Es fehlten: F. Rammig, S. Marcus

Organisatorisches:
Es sollen die DVI-Files der Protokolle mit abgelegt werden, solange sie nicht zu viele Postscriptbilder enthalten. Die Autoren der bisherigen Protokolle sollen Ihre DVI Files nachträglich in das Verzeichnis legen.
Die Folien der Vorträge sollen in dem Seminarordner im C2.227 abgeheftet werden.

Zuerst stellte die Gruppe Algorithmusentwurf Ihre bisherige Arbeit vor. Im Vordergrund stand dabei die Einarbeitung in Speedchart. Dabei trat das Problem auf, daß das Programm sich in diesem Jahr wegen Lizenzproblemen nicht mehr starten ließ. Bisher wurde nur ausgetestet, wie Speedchart aussieht und wie es bedient wird. Als Aufgabe wurde gestellt, ein fiktives Programm zum einarbeiten zu erstellen. Dabei ist festzustellen, ob Speedchart es erlaubt, Piplinemultiplizierer zu spezifizieren. Die Frage ist, ob es mit der Automatenmethode, die Speedchart anwendet, überhaupt machbar ist. Die zweite Frage ist, ob das Ergebnis mit Synopsys weiterverarbeitet werden kann, oder ob Befehle benutzt werden, die Synopsys nicht beherscht.

Zum Algorithmus selber wurde gesagt, das man dort noch nach einem anderen sucht. Im Moment würde der Algorithmus von Jürgen verfeinert. Es wurde der Vorschlag gemacht, im Internet weiter nach einem anderen Algorithmus zu suchen, da in der Bibliothek nichts gefunden wurde. Man will versuchen, die Matrix auf eine 3-Band Matrix zu beschränken.

Desweiteren stellte sich die Frage, wann überprüft werden soll, ob es bei der Berechnung der Splines mit den Werten Fehler geben kann. Dies kann Softwaremässig vorher geschehen, oder in der SFU. Es wurde beschloßen, das 0-Matrizen und gleiche X-Stellen durch die Software vorher abgefangen werden.

Die Gruppe Optimierung hat sich bisher die allgemeine Realisierung von Rechenwerken angesehen. Die Literatur für kombinatorische Multiplizierer haben sie jetzt auch. Dabei ist zum nächsten mal zu klären, ob die ES2 Bibliotheken Megazellen enthalten, und ob diese in Synopsys mit eingebunden werden können. Als nächstes will sich die Gruppe in Synopsys einarbeiten und sich kombinatorische Multiplizierer und Dividierer ansehen.

Christof Nagel hat seine Blizidee für einen Multiplizierer vorgestellt.

Wir haben eine 24x24 Matrix mit Volladdierer. Die geschätzte Laufzeit ist unter 8 MHz. Deshalb wird die Matrix in 2 24x12 Matrizen aufgeteilt, in deren Mitte eine Registerschicht eingefügt wird. Wir führen damit Pipelining ein, die Berechnung läuft jetzt über 2 Takte. Nach der Duchlaufzeit + Sicherheitszuschlag sind die Eingänge der Register stabil, und die Werte werden in den Registern gespeichert. Im nächsten Takt werden die Werte dann weiterberechnet. In den ersten Teil des Multiplizierers können dann schon neue Werte nachgeschoben werden. Wir berechnen also $a_{23}...a_0 * b_{23}...b_{12} * 2^{12} + a_{23}...a_0 * b_{11}...b_0$

Mit der Division kann ähnliches gemacht werden.

Die Gruppe Interface hatte keine guten Nachrichten. Da das Bit 13 doch bei einigen FPU Befehlen benutzt wird, muste die bisherige favorisierte Methode des Umschaltens zwischen den beiden Coprozessoren aufgegeben werden. Jetzt soll die SFU am Bus lauschen und anhand des Befehls entscheiden, ob der Befehl für die SFU oder FPU ist. Dazu wird in der SFU eine eigene Fetch implementiert, die, während die IU ihre Fetch ausführt, entscheidet, für wen der Befehl ist und umschaltet. Siehe dazu auch die Grafik am Ende des Protokolls.

Die 4 NAND Gatter zum Umschalten des *fins1* und *fins2* Signales bleiben externe Bauteile, da der

Hardwareemulator dafür zu langsam ist.

31 30	25	19	14	13	5	0
10	rd	Opcode IU	rs1	0	Opcode FPU	rs2
				1	simm13	

Abbildung 1: Aufbau eines Befehls

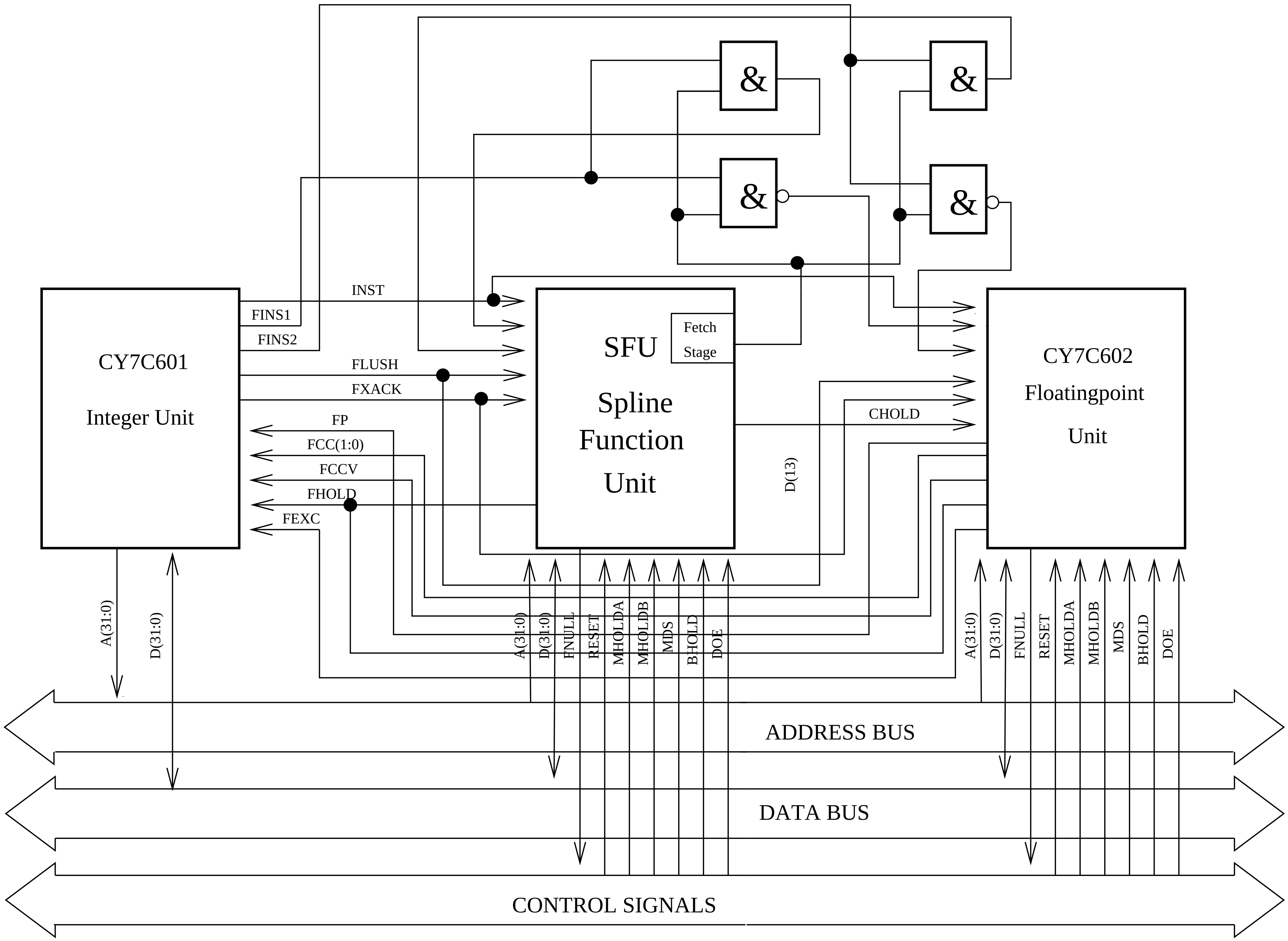
Der Aufbau eines Befehls ist in Abbildung 1 kurz angerissen. Es gibt nur 32 FPU Befehle, aber mit den Bits 5 - 12 lassen sich sehr viel mehr darstellen. Deshalb werden zwei unbenutzte Bitfolgen für die Befehle zum lesen und schreiben der SFU Register benutzt. Somit braucht die SFU auch nur noch die Bits 5 - 12 überprüfen, für wen der Befehl ist. Unbenutzte Bits gibt es leider nicht.

Das Interface von Wolfram Hardt soll im wesentlichen übernommen werden. Änderungen sind aber alleine schon wegen der nicht vorhandenen Hardware nötig. Die Sourcen dazu stehen unter spline/interface.

Es stellte sich die Frage, wie es mit den Ein-/Ausgaberegistern aussieht. Das Interface in der bisherigen Version sieht vor, das es 31 Register zum lesen und 31 Register zum schreiben gibt. Die andere Möglichkeit wäre, 31 Register zum lesen und schreiben zu implementieren, die vom Rechenkern auch zum zwischenspeichern benutzt werden können. Die Stimmung war mehr für die zweite Methode, da im jetzigen Fall die Registern während der Rechenzeit unbenutzt bleiben. Dazu kommt, das auch noch die doppelte Anzahl von Registern benötigt wird. Für den ersten Fall spricht ein kleinerer Verwaltungsaufwand und eine klare Hierachie, die die Daten von oben nach unten durchlaufen.

Bei 31 Schreib-/Leseregistern stellt sich dann die Frage, ob 1 Valid Bit für alle Register reicht, oder ob fuer jedes Register ein eigenes Valid Bit benötigt wird. Laut Algorithmen Gruppe werden alle Register gleichzeitig benötigt, dann reicht ein Valid Bit. Bei 32 Valid Bits muß der Rechenkern darauf achten, das die Daten nicht synchron ankommen, da das Datenprotokoll asynchron ist.

Da die Frage ist, welche Methode billiger ist (Logik zum abragen, 32 Valid Flags, etc.), wurde die Entscheidung vertagt. Nach Möglichkeit soll die Frage erst entschieden werden, wenn näheres über den Aufwand bekannt ist. Es sei denn, das Interface benötigt vorher eine Entscheidung, dann soll eine Methode festgelegt werden, die zu dem Zeitpunkt am günstigsten scheint.



CY7C601
Integer Unit
SFU
Fetch Stage
Spline Function Unit
CY7C602
Floatingpoint Unit
&
INST
FINS1
FINS2
FLUSH
FXACK
FP
FCC(1:0)
FCCV
FHOLD
FEXC
CHOLD
D(13)
A(31:0)
D(31:0)
FNULL
RESET
MHOLDA
MHOLDB
MDS
BHOLD
DOE
ADDRESS BUS
DATA BUS
CONTROL SIGNALS