FIT-IT Projekt DARTS: Dezentrale fehlertolerante Taktgenerierung

Ulrich Schmid, Andreas Steininger
{s,steininger}@ecs.tuwien.ac.at

Technische Universität Wien
Institut für Technische Informatik
Embedded Computing Systems Group (E182/2)
Treitlstraße 3, A-1040 Wien

Manfred Sust
manfred.sust@space.at

Austrian Aerospace GmbH
Stachegasse 16
A-1120 Wien

Kurzfassung: Dieser Artikel beschreibt Ziele und Lösungsansatz des prämierten FIT-IT „Embedded Systems“ Projekts „Distributed Algorithms for Robust Tick Synchronization“ (DARTS), das der Entwicklung einer fehlertoleranten Alternative zur herkömmlichen Taktgenerierung in VLSI Chips, Systems-on-Chip und anderen Hardware-Systemen gewidmet ist. Anstatt das Taktsignal eines einzelnen Oszillators mit Hilfe eines aufwändig balancierten Clock-Trees systemweit zu verteilen, wird in DARTS der Takt jeder Funktionseinheit mit Hilfe eines lokalen Taktgenerators erzeugt. Damit wird der „Single Point of Failure“ eliminiert, den ein zentraler Oszillator zwangsläufig darstellt. Im Gegensatz zum klassischen „Globally Asynchronous Locally Synchronous“ (GALS) - Ansatz sind die lokalen Taktgeneratoren beim DARTS-Ansatz jedoch keine Oszillatoren, sondern Instanzen eines verteilten Algorithmus zur fehlertoleranten Generierung *synchronisierter* lokaler Taktesignale.

1. Motivation

Die Entwicklung digitaler VLSI-Chips ist durch ständig wachsende Integrationsdichte, Komplexität und Taktfrequenz gekennzeichnet [ITRS05]. Die Bereitstellung des nach dem synchronen Design-Paradigma erforderlichen phasensynchronen Taktes in allen Funktionseinheiten eines Chips wird dadurch aus den folgenden Gründen immer schwieriger:

- Aufgrund ihrer begrenzten Ausbreitungsgeschwindigkeit brauchen Signale bei hohen Taktraten bereits eine ganze Taktperiode (und mehr), um zu allen Funktionseinheiten am Chip zu propagieren.
- Immer aufwändigere Verfahren zur Skew-Optimierung des für die Taktverteilung verantwortlichen *Clock-Trees* (siehe Abb. 1) werden erforderlich [Restle01][Friedman01] [Restle02][Intel02] .
- Clock-Tree und Buffers konsumieren viel Fläche und Leistung [NCF02]. Darüber hinaus verursachen sie signifikante leitungsgebundene Störungen, vor allem in der Spannungsversorgung, sowie auch Störabstrahlung [MA01].
- Die maximale Taktfrequenz kann erst sehr spät im Entwurfsprozess mit Hilfe einer Worst-Case Analyse bestimmt werden. Aufgrund von Prozessstreuung und der Abhängigkeit von Betriebsparametern müssen signifikante Timing-Reserven vorgesehen werden.
- Quarze sind teuer, groß und mechanisch empfindlich.

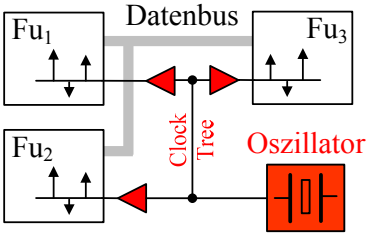

Abb. 1. Standard Clock-Tree

Das Taktsignal ist ein klassischer “Single Point of Failure”, sodass jeder Fehler im Taktsignal, auch wenn es sich nur um einen transienten Fehler handelt, in der Regel zu einer Fehlfunktion des gesamten Chips führt. Dieser Aspekt war bisher insbesondere für sicherheitskritische Bereiche von Belang, gewinnt aber generell an Bedeutung, da mit fortschreitender Verringerung der Strukturbreiten ein drastischer Anstieg der Fehlerraten erwartet wird

[Const02] **Fehler! Verweisquelle konnte nicht gefunden werden.**.

Existierende Techniken zur fehlertoleranten Taktgenerierung setzen *n* verteilte lokale Taktquellen voraus, aus denen mittels „Voting“ ein globaler fehlerfreier Takt ausgewählt wird [VanAlen91]][[Choi90]][[Vasan98]][[Shin88]. Diese Verfahren erfordern jedoch aufwändige Hardware und können nur bedingt mit Skew umgehen. Sie sind somit für die Taktgenerierung in VLSI-Chips wenig geeignet.

Eine Alternative stellen „Globally Asynchronous Locally Synchronous“ (GALS)-Architekturen [Chapiro84] dar, bei denen lokal begrenzte synchrone „funktionale Inseln“ untereinander asynchron kommunizieren. Problematisch ist hier allerdings die Schnittstelle zwischen den lokalen Clock-Domains [Chelcea00]. Zwar kann der GALS-Ansatz durch die in [Fai04] und [MA03] vorgeschlagenen Verfahren zur Taktgenerierung auch für VLSI-Chips nutzbar gemacht werden, allerdings sind diese Lösungen, die als Systeme lose gekoppelter Ringoszillatoren aufgefasst werden können, nicht fehlertolerant.

Das prämierte FIT-IT Embedded-Systems-Projekt *Distributed Algorithms for Robust Tick Synchronization* (DARTS, Projektnummer 809456), eine Kooperation zwischen der TU-Wien und Austrian Aerospace, ist der Erforschung der Möglichkeiten und Grenzen eines alternativen Lösungsansatzes gewidmet.

2. Das FIT-IT Embedded Systems Projekt DARTS

In klassischen verteilten Systemen ist man seit jeher gewohnt, mit einem nicht vernachlässigbaren Zeitaufwand für die Kommunikation umzugehen. Da mittlerweile, wie oben erwähnt, auch in VLSI-Chips die Leitungsverzögerungen eine dominante Rolle spielen, liegt es nahe, Ideen und Lösungen aus diesem gut erforschten Bereich auf den Bereich des Chip-Designs zu übertragen. Konkret soll der GALS-Ansatz mit einer aus dem Bereich der fehlertoleranten verteilten Systeme stammenden alternativen Methode zur lokalen Taktgenerierung kombiniert werden, und zwar mit dem Konzept der *fehlertoleranten Uhrensynchronisation* [Ram90].

Der zugrunde liegende, patentierte Ansatz [SS04] (siehe Abb. 2) beruht darauf, jeder Funktionseinheit[1] eine eigene lokale Taktgenerierungseinheit, *TG-Alg* genannt, zuzuordnen, die den jeweiligen lokalen Takt liefert. Die TG-Algs müssen taktfrei, mittels asynchroner Hardware, realisiert werden. Alle TG-Algs sind identisch aufgebaut („Instanzen" des gleichen Algorithmus) und miteinander über ein besonders einfaches Kommunikationsnetz (*TG-Net*) verbunden, welches die TG-Algs befähigt, ihre jeweiligen lokalen Taktsignale innerhalb einer bestimmten Genauigkeit zu synchronisieren.

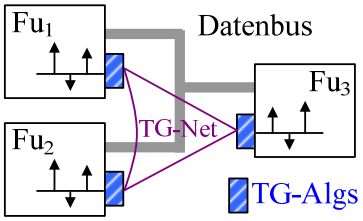

Abb. 2. DARTS-Ansatz

Die wichtigsten Ziele des 3-jährigen DARTS-Projekts sind:

- Design von TG-Algs und TG-Net, inklusive Korrektheitsbeweis und theoretischer Performance-Analyse.
- TG-Alg-Implementierung mittels asynchroner Logik in einem Application Specific Integrated Circuit (ASIC).
- Quantitative Bewertung der wichtigsten Eigenschaften des DARTS-Ansatzes durch experimentelle Evaluation in einer Demonstrator-Anwendung.

[1] Gilt auch für Gruppen von Funktionseinheiten, wenn wir diese Gruppe als eigene Funktionseinheit betrachten

3. Funktionsprinzip

Die TG-Algs basieren auf dem Algorithmus für „Consistent Broadcasting" (CB) von Srikanth und Toueg, der auch das Herzstück des optimalen Uhrensynchronisationsalgorithmus in [Srikanth87] darstellt. Die in Abb. 3 dargestellte Variante des CB-Algorithmus stammt aus [Widder03] .

```
0: VAR k: integer := 0;
1: initially send tick(0) to all [once];
2: /*     Relay Rule          */
3: if received tick(m) with m ≥ k from at least f + 1 instances
4: send tick(k), . . . , tick(m) to all [once]; k := m;
5: /*     Increment Rule      */
6: if received tick(k) from at least 2f + 1 instances
7: send tick(k + 1) to all [once]; k := k + 1;
```

Abb. 3. Consistent Broadcasting Algorithmus

Jeder TG-Alg generiert aufeinander folgende Flanken (= das zu generierende lokale Taktsignal), welche die Messages *tick(k)* $k = 1, 2, 3, \ldots$ des Algorithmus repräsentieren. Der Parameter k ist die aktuelle Tick-Nummer. In einem System mit $n \geq 3f + 1$ Instanzen des TG-Alg toleriert der Algorithmus bis zu f Instanzen, die beliebiges Fehlverhalten, so genannte Byzantinische Fehler, aufweisen können. Fehler im TG-Net sind hierbei eingeschlossen.
Das Funktionsprinzip des Algorithmus ist sehr einfach:
(1) Wenn eine Instanz *tick(m)* von f+1 verschiedenen Instanzen empfangen hat (Zeile 3), kann sie sicher sein, dass mindestens eine korrekte (d.h. fehlerfrei funktionierende) Instanz darunter ist. Sie sendet somit ihrerseits *tick(m)* und alle noch ausständigen früheren Tick-Messages (Zeile 4), sofern dies noch nicht erfolgt ist.
(2) Wenn nun mindestens eine korrekte Instanz TG_i *tick(k)* von 2f+1 Instanzen empfangen hat, kann sie selbst im Falle von f Fehlern noch sicher sein, dass alle korrekten Instanzen zumindest f+1 Messages *tick(k)* empfangen. Aufgrund von Regel (1) werden diese schließlich selbst *tick(k)* senden. Dies führt zwangsläufig dazu, dass jede korrekte Instanz demnächst *tick(k*+1*)* generieren wird (Zeile 6). TG_i darf daher ihrerseits *tick(k*+1*)* generieren, ohne die geforderte Synchronität zu verletzen: Die Zeitpunkte, zu denen zwei verschiedene Instanzen des Algorithmus *tick(k)* senden, liegen garantiert innerhalb eines von den Message-Delays abhängigen und somit beschränkten Zeitintervalls. Aus diesem Faktum lässt sich, durch eine detaillierte worst-case Analyse [FSFK06], schlussendlich die Synchronisationsgenauigkeit der lokalen Taktsignale der TG-Algs als Funktion der Routing- und Gatter-Delays ableiten.

Trotz seines einfachen Prinzips erweist sich der Algorithmus in Abb. 3 jedoch für die Taktgenerierung in Hardware als ungeeignet. Die Kernprobleme, die in der in [FSFK06] detailliert beschriebenen Adaptierung des Algorithmus zu lösen waren, sind:

- *Anonyme Taktflanken statt tick(k)-Messages:* Angesichts der für fehlertolerante Taktgenerierung notwendigen vollständigen Punkt-zu-Punkt-Verbindung im TG-Net [DHS86] ist die Minimierung dieser einzelnen Kanäle essenziell, um den Gesamtaufwand für die Kommunikation innerhalb akzeptabler Grenzen zu halten. Pro TG-Alg wird im TG-Net daher nur jeweils eine einzelne Broacast-Signalleitung vorgesehen, die alle anderen TG-Algs mit den lokal generierten ticks versorgt. Der sendende TG-Alg kann daher die in *tick(k)* enthaltene Tick-Nummer *k* nicht als Inhalt einer Message schicken, sondern nur jeweils eine einzelne Flanke übertragen, mittels derer am jeweiligen Empfänger der Wert von *k* ermittelt wird.

- *Fehlende Synchronität der Operationen:* Der Algorithmus in Abb. 3 setzt voraus, dass alle elementaren Operationen hintereinander ausgeführt werden ([once] in den Zeilen 4 und 7). Dies ist in asynchroner digitaler Hardware nicht a priori der Fall, sondern muss durch gezielte Maßnahmen sichergestellt werden. Die TG-Algs erzeugen dazu steigende und fallende Flanken des Taktsignals mittels dedizierter und wechselseitig verriegelter Funktionsblöcke: Wenn die nächste zu generierende Taktflanke z.B. eine steigende ist, sind die für die Generierung der fallenden Taktflanke zuständigen Funktionsblöcke deaktiviert und umgekehrt („implizites“ Handshaking).

- *Fehlertoleranz/Threshold-Gates*: Anders als in der klassischen, flankengesteuerten asynchronen Logik („Transition Signaling“ [Myers01]) gibt es im Algorithmus von Abb. 3 „unvollständig“ geschlossene Rückkopplungsschleifen: Die Threshold-Regeln (Zeile 3 und Zeile 6 in Abb. 3) erzeugen eine Message, *bevor* an allen *n*-1 Kanälen die erwarteten zugehörigen Eingangs-Messages eingetroffen sind. Infolge von Fehlern könnten ja bis zu *f* davon ausbleiben. Die TG-Algs können daher nicht vollständig in flankengesteuerter Logik beschrieben werden, sondern müssen als Kombination von klassischer Zustandslogik und flankengesteuerter Logik implementiert werden. Nicht zuletzt deshalb erfordert die korrekte Funktionsweise der TG-Algs die Einhaltung einiger Bedingungen bezüglich des Timings elementarer Komponenten („bounded delay conditions“). Diese Bedingungen, die glücklicherweise nicht sonderlich restriktiv sind, verlangen im wesentlichen eine obere Schranke für das Verhältnis der Delays längs gewisser kritischer Pfade.

- *Aufwändige Operationen*: Der Algorithmus in Abb. 3 enthält einige Operationen, die in Hardware relativ hohen Implementierungsaufwand nach sich ziehen. Hier sind insbesondere der Vergleich („$m \geq k$“ in Zeile 3) sowie die Überprüfung der Schwellwerte („at least $f + 1$“ in Zeile 3 bzw. „at least $2f + 1$“ in Zeile 6) samt zugehöriger „Buchführung“ zu nennen. Da diese Funktionen in allen Instanzen repliziert und auch intern z.T. mehrfach benötigt werden, schlägt der Aufwand für die Implementierung derselben sehr stark auf den Gesamtaufwand durch.

Abb. 4 zeigt den aus diesen Anforderungen resultierenden prinzipiellen Aufbau eines TG-Algs. Das von einem TG-Alg erzeugte lokale Taktsignal wird nicht nur an die zu versorgende Funktionseinheit geliefert, sondern über das, als paralleler Bus von *n* „Single Wire“ Punkt-zu-Punkt-Verbindungen ausgeführte, TG-Net auch an alle anderen TG-Algs verteilt, wo es einen der „Remote Clocks“ darstellt.

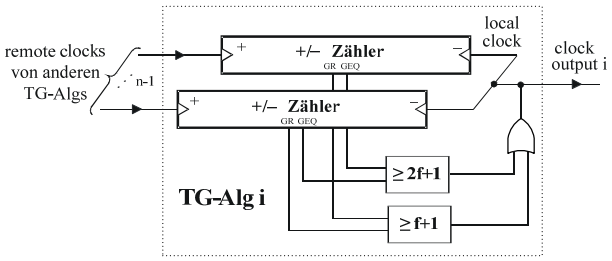

Abb. 4. Eine Instanz eines TG-Alg

Herzstück jedes TG-Algs sind die +/- *Zähler*, von welchen je einer pro eingehendem Remote Clock zur Verfügung gestellt werden muss. Diese verwalten den aktuellen Wert von *m* für jeden der eingehenden Remote Clocks, wobei zur Vereinfachung ausgenutzt wird, dass für den adaptierten Algorithmus nicht der absolute Wert von *m* benötigt wird, sondern nur die jeweilige Differenz von *m* zur Anzahl der vom betrachteten TG-Alg bereits selbst ausgegebenen Ticks (entspricht dem Zählerstand *k* für den „Local Clock“). Die aus dieser Differenz abgeleiteten Status-Signale GR und GEQ, die Zählerstand > 0 und ≥ 0 repräsentieren, werden von den Threshold-Gates, entsprechend den Regeln in Zeile 3 und Zeile 6 des Algorithmus (Abb. 3), ausgewertet und zur Generierung des nächsten Ticks verwendet.
Die detaillierte Beschreibung der TG-Algs und die formal-mathematische Korrektheits- und Performance-Analyse, die unter anderem eine Formel für die erreichbare worst-case Synchronisationsgenauigkeit liefert, sind in [FSFK06] zu finden. Interessanterweise hängt die Synchronisationsgenauigkeit $\pi = \pi(\Theta)$ des verwendeten Verfahrens lediglich vom <u>*Verhältnis*</u> Θ zwischen langsamstem und schnellstem End-zu-End-Pfad (TG-Alg – TG-Net – TG-Alg) ab, vgl. [LeLann03] , und ist somit „quasi-delay-insensitive“ [Myers01]. Θ ist übrigens auch der einzige implementierungsabhängige Parameter, der im VHDL-Code der TG-Algs auftaucht und konkret die Anzahl der Stufen der +/- Counter bestimmt.

4. Implementierung

Da die TG-Algs den Systemtakt erzeugen sollen, müssen sie selbst mittels asynchroner logischer Schaltkreise realisiert werden. Eine besondere Herausforderung stellt die saubere und effiziente Implementierung der +/- Zähler dar. Um keine Flanke zu „verlieren“, werden alle eingehenden Takte (remote und lokal) über einen Puffer geführt, der in Form einer *elastischen Pipeline* [Sutherland89] realisiert ist. Eine elastische Pipeline (Abb. 5) kann als First-In-First-Out (FIFO) Puffer für Signalflanken aufgefasst werden und besteht aus einer Kette von *Muller C-Elementen*, die ihrerseits AND-Gates für Signalflanken realisieren. Jede Stufe in der Pipeline kann eine Flanke von *data_in* speichern, bis sie via *data_out/ack_in* am Ende der Pipeline ausgelesen wird.

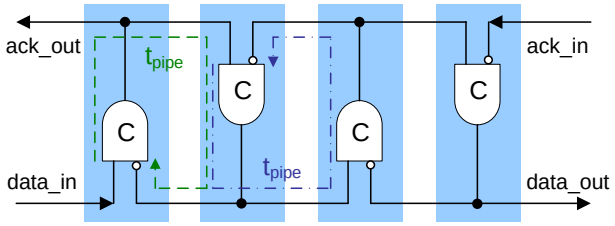

Abb. 5. Elastic Pipeline

Zur Realisierung eines +/- Zählers werden zwei elastische Pipelines verwendet, eine für Ticks vom jeweiligen Remote Clock und die andere für die lokalen Ticks. Ein dazwischen sitzendes *Differenz-Modul* (Diff-Module) hat die Aufgabe, in beiden Pipelines vorhandene, korrespondierende, Flanken auszulesen und somit zu entfernen, da diese keinen Einfluss auf die Differenz der Anzahl der Ticks haben. Das Differenz-Modul ist als asynchrone State-Machine realisiert. Eine Zustandslogik erzeugt in der Folge die Status-Signale GR und GEQ als Input für die Threshold Module. Diese führen wieder eine Konversion von Zustandslogik in flankengesteuerte Logik durch, weshalb bei ihrer Implementierung besondere Maßnahmen getroffen werden müssen, um Glitch-Freiheit zu gewährleisten.
Abb. 6 zeigt das resultierende Blockschaltbild eines TG-Algs. Für Details der Implementierung sei auf [FFSK06] verwiesen.

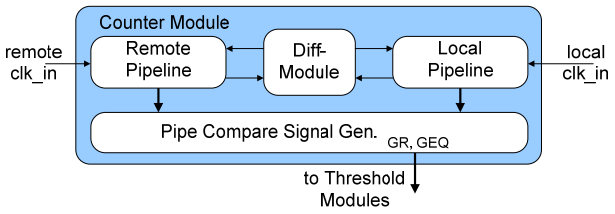

Abb. 6. Implementierung eines +/- Zählers

Eine VHDL-Implementierung von *n*=5 TG-Algs wurde zunächst für ein Altera APEX EP20K1000 FPGA synthetisiert. Abb. 7 zeigt eine Momentaufnahme der generierten Taktsignale; die Taktfrequenz beträgt ca. 24 MHz, der maximale Skew ist 4 ns. Die relativ geringe Taktfrequenz ist primär auf die schlechte Abbildbarkeit asynchroner Logik auf die im FPGA vorhandenen Ressourcen zurückzuführen. Erste Simulationen lassen erwarten, dass die erreichbare Taktfrequenz in einer ASIC-Implementierung um ein Vielfaches größer ist.

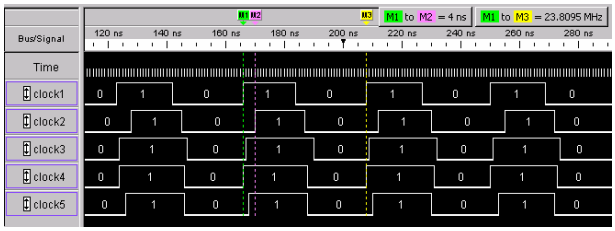

Abb. 7. Messergebnis: Taktgenerierung in FPGA

Darüber hinaus werden im Rahmen von DARTS auch alternative TG-Algs untersucht, die eine deutliche Steigerung der Taktfrequenz, der Frequenzstabilität und der Toleranz bezüglich transienter Fehler bieten.

5. DARTS Demonstrator-Anwendung

Wie bereits erwähnt werden im Rahmen von DARTS nicht nur eine ASIC-Implementierung der beschriebenen TG-Algs, sondern auch eine geeignete Demonstrator-Anwendung, sowie eine Testumgebung entwickelt. Anhand dieser prototypischen Implementierung sollen die wichtigsten Eigenschaften des DARTS-Ansatzes für praktische Anwendungen experimentell untersucht werden: Leistungsverbrauch, Robustheit, Fehlertoleranz, Aufwand/Kosten, erreichbare Taktfrequenz, Stabilität des Taktes bzw. dessen Abhängigkeit von Betriebs- und Prozessparametern usw. Die Testumgebung stellt alle hierfür erforderlichen Möglichkeiten, wie z.B. Fehlerinjektion, Messpunkte und Stecker, bereit. Aus den Messergebnissen sollen Ansätze für die weitere Verbesserung der zu Grunde liegenden Konzepte und der praktischen Implementierung abgeleitet werden.

Entscheidende Bedeutung kommt dabei natürlich der Wahl einer sowohl technisch als auch wirtschaftlich relevanten Demonstrator-Anwendung zu.

Im Vergleich zu herkömmlichen zentralen Taktquellen liegt der entscheidende Vorteil des DARTS-Ansatzes in der inhärenten Fehlertoleranz, die besonders für hoch integrierte Digitalschaltungen für den Weltraum-Einsatz von Interesse ist. Im Weltraum sind VLSI-Chips der Belastung durch verschiedene Arten ionisierender Strahlung ausgesetzt. Einerseits ruft die über einen bestimmten Zeitraum aus der Strahlung aufgenommene Energie als Total-Dose (TD) Effekte bezeichnete, bleibende Schäden hervor. Andererseits verursachen energiereiche Ionen und Protonen Single-Event-Effekte (SEE) in Form von Glitches, Spikes oder Zustandsänderungen in Speicherelementen, so genannte Single-Event-Upsets (SEUs)..

In diesem Zusammenhang liegen Aerospace-Applikationen etwas unglücklich zwischen industrieller Elektronik, mit vernachlässigbaren Strahlungs-anforderungen, und militärischen Anwendungen, die durch Verwendung spezieller Technologien für extreme Strahlenbelastungen „gehärtet" werden müssen. Letztere sind selbstverständlich für den Einsatz im Weltraum geeignet, aber leider nicht mit den wirtschaftlichen Randbedingungen vereinbar, die von den dahinter stehenden Märkten (z. B. Telekom, Meteorologie) oder wissenschaftliche Anwendungen (z. B. Astronomie) gefordert werden.

Interessante Alternativen resultieren aus der fort-schreitenden Miniaturisierung der Halbleiterstrukturen, wodurch manche TD-Effekte dermaßen reduziert werden, dass es sich lohnt der verbleibenden TD-Empfindlichkeit mit speziellen Design-Methoden auf Transistor-Ebene zu begegnen [Red04]. Die benötigte SEU-Härte kann dann über den VHDL-Code, durch Einfügen von Triple-Mode-Redundancy (TMR) auf Register- oder Funktions-Ebene, oder gar durch das Triplizieren von VLSI-Chips, eingestellt werden. Weniger geeignet ist TMR in den Bereichen Takterzeugung und Taktverteilung (Clock-Tree). Die somit verbleibende Lücke hoffen wir mit dem DARTS-Ansatz vollständig zu schließen.

Eine für Weltraum-Anwendungen interessante rekonfigurierbare TMR-Architektur basiert auf der Verwendung mehrerer parallel arbeitender Field-Programmable Gate Arrays (FPGAs). Betrachten wir hierzu das Problem der Verarbeitung eines Eingangs-Datenstroms zu einem Ausgangs-Datenstrom, wobei die Verarbeitungsoperationen außer von den Eingangs-Werten auch von internen Zuständen abhängen, wie dies z.B. bei einem Kalman-Filter der Fall ist. Wenn nun 3 FPGAs dieselben Operationen parallel ausführen und ein strahlungsunempfindlicher Voter deren Ergebnisse vergleicht, wird ein in einem einzelnen FPGA auftretender SEU transparent wegmaskiert. Der Voter stellt sicher, dass nur jenes Ergebnis weiterverwendet wird, das von mindestens 2 FPGAs übereinstimmend erzielt wurde. Das von dem SEU betroffene FPGA kann mittels geeigneter Reinitialisierungsmaßnahmen wieder auf den korrekten internen Zustand gebracht werden.

Die als DARTS-Demonstrator verwendete Lösungs-alternative vermeidet den strahlungstoleranten Voter[2]. Seine Aufgabe wird von einem verteilten fehler-toleranten Byzantine-Agreement-Algorithmus über-nommen, welcher gewährleistet, dass trotz eventuell divergierender Inputs alle Outputs denselben Wert haben. Wie in Abb. 8 gezeigt, wird der Agreement-Algorithmus verwendet, um sicherzustellen, dass die Rückkopplungspfade fehlerfreier FPGAs nicht von einem fehlerhaften Ausgang eines von einem SEU betroffenen FPGAs beeinflusst werden können. Optional kann ein weiterer Agreement-Algorithmus dazu verwendet werden, um die Daten redundanter Sensoren in Übereinstimmung zu bringen.

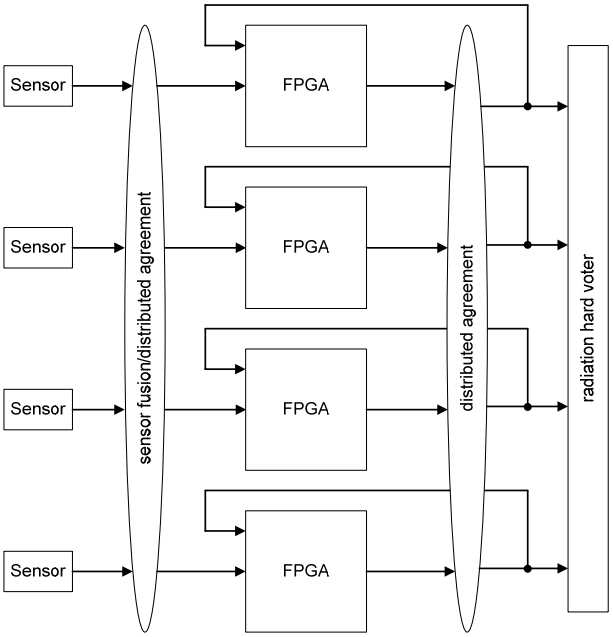

Abb. 8. Demonstrator-Anwendung

Die besondere Attraktivität der Kombination dieses Lösungsansatzes mit DARTS besteht in der „durchgängigen“ Fehlertoleranz, also der Robustheit auch gegenüber SEUs, die die Taktgenerierung betreffen, wie sie im Falle einer zentralen Taktgenerierung niemals gewährleistet werden kann.

6. Zusammenfassung

Diese Arbeit beschreibt eine im Rahmen des FIT-IT DARTS-Projektes entwickelte Alternative für die fehlertolerante Taktgenerierung in VLSI Chips. Die Vorteile gegenüber herkömmlicher Takterzeugung und Taktverteilung sind folgende:

- Das Verfahren toleriert (beweisbar korrekt) eine parametrisierbare Anzahl transienter und permanenter Fehler im TG-Net und in den TG-Algs.
- Das synchrone Paradigma kann nicht nur beim Design der lokalen Funktionseinheiten (basierend auf den lokalen Ticks), sondern auch bei der globalen Kommunikation (via heruntergeteilter Macro-Ticks) zwischen weit entfernten Funktionseinheiten beibehalten werden.
- Der Chip ist selbsttaktend, braucht also keinen externen Oszillator. Eine Temperaturkompensation kann zur Verbesserung der Frequenzstabilität zusätzlich realisiert werden.

[2] Genauer gesagt wird ein solcher Voter nur dann benötigt, wenn aus irgendwelchen anderen Gründen (etwa zwecks Ansteuerung eines nicht-redundanten Aktuators) ein einzelnes, korrektes Ergebnis aus den 4 parallelen Zweigen ausgewählt werden muß.

- Der Chip läuft automatisch mit der höchsten, unter den jeweiligen Betriebsbedingungen möglichen Frequenz. Das Timing beruht nicht, wie im synchronen Fall, auf worst-case Annahmen und ist äußerst robust.
- Die aufwändige Sonderbehandlung des Clock-Trees entfällt. Das Routing des TG-Net ist relativ unkritisch.
- Die Anzahl und Plazierung der TG-Algs kann je nach den Erfordernissen optimal gewählt werden, wodurch Performance (Taktfrequenz, Fehlertoleranz) und Overhead (Fläche, Leistungsaufnahme) kontrolliert werden können.
- Nicht synchrones Schalten der Funktionseinheiten verringert die Störabstrahlung und vermindert leitungsgebundene Störungen in der Spannungsversorgung.

Nähere Informationen zu DARTS sind unter http://ti.tuwien.ac.at/darts zu finden

REFERENZEN

[Widder03] Josef Widder. Booting clock synchronization in partially synchronous systems. In *Proceedings of the 17th International Symposium on Distributed Computing (DISC'03)*, vol. 2848 of LNCS, pp. 121-135. Springer 2003.

[LeLann03] Gerard Le Lann and Ulrich Schmid. How to implement a timer-free perfect failure detector in partially synchronous systems. *Technical Report* 183/1-127, Department of Automation, Technische Universität Wien, April 2003

[Intel02] Ram Bhamidipati, Ahmad Zaidi, Siva Makineni, Kah K. Low, Robert Chen, Kin-Yip Liu and Jack Dalgrehn. Challenges and Methodologies for Implementing High-Performance Network Processors. *Intel Technology Journal* Vol. 06, issue 03, August 15, 2002. Intel Corporation.

[Choi90] Bong-Rak Choi, Kyu Ho Park and Myunghwan Kim. An Improved Hardware Implementation of the Fault-Tolerant Clock Synchronization Algorithm for Large Multiprocessor Systems. *IEEE Transactions on Computers* Vol. 39, No. 3, March 1990, pp. 404-407. IEEE Computer Society Press.

[VanAlen91] D.J. VanAlen and A.K. Somani. An All Digital Phase Locked Loop Fault Tolerant Clock. *Proc. International Symposium on Circuits and Systems*, Vol. 05, June 1991, pp. 3170-3173. IEEE Computer Society Press.

[Vasan98] Nagesh Vasanthavada and Peter N. Marinos. Synchronization of Fault-Tolerant Clocks in the Presence of Malicious Failures. *IEEE Transactions on Computers* Vol. 37, No. 4, April 1988, pp. 440-448. IEEE Computer Society Press.

[Restle01] Phillip J. Restle, et. al. A Clock Distribution Network for Microprocessors. *IEEE Journal of Solid-State Circuits* Vol. 36, No. 5, May 2001, pp. 792 - 799. IEEE Computer Society Press.

[Srikanth87] T.K. Srikanth and S. Toueg. Optimal clock synchronization. *Journal of the ACM*. Vol. 34, no. 3, pp.626-645, July. 1987. ACM Press.

[Shin88] K.G. Shin and P. Ramanathan. Transmission Delays in Hardware Clock Synchronization. *IEEE Transactions on Computers*. Vol. C-37, No. 11, pp. 1465-1467, Nov. 1988. IEEE Computer Society Press.

[Chapiro84] Daniel M. Chapiro. Globally-Asynchronous Locally-Synchronous Systems. *Ph.D. thesis*, Stanford University, Oct. 1984.

[Myers01] Chris J. Myers. Asynchronous Circuit Design. John Wiley and Sons, July 2001

[Fai04] Scott Fairbanks. Method and apparatus for a distributed clock generator, *US patent no. US2004108876*,

[Const02] Cristian Constantinescu. Impact of Deep Submicon Technology on Dependability of VLSI Circuits. *Proceedings of the International Conference on Dependable Systems and Networks (DSN) 2002*. pp. 205-209, June 2002. IEEE Computer Society Press.

[ITRS05] International Technology Roadmap for Semiconductors 2005 (http://public.itrs.net)

[Sutherland89] I.E. Sutherland. Micropipelines, *Communications of the ACM*, Vol. 32, No. 6, Jun. 1989, pp. 720-738. ACM Press

[Restle02] J. Phillip Restle, et al. The clock distribution of the Power4 microprocessor. *Proc. IEEE International Solid-State Circuits Conference, 2002. ISSCC 2002*, pp.144 – 145.

[Friedman01] Eby G. Friedman. Clock Distribution Networks in Synchronous Digital Integrated Circuits. *Proceedings of the IEEE*. Vol. 89, No. 5, pp. 665 – 692,May 2001.

[Ram90] P. Ramanathan, K. G. Shin and R. W. Butler. Fault-Tolerant Clock Synchronization in Distributed Systems. *Computer*, Vol. 23, No. 10, pp. 30 – 42, Oct. 1990. IEEE Computer Society Press.

[Chelcea00] Tiberiu Chelcea and Steven M. Novick. A Low-Latency FIFO for Mixed-Clock Systems. *IEEE Computer Society Workshop on VLSI* , April 2000, pp. 119 – 126. IEEE Computer Society Press.

[Hauck95] S. Hauck. Asynchronous Design Methodologies: An Overview. *Proceedings of the IEEE*, Vol. 83, pp. 69 – 93, 1995.

[DHS86] Danny Dolev, Joseph Y. Halpern and H. Raymond Strong. On the Possibility and Impossibility of Achieving Clock Synchronization". *Journal of Computer and System Sciences*. Vol. 32, pp.230 250, 1986.

[FSFK06] Matthias Függer, Ulrich Schmid, Gottfried Fuchs and Georg Kempf. Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip. *Proceedings European Distributed Computing Conference (EDCC'06),* 2006, pp. 87-96. IEEE Computer Society Press.

[MA01] M. S. Maza and M.L. Aranda. Analysis of clock distribution networks in the presence of crosstalk and groundbounce. *Proceedings International IEEE Conference on Electronics, Circuits, and Systems (ICECS)*, p. 773-776, 2001.

[MA03] M. S. Maza and M. L. Aranda. Interconnected rings and oscillators as gigahertz clock distribution nets. *Proceedings of the 13th ACM Great Lakes symposium on VLSI (GLSVLSI '03)*, p. 41-44. ACM Press, 2003.

[SS04] U. Schmid and A. Steininger. Dezentrale Fehlertolerante Taktgenerierung in VLSI Chips. Research Report 69/2004, Technische Universität Wien, Institut für Technische Informatik, 2004. (Österr. Patentanmeldung A 1223/2004).

[NCF02] S. D. Naffziger, G. Colon-Bonet, T. Fischer, R. Riedlinger, T. J. Sullivan, and T. Grutkowski, The Implementation of the Itanium 2 Microprocessor, *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, pp. 1448--1460, Nov. 2002.

[FFSK06] M. Ferringer, G. Fuchs, A. Steininger, G. Kempf, VLSI Implementation of a Fault-Tolerant Distributed Clock Generation. *The 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'06)*, October 2006.

[Red04] S. Redant, R. Marec, L. Baguena, E. Liegeon, J. Soucarre, B. Van Thielen, G. Beeckman, P. Ribeiro, A. Fernandez-Leon, B. Glass, "The Design Against Radiation Effects (DARE) Library, " *Proc. READECS 04*, September 2004.