Beginn: 13.00 Uhr
Ort: Arbeitsgruppenraum C2.227
Anwesende: Alle Projektgruppenteilnehmer außer Franz Rammig

Termine für weitere Treffen in der vorlesungsfreien Zeit:

- am 23.02.1995 um 13.00 Uhr im C2.227
- am 16.03.1995 um 13.00 Uhr im C2.227
- am 30.03.1995 um 13.00 Uhr im C2.227

Gruppe Rechenkern:

Problem: In Speedchart kann man keine Floats spezifizieren und nach Synopsys exportieren. Es stellte sich heraus, daß Synopsys keine Floats unterstützt.
Erster Lösungsansatz ist, Floats in Speedchart als Bitstrings zu identifizieren, Großer Nachteil dieses Ansatzes sind aber die fehlenden mathematischen Operationen. Es müßten alle atomaren Floatoperationen (Floataddition,-subtraktion,-division,-multiplikation) auf Bitstringebene nachgebildet werden. Dies würde aber einen zu großen Aufwand für die Rechenkerngruppe bedeuten.
Eine weitere Lösung wäre z.B. Float sofort in Speedchart als Tupel von Integers zu betrachten, wurde aber auch verworfen.
Eine andere Möglichkeit (die von der Rechenkerngruppe sehr stark beführwortet wurde) ist die direkte Entwicklung unter VHDL. Dabei müßte der gesamte Algorithmus in VHDL geschrieben und simuliert werden. Bei "handgeschriebenen" VHDL-Code ist aber nicht sichergestellt, daß dieser auch synthetisierbar ist. Die Synthetisierbarkeit des Codes ist also nur bei Speedchart gewährleistet, daher wird auch diese Lösung verworfen.
Es wird daher beschlossen, in Speedchart mit Floats zu validieren, und später beim Exportieren nach VHDL/Synopsys Floats in Bitstrings zu portieren. Um zu sehen, ob dieses Prinzip funktioniert, soll die Rechenkerngruppe ein kleines Beispiel ausarbeiten und in Synopsys simulieren sobald der Float-Addierer der Rechenwerkgruppe fertiggestellt ist. Weitere Aktivitäten werden Unabhängig weitergeführt.

Gruppe Interface:

Das Interface von Wolfram Hardt läuft in der Testumgebung soweit stabil, ist aber noch nicht auf INCA portiert.
M.a.W., es funktioniert im Prinzip, allerdings sind keine ausführlichen Tests möglich. Um größere Tests durchzuführen müßte die FPU algorithmis simuliert werden. .
Unklarheit besteht auch noch über das Timing des Interfaces. Lars möchte daher das Verhalten der FPU mit einem Logikanalyser überprüfen. Dies ist aber nicht nötig, da zu allen Bausteinen Timingdiagramme existieren. Die Interfacegruppe soll bis zum nächsten Treffen klären, ob es dabei Probleme auftreten.
Die Sonne scheint.

Gruppe Rechenwerke

Der Float-Addierer sollte bis heute simuliert sein, dies war aber nicht möglich, da dieser angeblich triviale Baustein sich als einer der Kompliziertesten herrausgestellt hat.
Der Float-Addierer wird genau vorgestellt (siehe Abb.1 aus Haeyes). E1 und E2 sind die Exponenten,

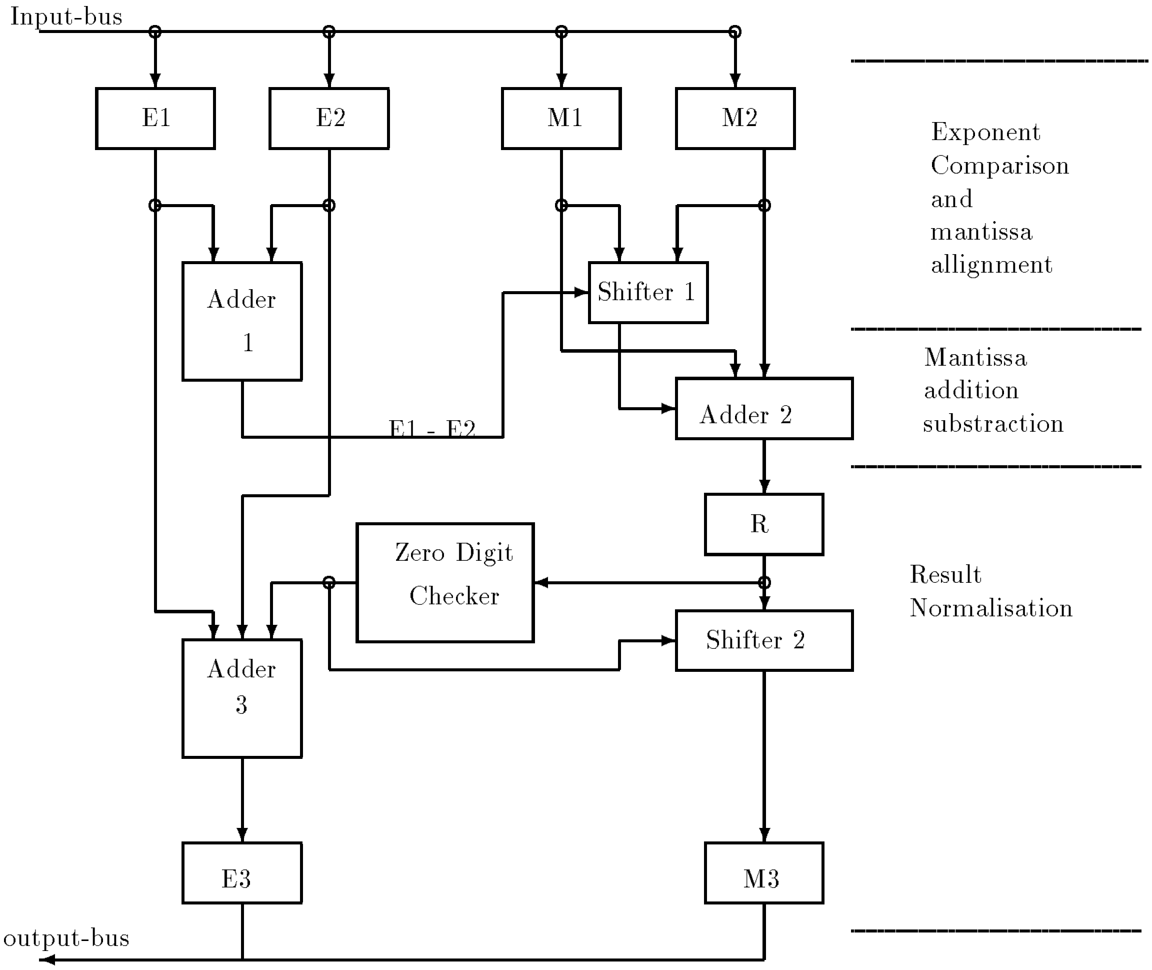

Abbildung 1: Float-Addierer

M1, M2 die Mantissen der beiden Float-Zahlen.

Am optimalsten bzgl. Zeitverbrauch ist es, zum Normalisieren (shiften) eine Links- und einen Rechtsshifter einzusetzen, allerdings benötigt diese Lösung sehr viele Gatter. Man könnte den Addierer auch so Umbauen, das man nur einen Linksshifter benötigt, dann benötigt man aber eine größere Vorschaltlogik. Da aber auf dem Simulator der Speicherplatz sehr beschränkt ist, haben wir einen universellen Links-, Rechtsshifter gewählt, der allerdings mit jedem Takt nur um eine Stelle shiftet. Um nicht ein universelles Schieberegister neuentwickeln zu müssen, das in einem Takt um mehrere Stellen schieben kann, greifen wir auf Ergebnisse der Projektgruppe Systementwurf 92/93 zurück, die bereits einen solches optimieres Register erstellt hat (siehe PG 92/93).
Der Aufbau der Rechenwerke erfolgt stufenweise durch primitive Gatter, die der Libary entnommen wurden. Hiermit werden dann immer kompliziertere Bausteine entwickelt (bootom-up-Entwurf: FF $\rightarrow$ HA $\rightarrow$ FA $\rightarrow$ Int.-Addierer). Mit dieser Vorraussetzung ist das Optimieren auf BitSlice-Ebene (2 Bits werden optimiert) möglich. Es wird daher weiter auf Gatterebene konstruiert.

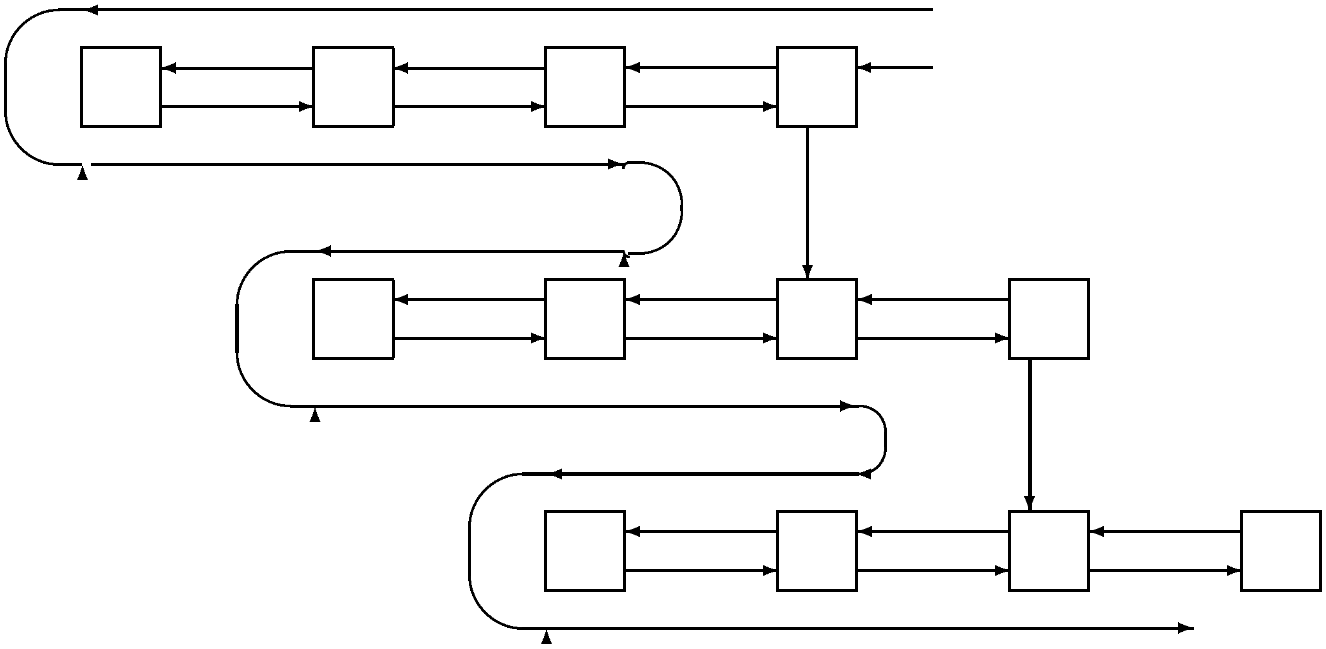

Abbildung 2: Kritischer Pfad im Dividierer

Abbildung 2. zeigt den kritischen Pfad eines Dividierers. Nach Christophs Berechnung liegt die Gatterlaufzeit dieses Pfades bei 454 ns für eine 16-Bit Mantisse, das entspricht einer Taktrate von 2,2 Mhz falls der Division in einem Takt berechnet werden soll. Die Berechnungen beruhen auf Gatterlaufzeiten der ECPD10-Libary (Zum Vergleich, ein Motorola 68000-er benötigt für eine Division 64 Takte). Da sich der SPARC-prozessor der uns zur Verfühgung steht aber auf max. 20 MHz runtertakten läßt, muß eine Division in mind. 16 Takte aufgespalten werden (Pipelining). Der im A30 aufgeführte Dividierer benötigt aber keinen Takt, man muß lediglich warten, bis nach einer bestimmten Zeit die Ausgänge stabil anliegen. Da aber Synopsys keine Wait-Statements synthetisieren kann, kommt eine Realisierung nach dem A30-Paper nicht in Frage. Deshalb, und da bei dem Algorithmus nicht 16 Divisionen hintereinander, bzw. parallel ausgeführt werden, beschliessen wir die Division durch mehrfache Subtraktion darzustellen. Dies optimiert den Gatterverbrauch des Dividieres erheblich. Das oben genannte Problem existiert ebenfalls für den Multiplizierer, es ist daher zu überlegen, ob man analog zum Dividierer den Pipeline-Multiplizierer durch mehrfache Addition darstellt.

Plan für die Semesterferien

Interface:

- Modifikation des Interfaces
- Synthese des Interfaces
- Nach INCA portieren, testen über PC-Interface

alles bis 30.03.95

Rechenkern/Algorithmus:

- Spezifikation Speedchart (bis 16.03.95)

- kleines Beispiel (bis 23.02.95)
- nach Synopsys portieren und simulieren (bis 30.03.95)

Rechenwerke:

- Addierer/Subtrahierer (Integer) synthetisieren (bis 16.03.95)
- Fload-Add./Sub. modelieren und synthetisieren (bis 23.02.95 erste Ergebnisse, fertig bis Ende Feb.)
- Float Mult./Div. (bis 30.03.95)