ENTWURF VON ASYNCHRONEN LOGIKSYSTEMEN

P. Balog

HTL und FhE am Technologischen Gewerbemuseum, Wien

ZUSAMMENFASSUNG:
Heute basieren fast alle modernen Entwurfsstile im Bereich der Digitaltechnik auf zeitdiskreten, synchronen Modellen. Mit steigender Verarbeitungsgeschwindigkeit und höheren Integrationstechniken beeinflussen die Grenzen dieser Modelle entscheidend den Entwurf und die Funktionalität der Produkte. Moderne asynchrone Modelle stellen in vielen Fällen eine echte Alternative dar.

Einleitung

Asynchrone digitale Systeme /1/ /2/ /3/ /8/ sind sequentielle Schaltungen welche keinen externen, globalen Takt zur Koordination ihrer internen Abläufe benötigen.
In den Anfängen der digitalen Schaltungstechnik mit integrierten Bausteinen erfolgte der Systementwurf nach heuristischen, manuellen Methoden und lieferte i.a. asynchrone Schaltungen. Die Blütezeit der (manuellen) asynchronen Schaltungstechnik waren die 60er Jahre. In dieser Zeit findet man auch die ersten Überlegungen in Richtung des systematischen Entwurfs von (asynchronen) Logiksystemen. Mit der Einführung der taktflankengesteuerten Flip-Flops und dem Modell des synchronen Schaltwerkes (*clocked FSM*) mit den entsprechenden Spezifikations-, Verfikations- und Synthesemethoden in den 70er Jahren trat das Interesse an systematisch entworfenen asynchronen Systemen in den Hintergrund. In den 80er Jahren verschwand diese Thematik komplett aus der Literatur.
Das Modell der synchronen Systeme erlaubt die Abstraktion von den Implementierungsdetails der notwendigen kombinatorischen Logik, da die Ausgänge dieser

nur zu diskreten Zeitpunkten abgetastet werden. Diese Abtastzeitpunkte sind durch einen externen, globalen Systemtakt vorgegeben. Der einzige Parameter der kombinatorischen Logik der für den Systementwurf von Bedeutung ist, ist die maximale Verzögerungszeit. Laufzeitprobleme innerhalb der kombinatorischen Logik, *Hazards* und infolge *Glitches* an den Ausgängen der kombinatorischen Logik sind ohne Belang, wenn sie innerhalb der definierten maximalen Verzögerungszeit auftreten. Aus dieser Zeit und den Flip-Flop Parametern kann dann eine maximale Systemtaktfrequenz errechnet werden, bis zu der das System korrekt arbeitet. Das synchrone Konzept vereinfacht den Entwurf entscheidend,- so ist es nicht verwunderlich, daß in den letzten 20 Jahren der Großteil der Entwicklungen an Hardware und CAE-Werkzeugen in dieser Domäne zu finden sind.
Mit der Hochintegrationstechnik auf der einen Seite und dem Verlangen nach immer größeren Verarbeitungsgeschwindigkeiten auf der anderen Seite gepaart mit dem Wunsch von geringerem Leistungsverbrauch sind die Grenzen der synchronen Systeme offensichtlich geworden:

- ***Clock Skew***: In hochintegrierten Bausteinen mit tausenden Flip-Flops ist es von Bedeutung, daß alle Flip-Flops zur selben Zeit mit der Taktflanke versorgt werden. Bei den hohen Verarbeitungsgeschwindigkeiten stellt dies einen entsprechenden Aufwand in der Taktversorgung dar.
- ***Clock Distribution***: Dieser oben angesprochene Aufwand der Taktversorgung hat zwei Aspekte; einerseits die notwendige Chipfläche (Beim DEC - Alpha 21064 Prozessor wurden ca. 30% der Chipfläche für das Taktnetzwerk verbraucht) und andererseits der dauernde Leistungsverbrauch, auch wenn das System nichts tut.
- ***Power Distribution***: Zufolge des dauernden Leistungsverbrauchs des Taktnetzwerkes und der angschlossenen Flip-Flops werden die günstigen statischen Eigenschaften von CMOS sicher nicht ausgenützt. Der Großteil der Stomversorgung sowohl innerhalb des Chips als auch im Gesamtsystem müssen unter dem Aspekt des hohen Stromverbrauchs der synchronen Komponenten betrachtet werden.
- ***Asynchrone Eingänge***: An der (asynchronen) Peripherie von synchronen Systemen und zwischen synchronen Systemen mit unterschiedlichen Taktfrequenzen treten Synchronisierungsprobleme (Metastabiltätsgefahr) auf.

Aus diesen Überlegungen heraus beginnt Ende der 80er Jahre eine intensive Forschung auf dem Gebiet der asynchronen Schaltungstechnik („*The Return of Asynchronous Logic*“ /1/) mit dem Ziel Modelle und Methoden zu kreieren, die einen konstruktiven Entwurf ermöglichen. Die Anwendung von asynchronen Systemen bzw. Teilsystemen bietet sich überall dort an, wo eine „natürliche“ Asynchronität vorliegt; wo also aus asynchronen Eingängen nicht zwingend synchronisierte Ausgänge berechnet werden müssen. Asynchrone Systeme haben ein adaptives Verhalten bezüglich des Leistungsverbrauchs; ein asynchrones System (CMOS) verbraucht keine Energie wenn keine Schaltvorgänge ausgeführt werden. Damit sind Eigenschaften wie *Power-Save*, *Standby*, *Hot-Standby*, etc. automatisch gegeben, die bei synchronen Systemen einen erheblichen konstruktiven Aufwand darstellen. Außerdem sei noch erwähnt, daß asynchrone Systeme „maximal schnell“ arbeiten, da keine diskreten Synchronisationszeitpunkte abgewartet werden müssen,- dies führt zu sogenannten *Low-Latency Outputs*.

Stand der Technik

Der asynchrone Logikentwurf gestaltet sich weitaus schwieriger als der synchrone, da den *Hazards* in kombinatorischen Logiksystemen /5/ /7/ , und damit der Hardwareimplementierung eine enorme Bedeutung zukommt. Der manuelle Entwurf von korrekt arbeitenden Systemen ist sogar für kleinste Aufgaben fast unmöglich. Es gilt daher Modelle zu kreieren, die sowohl eine Spezifikation als auch eine Validierung (Verifikation, Simulation, Test) ermöglichen und weiters Algorithmen bieten, die eine automatisierte Synthese in die gewählte Zieltechnologie (CMOS, FPL) ermöglichen (*hazard free implementation* /6//7/).
Ein asynchrones System bzw. Teilsystem (z.B. ein *Controller*) darf in seiner Funktion nicht isoliert betrachtet werden, sondern die Umgebung in der er arbeitet muß mit einbezogen werden. Aus diesem Grund teilt man die Modelle in *Circuit-* und *Environment-Models* ein:

- ***Circuit-Models***:
  Eine Schaltung besteht aus logischen Elementen bzw. Gattern (*gates*) und den Verbindungen (*wires*). Die korrekte Funktion ist nun von den *Gate-Delays* und/oder *Wire-Delays* abhängig bzw. unabhängig, was die folgende grobe Einteilung der asynchronen Logiksysteme in *Bounded-Delay-*, *Delay-Insensitive-* und *Speed-Independent-Circuits* ermöglicht.

- *Bounded-Delay-Model*:

  Die Verzögerungen (*delays*) in allen Gattern und Verbindungen sind bekannt oder zumindest begrenzt. Synchrone Schaltwerke basieren ebenfalls auf diesem Modell.

- *Delay-Insensitive-Model*:

  Sowohl die Verzögerungen in den Gattern als auch in den Verbindungen sind unbegrenzt. Ein DI-Schaltung besteht i.a. aus Makromodulen mit einem Verbindungssystem. Die Makromodule sind so entworfen, daß jede Variation im Verzögerungsverhalten des Verbindungssystems keinen Einfluß auf die korrekte Funktion hat. Der interne Aufbau der Makromodule muß nicht zwingend auf dem DI-Modell basieren, jedoch die Kommunikation zwischen den Modulen. Die Implementierung von DI-Systemen ausgehend von textuellen oder graphenteheoretischen Spezifikationen und deren Verifikation stellt in den letzten Jahren ein großes Interessensgebiet dar, mit einer hohen Zahl an Publikationen /9/.

- *Speed-Independent-Model*:

  Die Verzögerungen der Gatter ist unbegrenzt und die Verzögerungen der Verbindungen sind vernachlässigbar. Wenn ein Ausgangssignal an mehrere Eingänge gelegt wird, so erreicht das Ausgangssignal alle Eingänge gleichzeitig (*isochronic fork*). Dieses Modell war in der Anfangszeit der VLSI-Technik üblich, da die Laufzeiten durch die Logikfunktionen viel größer war als die durch die Verbindungselemente. Bei heutigen Integrationsdichten und Geschwindigkeiten ist dieses Modell nur mehr beschränkt anwendbar.

- ***Environment-Models***:

  Diese Modelle beschreiben, wie die Umgebung auf die Schaltung reagiert bzw. reagieren darf. Folgende Modelle werden verwendet:

  - *Fundamental-Mode*:

    Die Eingänge dürfen erst dann einen neuen Wert annehmen, wenn das System zufolge des alten Wertes einen stabilen Zustand erreicht hat.

  - *Input-Output-Mode*:

    Die Eingänge dürfen dann einen neuen Wert annehmen, wenn die Ausgänge des Systems zufolge des alten Wertes eine Reaktion zeigen.

  - *Single-Input-Change*:

    Eine Wertänderung am Eingang des Systems darf nur durch die monotone Änderung eines einzigen Bits erfolgen.

- *Multiple-Input-Change*:
  An einer Wertänderung am Eingang des Systems dürfen mehrere Eingangsbits (z.B. in Form eines Eingangs-*Bursts*) beteiligt sein. Übliche Einschränkung liegen darin, daß die Eingangssignalinformation in den Flanken und nicht in den Pegeln liegt (*transition signaling*) und das die Signale monoton sind.

Asynchrone Entwurfsmethoden werden sowohl für Steuerungsschaltungen (Automaten, *Sequencer*, *Controller*) als auch für Datenpfadanwendungen eingesetzt. Der Hauptanwendungsbereich liegt wahrscheinlich im Bereich der *Controller* und Schnittstellenschaltungen (*Interfaces*) in heterogenen (gemischt synchron-asynchronen, multigetaktet synchronen) Umgebungen.

Asynchrone Datenpfade

Herkömmliche synchrone Datenpfade sind *Pipelines*; jede Stufe der Pipeline besteht aus einer kombinatorischen Logik mit nachgeschaltetem Register, das die Daten für die nächste Stufe zwischenspeichert. Die *Pipeline* wird von einem globalen Takt angesteuert.
Asynchrone Datenpfade basieren auf einem *Handshake*-Mechanismus zwischen den einzelnen Verarbeitungsstufen. In Sutherlands *Micropipeline* /2/, die 1989 vorgestellt wurde, wird das 2-Phasen-*Handshake*-Protokoll definiert, welches sowohl die steigende als auch die fallende Flanke der verwendeten *Request*- und *Acknowledge*-Leitung verwendet. Die Daten werden zwischen den einzelnen Stufen als Datenbündel (*bundled data*) übertragen und in speziellen, *handshake*-fähigen Latches zwischengespeichert, wobei sicher gestellt werden muß, daß die Daten vor den *Handshake*-Signalen ankommen. Ein etwas modifizierter Ansatz /6/, der mit „normalen“, flankengesteuerten Registern das Auslangen findet, verwendet das 4-Phasen-*Handshake*-Protokoll, bei dem nach einer 2-phasigen Signalisierung 2 Rücksetzphasen (*return to zero*) erfolgen. Methoden die auf *Bundled-Data* mit seperaten *Handshake*-Leitungen basieren fallen nicht in die Klasse der *Delay-Insensitive-Circuits*. Bei echten DI-Implementierungen wird jedes Datenbit *Dual-Rail-Encoded*,- d.h. für jedes Datenbit (Di) werden 2 Signale (Di.t und Di.f) übertragen. Man verwendet meist das 4-Phasen-*Handshake*-Protokoll, wobei jedes Datenbitpaar sowohl den Datenwert (0 oder 1) als auch den *Request* enthält. Im Empfänger befindet sich eine *Completion-Detection*-Logik, die erkennt, daß alle Datenbits eingetroffen sind. Der Ausgang des *Completion-Detectors* stellt überdies den notwendigen *Acknowledge* zum Sender dar.

Asynchrone Steuerungen (*Controller*)

Für die Beschreibung von asynchronen Ablaufsteuerungen gibt es eine Vielzahl von Methoden und Modellen,- von textuellen, an parallele Hochsprachen (z.B. OCCAM) bzw. Formalismen (z.B. CSP) angelehnte über graphentheoretische (interpretierte Petri-Netze und deren Abwandlungen) bis hin zu den AFSMs (*Asynchronous Finite State Machines*). Wichtig für den Konstrukteur ist, daß das gewählte Modell mit seiner Spezifikationsmethode auch eine Implementierung mit automatisierter Synthese ermöglicht. In dem Beitrag „*Asynchrone State-Machines*“ in diesem Tagungsband findet sich eine kurze Einführung in diese Thematik mit dem Schwerpunkt auf den *Extended-Burst-Mode* Automaten /4/, dessen Implementierung und den Synthesemöglichkeiten.

Schlußbemerkungen

Mit dem asynchroner System- und Schaltungsentwurf ist ein stetig wachsender Forschungsbereich befaßt, der in engem Zusammenhang mit den Arbeiten auf dem Gebiet der formalen Spezifikation und Verifikation von digitalen System steht. Viele unterschiedliche Spezifikationsmethoden, Implementierungstechniken und Syntheseansätze stehen derzeit zur Diskussion. Entwurfswerkzeuge (z.B. 3D, SIS, ASSASSIN, VERDECT) stehen vor allem im akademischen Bereich zur Verfügung. In den nächsten Jahren ist damit zu rechnen, daß die asynchronen Entwurfsstile auch in kommerziellen CAE-Werkzeugen angeboten werden. Während die Implementierungen derzeit meist auf *Full-Custom*- oder *Standard-Cell-ASICs* beschränkt sind, werden zukünftig dem Konstrukteur auch neue (asynchrone) FPL-Bausteine /6/ als Zieltechnologie zur Verfügung stehen. Es ist sicher nicht mit einem Paradigmenwechsel von synchroner zu asynchroner Entwurfsmethodik zu rechnen, jedoch bieten sich in vielen Bereichen Chancen für asynchrone Entwürfe um die synchronen Methoden sinnvoll zu ergänzen.

Literatur:

/1/ S. B. Furber, „*The Return of Asynchronous Logic*“, http://www.cs.man.ac.uk/amulet/async/async_desc.html

/2/ D. Pountain, „*Computing Without Clock*“, Byte Magazine, Jannuary 1993, p145-p150

/3/ S. Hauck, „*Asynchronous Design Methodologies: An Overview*“, Proceedings of the IEEE, Vol. 83, No. 1, Jannuary 1995, p69-p93

/4/ K. Y. Yun, „*Synthesis of Asynchronous Controllers for Heterogeneous Systems*“, PhD thesis, Stanford, 1994, p1-15, p18-31

/5/ L. Lavagno, „*Synthesis and Testing of Bounded Wire Delay Asynchronous Circuits from Signal Transition Graphs*“, PhD thesis, UCB, 1992, p25-106

/6/ R. Payne, „*Self-Timed FPGA Systems*“, published in FPL’95, 1995

/7/ K. Maheswaran, V. Akella, „*Hazard-Free Implementation of the Self-Timed Cell Set in a Xilinx FPGA*“, technical report, U.C.Davis, 1994

/8/ „*Asynchronous Logic Homepage*“, http://www.cs.man.ac.uk/amulet/async bzw. *Mirrorsite* http://maveric0.uwaterloo.ca/amulet/async/index.html

/9/ „*Encyclopedia of Delay-Insensitive Systems (EDIS)*“, http://www.win.tue.nl/cs/pa/edis/edis.html