Funktionsblock, welcher später im VDHL-Code durch zwei Prozeße modelliert wird.

5.1 Welche externen und internen (lokalen) Datensignale steuern den Zustandsautomaten? Eintragen dieser in das Blockdiagramm und ergänzen der Signalliste.

5.2 Sicherstellen das der Automat alle intern erforderlichen Steuersignale für den Datenpfad, sowie alle externen Steuersignale erzeugt.

5.3 Festlegen der Automatenzustände (schriftlich): Wieviele Zustände? Welche Namen? Welche Funktion?

5.4 Analysieren, welche Steuersignale Mealy- bzw. Moore-Charakter haben sollen.

5.5 Zeichnen eines Zustandsdiagramms auf Basis der Datenpfadarchitektur und des zuvor analysierten Timings für die verschiedenen Systemfunktionen. Sind abstrakte Bedingungen erforderlich, so ist dies ein Zeichen dafür, dass die Strukturierung des Datenpfads noch nicht detailiert genug ist.

5.6 Prüfen der Vollständigkeit des Zustandsdiagramms:

- Welches ist der Reset Zustand?
- Gibt es Zustände aus denen der Automat nicht mehr heraus kommt?
- Werden alle Kobinationen der Eingangssignale im Zustandsdiagramm berücksichtigt?
- Gibt es in allen Zuständen für alle Ausgangssignale eine Wertzuweisung? Welche Signalwerte eignen sich ggf. als Dafault im VHDL-Code?

6. Schreiben einer VHDL-entity unter Berücksichtigung aller im Blockdiagramm definierten Schnittstellensignale.

7. Schreiben der architecture wie folg:

7.1 Deklaration aller lokalen Signale der zuvor erstellten Signalliste. Überlegungen, welcher Signaltyp verwendet werden soll.

7.2 Deklaration jeweils eines Prozeßes und Entnahme aus dem Blockdiagramm, welche Datenpfadsignale erzeugt werden sollen.

7.3 Erstellen zweier Prozeße für den Zustandsautomaten: hiervon muss ein Prozeß getaktet sein und einen in der Regel asynchronen Reset besitzen. Der andere Prozeß ist Kombinatorisch und erzeugt das Folgezustandsignal sowie in der Regel alle steuersignale.

7.4 Ergänzen aller Prozeße durch ihre Empfindlichkeitslisten:

* Bei kombinatorischen Prozeßen: Alle Eingangs- und Entscheidungssignale
* Bei taktsynchronen Prozeßen: Nur Systemtakt, sowie ggf. vorhandenen Reset

7.5 Ergänzen aller sequentieller Anweisungen, die die individuelle Funktion des Prozeßes beschreiben.

7 Umsetzung der Checkpunkte

`Identifizieren der Eingangs- und Ausgangssignale des Systems auf einem Blatt Papier (Eingänge von links, Ausgänge nach rechts)`

Aufgrund dieser Forderung ließ sich Folgende Graphik erstellen:

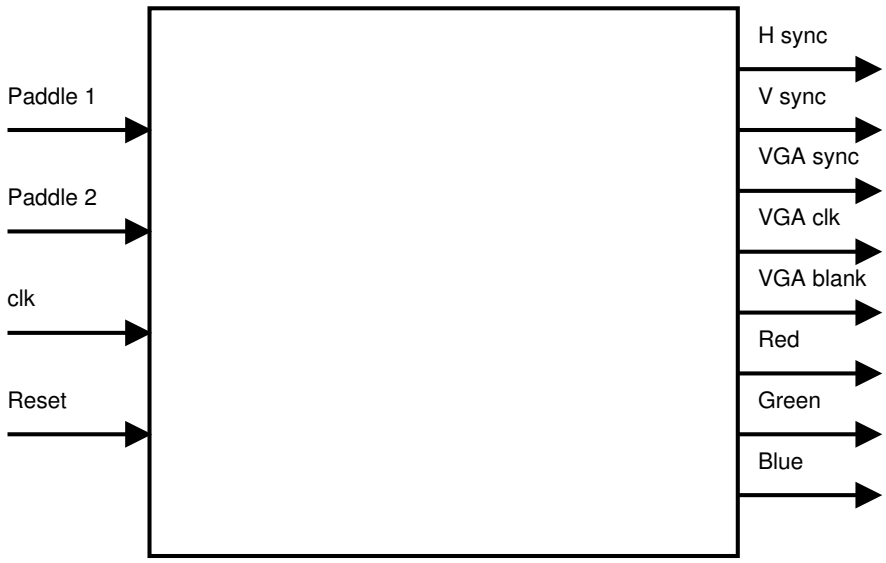

Abbildung 3: Eingangs- und Ausgangssignale des Systems

Literatur

[1] Jürgen Reichardt and Bernd Schwarz. *VHDL-Synthese - Entwurf digitaler Schaltungen und Systeme*. Oldenbourg Verlag, München, aktualisierte aufl. edition, 2009.

[2] *Introduction to the Quartus® II Software*.

[3] *ISE 10.1 Quick Start Tutorial*.

[4] Perry Perry. *Vhdl - Programming by Example*. McGraw-Hill, New York, 4. a. edition, 2002.

Abbildungsverzeichnis

Index