

Asynchrone Logik – ein Überblick

Hohe Leistung bei kleiner Leistungsaufnahme – kein Widerspruch mehr

Im Augenblick werden die meisten Chips in CMOS hergestellt. Mit höherer Frequenz steigt der Stromverbrauch linear an, und auch die Störstrahlung steigt, da die Metallisierung wie eine Antenne wirkt. Diese Chips werden fast zu 100 Prozent in so genannter synchroner Logik entwickelt, eine Technologie, die bei manchen Designs an ihre Grenzen stößt. Man kann zwar nicht-aktive Bereiche abschalten, aber gibt es auch andere Methoden? Wie kann man die Taktprobleme großer Chips in den Griff bekommen? Schlagwörter in diesem Zusammenhang sind: IP, Reuse, Einkauf fremder Funktionsblöcke, geringe Stromaufnahme, Batteriebetrieb, Entwicklungszeit, Verifikation und Testbarkeit. Eine Lösungsmöglichkeit liegt im Einsatz asynchroner Logik. Dieser Artikel beschreibt, was das eigentlich ist, was die Unterschiede sind und wo die Vorteile liegen.

Neue System-on-Chip-Designs werden immer komplexer, benötigen immer mehr Gatter und erfordern immer längere Entwicklungszeiten. Die Maskenkosten steigen bei den kleinen Linienbreiten so stark an, dass die Mindeststückzahl für ein ASIC immer größer wird, das Risiko eines Re-designs steigt und jedes Mal ein neuer teurer Maskensatz erforderlich ist sowie zwei bis sechs Monate Wartezeit bis zu den neuen Samples vergehen. Ein weiteres Problem ist die Stromaufnahme dieser SoCs: Mit den steigenden Taktfrequenzen wird zwar immer mehr Durchsatz erreicht, aber auch mehr Strom verbraucht, denn die Stromaufnahme eines CMOS-Bausteins steigt

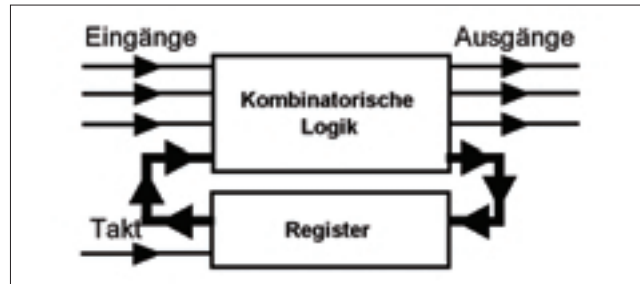


Bild 1a. State Machine (FSM = Finite State Machine), zeitdiskret, Status in Speicherzellen gehalten

linear mit der Taktfrequenz. Da die Komplexität steigt, die Entwicklungszeit aber nicht zu stark ansteigen darf, werden Funktionsblöcke als IP (Intellectual Property) von Drittanbietern zugekauft. Die Integration ergibt neue Risiken, da das »Innenleben« des Blocks unbekannt ist. Viele ICs werden für mobile Anwendungen eingesetzt, also muss eine möglichst geringe Stromaufnahme erreicht werden, damit die Batterie lange hält – ein Widerspruch bei immer höherem Takt und mehr Gattern. Der Einsatz asynchroner Techniken bietet hier einen Ausweg.

Schon vor 20 Jahren war asynchrone Logik nichts Neues, aber immer sind die Anwender vor ihrem Einsatz zurückgeschreckt, denn es gab keine Werkzeuge. Die Entwicklungs-Tools für synchrone Designs haben dazu geführt, dass große Fortschritte bei der Produktivität der Entwickler erreicht wurden. Wenn alle Signale eines Systems sich zum selben Zeitpunkt ändern, wird die Verifikation des Systems vereinfacht. Es muss »lediglich« sichergestellt werden, dass die Verzögerungen in der Kombinatorik zwischen den Registern eingehalten werden. Dies ist nur »Arbeit« im Vergleich zum asynchronen Entwicklungsprozess, bei dem alle Ablaufdetails sehr genau betrachtet werden müssen: Welches Signal ist schneller, wo sind instabile Zustände, oder ob das System in einem Zustand stecken bleiben kann. Aber man sollte nicht vergessen, dass asynchrone Systeme eigentlich nichts Besonderes sind. Eher umgekehrt: Ein synchrones System stellt einen Spezialfall der asynchrone

nen Entwicklungsmöglichkeiten dar.

Ein Beispiel für ein asynchrones System im realen Leben: Soldaten und eine Brücke – im Gleichschritt besteht eine Gefahr von Schwingungen der Brücke, ohne Gleichschritt verschwindet das Problem. Dies kann auch als Beispiel

für die Unterschiede in der Störstrahlung dienen: Das synchrone System hat starke Spitzen an den Taktflanken, asynchron ergibt sich eine Verteilung und weniger Störungen (Tabelle 3).

Wie funktioniert nun synchron und welche Möglichkeiten gibt es bei asynchroner Implementierung? Bei einem synchronen System werden alle Eingangs- und Ausgangssignale synchronisiert. Man kann den Chip also als große synchrone State-Machine ansehen (Bild 1a). Sie besteht aus zwei Blöcken: dem Teil für die logischen Verknüpfungen ohne Rückführungen und dem Register, in dem alle Zustände zu einem bestimmten Zeitpunkt über das Taktsignal gespeichert werden und dann als zusätzliche Eingänge im kombinatorischen Teil erscheinen. Steigt die Taktflanke, wird der Zustand am Ausgang der Kombinatorik »eingefroren«. Der kombinatorische Teil muss sich stabilisiert haben, bevor die nächste Taktflanke erscheint. Die Taktperiode muss also länger sein als der kritische (längste) Pfad. Umgekehrt muss die Taktfrequenz hoch genug sein, damit keine Änderung des Eingangssignals ausgelassen wird (Bild 1b). Da alle Signale in dieses Zeitraster gezwun-

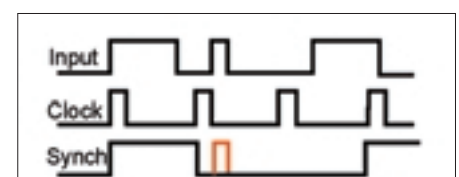


Bild 1b. Synchronisieren des Eingangssignals und mögliche Probleme

gen werden, ergibt sich eine klare Zuordnung sowohl für das Design als auch für Verifikation und Test.

Wenn wie in Bild 1a alles recht nahe beieinander liegt, ist das recht übersichtlich, aber Bild 1c zeigt, was sich in der Realität auf dem Chip ergibt. Die einzelnen Register liegen nicht mehr nebeneinander, sondern sind bei den teilweise Millionen Gattern über den gesamten Chip verteilt. Die längste und kritischste Verbindung ist also das Taktnetzwerk, das mehrfach gepuffert werden muss. Bei den hohen Taktfrequenzen wirkt dieser Taktpfad wie eine Antenne und hat wegen der hohen Frequenz auch eine hohe Stromaufnahme. Diese Leistung wird auch verbraucht, wenn gar nichts auf dem Chip passiert.

In der Vergangenheit waren die Verhältnisse auf dem Chip so, dass die Verzögerungen in den Metallisierungen recht kurz waren, fast immer sind die Signale dort schneller gewesen als die innerhalb der Funktion. Damit gab es kein Taktproblem. Das hat sich bei den heutigen komplexen SoCs mit kleinen Linienbreiten geändert. Hier werden die »Drähte« auf dem Chip so lang, dass eine Synchronisation aller Clock-Signale immer schwieriger wird. Der schwierigste »Draht« ist damit der Takt, denn der muss alle Register verbinden; und alle Signale am Eingang der Register müssen korrekt sein. Bei asynchroner Logik fehlt dieser übergeordnete Takt. Damit muss jedes Subsystem unabhängig von allen anderen arbeiten, und die korrekte Zusammenarbeit der einzelnen Blöcke muss auf andere Weise sichergestellt werden. Asynchrone Systeme kann man am einfachsten in drei Gruppen aufteilen, jeweils für verschiedene Anforderungen: kleine, mittlere und große Systeme.

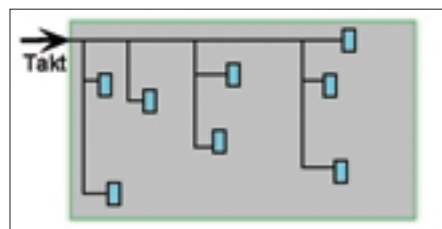


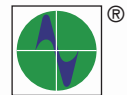
Bild 1c. Der lange Taktbaum auf dem Chip

Welche Aspekte gibt es hier und was bedeuten sie (Tabelle 1): Es gibt die Signale, Signalübertragung und die Ausbreitung auf dem Chip. Die Boole'sche Algebra beschreibt das logische Verhalten in idealer Weise, betrachtet aber in keiner Weise die Verzögerungen in der Logik, Laufzeiten, Abhängigkeiten von der Betriebsspannung, der Temperatur, Längen der Verbindungsleitung oder Fertigungstoleranzen. Ein Signal wird normalerweise über High und Low definiert. Damit können langsam ansteigende Flanken Verzögerungen erzeugen. Besser wäre es, nur auf Signaländerung zu reagieren, also die Taktflanke zu verarbeiten. Alternativ kann man jedes High- und Low-Signal auf einer separaten Leitung führen (Bild 3). Das erhöht zwar die Verdrahtung, aber es gibt nur noch aktive Signale, bei denen dann beim selben Signal nur eines von beiden aktiv sein darf.

Müssen mehrere Leitungen weitergeführt werden, kann man die Datenleitungen mit dem entsprechenden Taktsignal bündeln, sodass die relative Beziehung erhalten bleibt. Das System sollte so entwickelt werden, dass es geschwindigkeits- und verzögerungsunabhängig ist. Die am häufigsten eingesetzte Technik bei asynchronen Designs ist im Augenblick die doppelte Signalführung mit pegelsensitiver Signalcodierung. Auf diese Weise können vollkommen verzögerungsun-

den die »Drähte« auf dem Chip so lang, dass eine Synchronisation aller Clock-Signale immer schwieriger wird. Der schwierigste »Draht« ist damit der Takt, denn der muss alle Register verbinden; und alle Signale am

Wir erfüllen auch ganz individuelle Wünsche



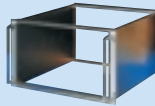
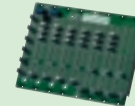
Besuchen Sie uns auf der Electronica 2002 vom 12.-15. November: Halle A2, Stand A2.407

SYSTEMS

BACKPLANES

ROTARY COMPONENTS

ELMASET



Manchmal wünscht man sich Dinge, die es so eigentlich gar nicht gibt. Dann braucht man eine individuelle Lösung – etwas Maßgeschneidertes.

Elma Trenew bietet Ihnen neben einer Fülle von Standardprodukten ein umfangreiches Know-How bei speziellen Aufgabenstellungen.

Unser Full-Service bedeutet für Sie:

Kompetente Beratung: Von der Idee bis zur Serie unterstützen wir Sie mit unserem Know-How.

Individuelle Entwicklung: Nutzen Sie unser weltweit vernetztes Design mit neuesten CAD-Tools.

Modernste Fertigung: Fortschrittliche Produktionsanlagen garantieren Ihnen höchste Qualitätsstandards.

ELMA
Your Solution Partner

TRENEW
An Elma Company

Elma Trenew Electronic GmbH
Stuttgarter Straße 11
D-75179 Pforzheim
Tel. +49 (0) 72 31 97 34 0
Fax +49 (0) 72 31 97 34 97
www.elma.de · info@elma.de

Module	synchron	NCL	Verhältnis
AddrConv	41	62	1.5
X2VHD	395	826	2.1
Decoder	1010	1804	1.8
32x16FIFO	1691	1123	0.7

Tabelle 2. Vergleich der Transistoren bei synchronem und NCL-Design, entweder als direkte Implementierung oder als Redesign in NCL

abhängige Schaltungen aufgebaut werden. Betriebsspannung, Temperatur und Fertigungstoleranzen beeinflussen nicht mehr die Funktion, sondern nur noch den Durchsatz.

- Kleine asynchrone Systeme oder Blöcke: Hier kommen zum Beispiel Kontrollfunktionen in Betracht, die relativ klein sind und damit gut beschrieben werden können. In Bild 1 wurde die Statemaschine gezeigt. Stellt man über das Design sicher, dass sich maximal ein Signal auf einmal ändern kann (Beispiel: die zehn Tasten einer Dezimaltastatur oder Codierung im Gray-Code) und die Signale sich stabilisiert haben, bevor es zur nächsten Änderung kommt, kann eine Huffman-FSM ohne Takt eingesetzt werden (Bild 2).

- Doppelte Leitungsführung: Eine andere Möglichkeit besteht darin, nur aktive Signale zuzulassen. Es wird oft vergessen, dass bei High und Low beide Signalpegel ein aktives Signal darstellen. Teilt man High und Low auf zwei Leitungen auf, ist automatisch auf beiden Leitungen der Takt implizit mit codiert (Bild 3). Es ist dann nur noch der High-Zustand wichtig, Low bekommt die Bedeutung »inaktiv«. Auf diese Weise führt dann jedes aktive Signal den eigenen Takt mit sich. Gibt es keine Änderungen, schaltet nichts, und bei CMOS bleibt damit nur der Reststrom. Interessant wird es am Gatter mit den verschiedenen Eingängen: Es ist so aufgebaut, dass der Ausgang sich nur ändert, wenn alle Eingangssignale aktiv sind und damit vollkommen unabhängig von irgendwelchen Verzögerungen. Es ist wie bei einer Wahl. Man kann sozusagen eine Ja-, eine Nein- oder keine Stimme

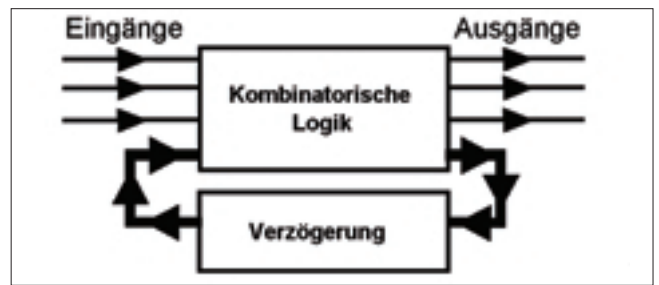


Bild 2. Huffman asynchrone FSM (ohne Takt), zeitkontinuierlich, Gedächtnis durch Selbsthaltung in kombinatorischen Schleifen

abgeben – beide gleichzeitig sind nicht erlaubt. Dies funktioniert nicht nur auf der Gatter-Ebene, sondern es sind auf diese Weise auch komplette Mikrocontroller entwickelt worden. NCL (Null Convention Logic) ist eine von Theseus patentierte Technik auf dieser Basis. Die komplizierteren Aspekte von NCL sind hier nicht beschrieben, aber auf deren Website zu finden.

Ein Beispiel für ein asynchrones Gatter mit zwei Eingängen ist das Muller-Element (Bild 4). Durch die Rückführung ergibt sich ein

Aspekt	Erklärung
Datenübertragung über Potenzial	0 und 1 wie in normaler Logik. Der Übergang von Null auf Eins kann Probleme machen, je nach Anstieg der Flanke. Der Übergang stellt einen dritten logischen Pegel (»unknown«) dar.
Datenübertragung über Flanke	Erst nach der Flanke ist das Signal akzeptiert. Damit spielen Spannungspegel und Flankensteilheit keine Rolle mehr – außer durch die Verzögerung.
Daten bündeln	Bei Datenbussen werden jeweils eine Gruppe von Daten zusammen mit dem Taktsignal übertragen (Bild 1).
doppelte Signalführung	Low und High werden auf separaten Drähten als aktive Signale geführt. Nur das jeweilige High-Signal ist relevant (Bild 2).
geschwindigkeitsunabhängig	Das System arbeitet unabhängig von der Geschwindigkeit der Gatter, verursacht durch Betriebsspannung, Temperatur, Fertigungsstreuungen. Die Empfindlichkeit gegenüber Verzögerungen in den Verbindungen bleibt.
verzögerungsunempfindlich	Hier ist eine vollkommene Unabhängigkeit von allen Verzögerungen erreicht.
Das Optimum wäre damit	Verzögerungsunempfindlich, doppelte Signalführung, Datengültigkeit nur nach Flankenübergang.

Tabelle 1. Einige Aspekte bei der Implementierung asynchroner Systeme

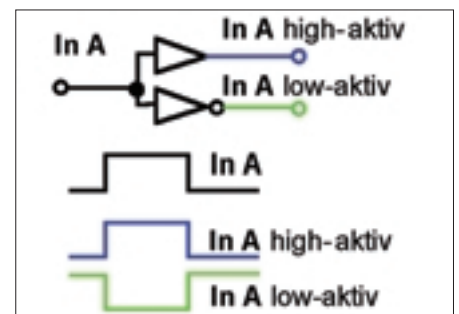


Bild 3. Aufsplitten des Signals in zwei aktive Leitungen

spezielles zeitabhängiges Verhalten: Ohne Eingangssignal ist der Ausgang inaktiv (low). Wenn das erste Signal ankommt, passiert noch nichts. Ist der zweite Ausgang auch aktiv, schaltet der Ausgang auf aktiv (high) um. Schaltet ein Eingangssignal auf inaktiv (low), bleibt das Ausgangssignal noch erhalten. Erst wenn beide Signale inaktiv sind, geht auch der Ausgang in die Inaktiv-Position (low).

Bei größeren Systemen kommt zu den Kontrollfunktionen das Routen von parallelen Signalen und Bussen hinzu, und die korrekte Datenübergabe muss organisiert werden.

- Datenübergabe mit zusätzlichem »Fertig«-

- instruktionskompatibel mit dem Motorola HC05, HC08, STAR08
- Core implementiert in taktfreiem NCL
- 40 Prozent weniger Leistungsaufnahme als synchroner Core
- keine Verzögerung aus STOP- und WAIT-Mode
- etwa 10 dB weniger Störstrahlung
- 1,6 bis 3,6 V Betriebsspannung
- 44 Pin QFP-Gehäuse
- weitere Daten bei www.theseus.com

Tabelle 4. Einige Features des asynchronen NCL-Prozessors NCL08GP32

Signal: Einzelne Funktionsblöcke kommunizieren die Daten über Signale, die anzeigen, dass die Daten stabil sind (Bild 5). Dies kann zum Beispiel dadurch erreicht werden, dass beim Anlegen der Eingangsdaten über ein Start-Signal eine Verzögerung anläuft, die länger ist, als die Worst-case-Verzögerung in der Kombinatorik. Ändert sich dann das »Fertig«-Signal, sind die Ausgangsdaten stabil. Hier läuft nicht mehr jedes Gatter mit maximaler Geschwindigkeit, sondern der Sicherheits-

faktor beim parallelen »Fertig«-Signal bestimmt den jeweiligen Durchsatz. • Datenübergabe mit Handshake: Besser ist es, wenn die »Nachbarn« zusammenarbeiten, indem der Lieferant bei der passiven Version mitteilt, er hat Daten. Der nächste Empfänger nimmt diese an und quittiert, dass sie angenommen sind. Danach kann ein neuer Zyklus beginnen (Bild 6). Daten und Handshake laufen also in dieselbe Richtung (nach rechts). Bei der aktiven Version legt der »Kunde« die Anforderung an, weil er frei ist und wartet auf Daten vom Lieferanten. Die Lieferung wird vom Lieferanten quittiert.

Hier laufen Daten und Anforderung in entgegengesetzte Richtungen.

- Mikropipelines: Kombiniert man die Datenbearbeitung mit paralleler Verzögerung und Datenquittierung, erhält man eine elegante Lösung für asynchrone Systeme (Bild 7). Eine ähnliche Art der Datenführung wurde beim Amulet-Prozessor eingesetzt, der auf dem ARM-Core basiert. Um Zeitverschiebungen bei aktiven Daten zu koordinieren, sind zusätzlich Funktionen wie zum Beispiel FIFOs und LIFOs notwendig, die die asynchron eintreffenden Daten zur Weiterverarbeitung bereithalten.

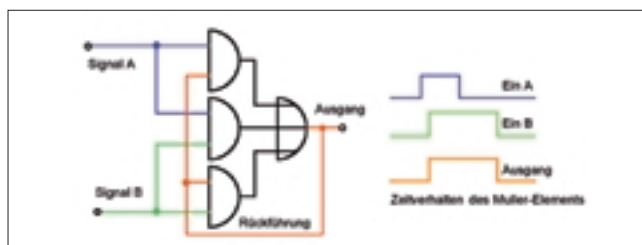


Bild 4. Das Muller-Element, es realisiert eine UND-Schaltung

FLASH-Microcontrollers

Sparen Sie wichtige Entwicklungszeit

Neben der Leistung eines 16-Bit Cores bietet der neue PIC18 FLASH-Microcontroller von Microchip bis zu 32kByte selbstprogrammierbaren FLASH-Speicher und eine Mio Lsch-/Schreibzyklen.

Weitere PIC FLASH Microcontroller sind mit verschiedensten Peripherieschaltungen erhältlich:

- Mehrere Schnittstellen wie I²C[™], CAN und RS-232
- 10-bit A/D-Wandler

Alles für schnelles Entwickeln, flexibel genug für Änderungen während der gesamten Produkt-Lebensdauer.



Microchips Weltklasse-Entwicklungstools wie die innovative integrierte Entwicklungsumgebung MPLAB[®] und der Low-cost In-Circuit Debugger beschleunigen die Prototypen-Erstellung, damit Sie schnellstens Umsatz machen.

Testen Sie die breite Palette reprogrammierbarer Microchip FLASH-MCUs dann wird klar, warum PIC MCUs heute die Architektur der Wahl sind.

PIC FLASH Microcontrollers										
Package Size	Processor	Bits in Production	Memory Configurations (Bytes)			Digital I/O	Communication Ports	Analogue Interface (Channels)	Low-Cost Flash In-Circuit Debug	Development Tool Site
			Programme	RAM	E ² PROM					
18/20 Pin	PIC16	5	1.7K-3.5K	68-224	64-128	13	RS-232, I ² C/SPI [™]	-	Available	Available
28 Pin	PIC16	6	3.5K-14K	128-368	64-256	22	RS-232, I ² C/SPI	10-bit ADC (5)	Available	Available
28 Pin	PIC18	4	16K-32K	768-1536	256	23	RS-232, I ² C/SPI, CAN 2.0B	10-bit ADC (5)	Available	Available
40/44 Pin	PIC16	5	3.5K-14K	128-368	64-256	33	RS-232, I ² C/SPI	10-bit ADC (8)	Available	Available
40/44 Pin	PIC18	4	16K-32K	768-1536	256	34	RS-232, I ² C/SPI, CAN 2.0B	10-bit ADC (8)	Available	Available

PIC18F452
 • 10-bit A/D-Wandler
 • RS-232
 • 10 MIPS @ 10 MHz

PIC16F628
 • 2 analoge Komparatoren
 • USART
 • Interner Oszillator

Weitere Informationen finden Sie auf unserer Website.
www.microchip.com/flashmcu



Der Name Microchip und das Microchip-Logo, PIC und MPLAB sind eingetragene Warenzeichen von Microchip Technology, Inc. in den USA und anderen Ländern. SPI ist ein Warenzeichen von Motorola. I²C ist ein Warenzeichen der Philips Corporation. © 2001 Microchip Technology Inc. Alle Rechte vorbehalten. Ref: ME1102er/04.02

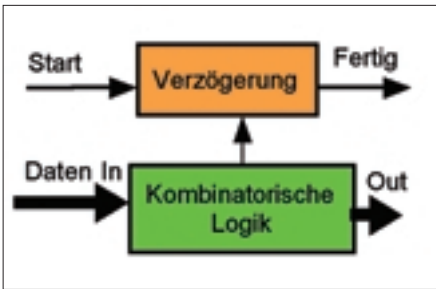


Bild 5. Pipeline über Verzögerung

Das Problem asynchroner Datenverarbeitung gibt es nicht nur bei einem Prozessor. Genau dieselben Probleme ergeben sich bei der Parallelverarbeitung auf mehreren Prozessoren oder bei der Verarbeitung paralleler Prozesse auf einem Prozessor, wenn diese nicht synchronisiert sind. In beiden Fällen muss sichergestellt werden, dass die Daten für den nächsten Einsatz korrekt sind. Es sei hier nur an Petri-Netze erinnert oder an kommunizierende sequenzielle Prozesse (GSP) und an den Transputer, dessen Kommunikation in Multi-Transputersystemen darauf basierte.

Das Software-/Hardware-Codesign kommt bei der Simulation und auch bei der Entwicklung immer öfter vor. Hier wird das komplette System vor Beginn der Implementierung in C simuliert. Erst später wird dann entschieden, welcher Teil in Hardware umgesetzt wird und welcher Teil als Software auf einem Prozessor abläuft. Diese Entscheidung kann die Kosten extrem beeinflussen, denn hier geht es um Entscheidungen, abhängig

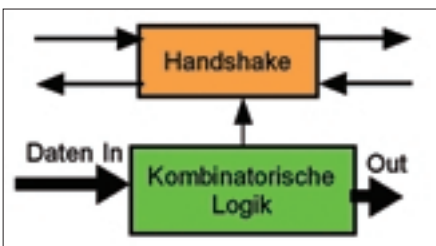


Bild 6. Datenübergabe mit Handshake, aktiv oder passiv

von der Stückzahl wie: Prozessor, Taktfrequenz, Entwicklungszeit, Entwicklungskosten, und von daher kann der parallele Prozess als Hardware oder als Software implementiert werden. Die Software wird dann im Compiler optimiert und die Hardware über die Synthese als Logik. Asynchrone Techniken und Standard-Tools schließen sich

Funktion	Manuell	Synthese	% Synthese
MUX	40	40	100
AND4	66	68	103
Test7	140	126	90
CLIPPER	339	212	63
Set_Cnt	238	208	87
AND16	352	342	99
SHIFT	506	248	56
CASE	594	482	81
Synch-State	1008	814	81
Bit_Cnt	1059	794	75

Tabelle 3. Vergleich der Transistoren bei manuellem Design und Synthese mit Design-Compiler, basierend auf der LSI LCB500k Bibliothek

durchaus nicht aus. Tabelle 2 zeigt anhand einiger Beispiele, wie man den Aufwand und Transistoren für die Implementierung in synchroner Logik oder in NCL-Logik vergleichen kann. Es ergibt sich ein Verhältnis von

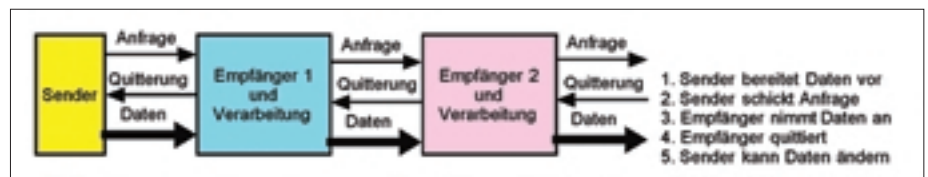


Bild 7. Mikro-Pipeline

0,7 bis 2,1. Hier muss beispielsweise abgeklärt werden, wie die Verkürzung der Entwicklungszeit einzuschätzen ist. Tabelle 3 gibt einen Überblick bezüglich der Zahlen bei Layout mit der Hand und Einsatz des Designcompilers von Synopsys. Auf Basis der Motorola-Architektur HC05/HC08/ STAR08 wurde von Theseus der Core in NCL entwickelt. Zusammen mit 32 KFlash und 4 KRAM und Standard-Peripheriefunktionen wird ein kompletter Mikrocontroller angeboten. Einige Kerndaten sind in Tabelle 4 angegeben.

Die Frage: Einsatz von asynchroner Logik ja oder nein? wird sich in Zukunft wohl so

nicht mehr stellen. Da die Anzahl der Gatter weiter steigt und speziell bei mobilen Consumerprodukten die Leistungsanforderungen steigen, ist dies nur durch mehr Gatter und höhere Taktfrequenzen zu erreichen. Umgekehrt soll aber der Stromverbrauch so gering wie möglich sein. Man kann zwar bestimmte Blöcke im synchronen System zeitweise abschalten, um Strom zu sparen, aber damit ergibt sich auch ein Zusatzaufwand, um den korrekten Zeitpunkt des Wiederanschaltens zu bestimmen. Ein weiterer Aspekt in diesem Zusammenhang ist IP. Das Problem beim Einsatz von Intellectual Property, also fertig entwickelte Funktionsblöcke von Drittanbietern, besteht darin, dass alle Module im System zuverlässig zusammenarbeiten müssen. Bei einem synchronen Design muss die Clock alle IPs ansprechen. Geht man hier zu einem asynchronen Interface über, ist jeder Block für

sich selbst verantwortlich, und es kann beim Design viel Zeit gespart werden.

Die Frage ist wohl so besser gestellt: Wieviel asynchrone Funktionen sind auf dem Chip sinnvoll und wann? Wichtig für den Entwickler ist die Aussage, dass es bereits jetzt Mikroprozessoren auf dieser Basis gibt wie zum Beispiel mit Motorola-Core auf NCL-Basis von Theseus, und dass die Entwicklung mit Standardwerkzeugen von Synopsys durchgeführt werden kann. Weitere Cores, auch von anderen Herstellern, sind in der Entwicklungsphase.

(Jürgen Pintaske, Exemark/pa)

Die Links zu den Firmen:

- NCL-Logic: Theseus.com/
- Synthese: Synopsys.com/
- Synopsys und Theseus: http://www.synopsys.com/products/success/theseus_ss_A4.pdf
- NCL Prozessor: <http://www.theseus.com/FramesProducts.htm>
- Der AMULET-Prozessor: www.cs.man.ac.uk/amulet/
- Asynchrone Logik Homepage: www.cs.man.ac.uk/async/
- Weitere Infos: www.exemark.com