Advances in Radio Science (2003) 1: 253–258

Advances in Radio Science

Konzeption und Entwurf einer auf Petri-Netzen basierenden programmierbaren Kontrollschaltung

H. Baur and H.-J. Pfleiderer

Universität Ulm, Abteilung Allgemeine Elektrotechnik und Mikroelektronik, Albert-Einstein-Allee 43, D-89081 Ulm, Germany

Zusammenfassung. Petri-Netze eignen sich aufgrund der direkten Darstellbarkeit von Nebenläufigkeiten und leistungsfähiger Analyseverfahren zur Beschreibung von komplexen Kontrollabläufen. Die hier vorgestellte PNDU (Petri Net Decision Unit) ist ein programmierbarer Hardware-Interpreter für Kontrollalgorithmen, die durch Petri-Netze beschrieben werden. Die Petri-Netze werden kodiert in einem Speicher abgelegt und unter Einbeziehung der Eingangssignale und interner Zustände nach einem festgelegten Algorithmus abgearbeitet. Der implizite Wartezustand der PNDU motiviert in Hinblick auf den Einsatz in eingebetteten, batteriebetriebenen Geräten eine asynchrone Realisierung. Die PNDU wurde mit VHDL als synchrone und vergleichend dazu als selbstgetaktete Schaltung entworfen.

1 Einleitung

Eine Vielzahl von elektronischen Systemen stehen in Interaktion mit dem Benutzer oder anderen Systemen. Oft haben sie spezielle Kontrollaufgaben zu erfüllen, zeigen interaktives oder reaktives Verhalten und fallen damit in die Klasse diskreter ereignisgesteuerter Systeme. Diese Systeme werden charakterisiert durch eine diskrete Zustandsmenge und ereignisgesteuerte Zustandsänderungen. Zur Modellierung dieser Systeme eignen sich Petri-Netze aufgrund ihrer einfachen Darstellung von Nebenläufigkeiten und deren Abhängigkeiten. Bei batteriebetriebenen Geräten sind neben niedrigem Leistungsverbrauch auch elektromagnetische Verträglichkeit ein wichtiger Gesichtspunkt bei der Entwicklung.

Ein neuartiges Konzept eines programmierbaren Petri-Netz Controllers wird hier vorgestellt und der Entwurf als synchrone und selbstgetaktete Schaltung beschrieben.

Correspondence to: H. Baur
(helmut.baur@e-technik.uni-ulm.de)

2 Schaltungsimplementierungen von Petri-Netzen

Petri-Netze wurden von Carl Adam Petri (1962) eingeführt und konnten sich durch vielfältige Erweiterungen zu einem sehr leistungsfähigen Werkzeug zur Modellierung und Analyse diskreter ereignisgesteuerter Systeme entwickeln. Das Modell des Systemverhaltens kann mit Petri-Netzen sowohl graphisch als auch mathematisch dargestellt werden. Die statische Netzstruktur kann graphisch durch Plätze, Transitionen und deren Verbindung durch gerichtete Kanten dargestellt werden. Abbildung 1 zeigt diese Grundelemente eines Petri-Netzes und eine beispielhafte Netzstruktur mit vier verschiedenen Verknüpfungsmöglichkeiten zwischen Plätzen und Transitionen. Durch Token werden aktive Plätze symbolisiert und die Verschiebung dieser Token im Netz erfolgt durch das Auslösen/Feuern von Transitionen. Die Dynamik des Netzes wird durch die Feuerungsregel gesteuert. Eine Transition wird nur dann gefeuert/ausgelöst, wenn sie aktiviert ist bzw. Konzession hat. Die Darstellung von Nebenläufigkeit und Synchronisation von Prozessen gelingt durch Petri-Netze sehr übersichtlich, da die Anzahl der Token nicht wie bei endlichen Automaten auf einen einzelnen begrenzt ist. Mathematisch wird die Netzstruktur durch die Netzmatrix beschrieben. Jede Verbindung zwischen einem Platz und einer Transition wird entsprechend ihrer Richtung durch den Eintrag “+1” bzw. “−1” berücksichtigt. Alle anderen Stellen dieser Matrix sind mit dem Wert Null belegt. Durch den Markierungsvektor wird die aktuelle Verteilung der Token beschrieben. Folgemarkierungen können mit Hilfe von Matrixmultiplikationen einfach berechnet werden. Die Netzeigenschaften können aufgrund sehr leistungsfähiger Analyseverfahren untersucht werden (Murata, 1989).

Durch die Erweiterung der Petri-Netze zu höheren Netzklassen wurde es ermöglicht, das Verhalten von komplexen Systemen zu modellieren. Beispielhaft seien hier die Ausweitung der zugelassenen Netzstrukturen, Erhöhung des Kantengewichts und der Kapazität von Plätzen genannt, wodurch mehrere Token gleichzeitig in einem Platz sitzen und verschoben werden können. Die Einführung von komplexen

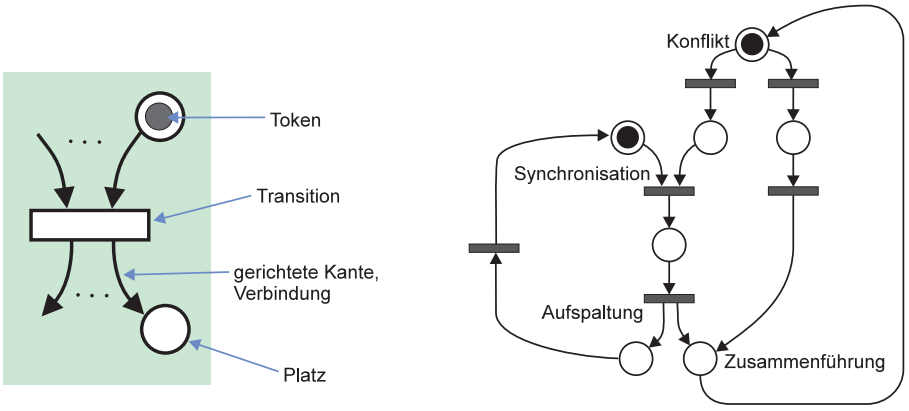

Abbildung 1. Grundelemente von Petri-Netzen und eine beispielhafte Netzstruktur.

Datentypen und die Erweiterung der Feuerungsregel führte zu Farbigen Petri-Netzen (Jensen, 1997).

Werden nun Eingangs- und Ausgangssignale eines Systems in die Feuerungsregel mit einbezogen, so spricht man von interpretierten Petri-Netzen. Für die Überprüfung der Konzession einer Transition bedeutet das neben dem Vorhandensein eines Tokens in jedem Eingangsplatz zusätzlich noch die Überprüfung bestimmter Bedingungen an die Eingangs- und Ausgangssignalwerte des Systems. Durch das Auslösen der Feuerung werden alle Token von den Eingangsplätzen entnommen und je ein Token auf die Ausgangsplätze gesetzt. Außerdem können die Werte der Ausgangssignale verändert werden. Mit dieser Erweiterung sind diese interpretierten Petri-Netze zur Darstellung von Kontrollanwendungen und deren Steuerung geeignet und bilden die Grundlage der Implementierung für steuerungstechnische Anwendungen.

Die erste Abbildung von Petri-Netzen auf eine ausführbare Schaltungsimplementierung wird von Patil (1972) beschrieben. Die Netzstruktur wurde hier direkt in eine Schaltungsstruktur abgebildet, wobei jeder Platz durch ein Flip-Flop realisiert wurde. Die Feuerung der Transitionen wird durch die Setz- und Rücksetzeingänge der Flip-Flops ausgelöst und diese Schaltungsimplementierung funktioniert asynchron. Auch die erste programmierbare Hardware-Abbildung geht auf Patil (1975) zurück und verwendete Programmierbare Logik-Arrays (PLA). Diesen Ansatz mit frei programmierbaren Array-Strukturen verfolgte später auch Hartenstein (1987). Während diese Ansätze auf der direkten Abbildung der Netzstruktur beruhten, wurde mit Hilfe von Mikroprozessoren und Mikrocontrollern (Valette et al., 1983; Murata et al., 1986; Crockett et al., 1987) die Dynamik des Petri-Netzes durch die Berechnung der Matrixgleichungen umgesetzt. Ein Nachteil dieser prozessorbasierten Implementierung ist die indirekte Art der Umsetzung, bei der sehr viele Rechenoperationen (Matrixmultiplikation) durchgeführt werden, insbesondere unnötige Berechnungen mit vielen Null-Werten der Netzmatrix. Dies führte zu einer vergleichsweise langsamen Netzausführung. Ein speziell entwickelter ASIC-Controller (Murakoshi et al., 1990; Kamakura et al., 1997) bildete ebenfalls die komplette Netzstruktur in Hardware ab, erreichte aber eine schnellere Netzausführung durch parallele Umsetzung der Matrixberechnungen. Da die Werte der Netzmatrix in einem Speicher abgelegt wurden, erfordert dieser Ansatz jedoch eine hohe Busbreite dieses Speichers und war zudem auf eine feste Netzgröße ausgelegt.

In dieser Arbeit wird ein neuartiger Ansatz eines programmierbaren Petri-Netz Controllers vorgestellt. Die Netzstruktur ist in einem Speicher mit geringer Wortbreite kodiert abgelegt. Ein spezieller Controller führt lokale Feuerungen der aktivierten Transitionen durch. Die Netzdimension ist nicht a priori festgelegt, wodurch eine effektive Ausführung der Petri-Netze für unterschiedliche Netzgrößen gewährleistet wird.

3 Konzeption der PNDU (Petri Net Decision Unit)

In Abb. 2 ist der prinzipielle Aufbau der PNDU und die Aufteilung in Hardware und Software dargestellt. Die Netzstruktur wird kodiert in einem programmierbaren Speicher abgelegt. In einem speziellen Controller ist der Bearbeitungsalgorithmus nach dem Prinzip der lokalen Feuerung aktivierter Transitionen in Hardware abgebildet. Dabei werden bei der Konzessionsprüfung die Werte der Eingangs- und Ausgangssignale des Systems abgefragt. Bei der Feuerung einer Transition können die Werte der Ausgangssignale individuell verändert werden und optional eine Adresse auf einen Ausgangs-Port geschrieben werden. Intern werden im Controller die Transitionen sequentiell bearbeitet, jedoch werden die neuen Ausgangssignalwerte erst nach Berechnung der Folgemarkierung an die Systemumgebung freigegeben.

Die Netzstruktur ist frei programmierbar und wird kodiert in einem Speicher mit geringer Busbreite abgelegt. Aufgrund der lokalen Feuerung einzelner Transitionen wird das Petri-Netz im Speicher durch einzelne Transitionsblöcke kodiert. Die Transitionsblöcke sind so angeordnet, daß jeweils alle Folgetransitionen eines Platzes hintereinander aufgelistet werden. Abbildung 3 zeigt die Kodierung einer Transition für eine 8-Bit PNDU, die minimale Wahl der parametrisierbaren Konfiguration. Ein Transitionsblock besteht aus 4–6

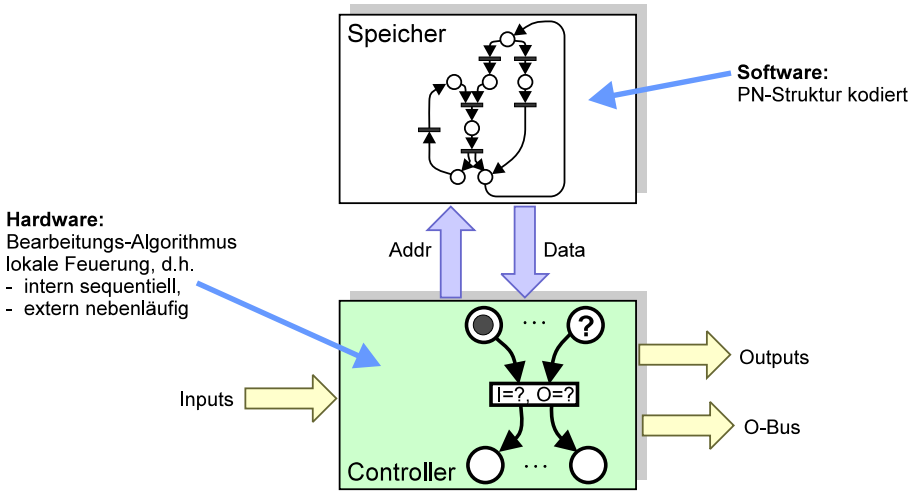

Abbildung 2. Aufbau der PNDU (Petri Net Decision Unit).

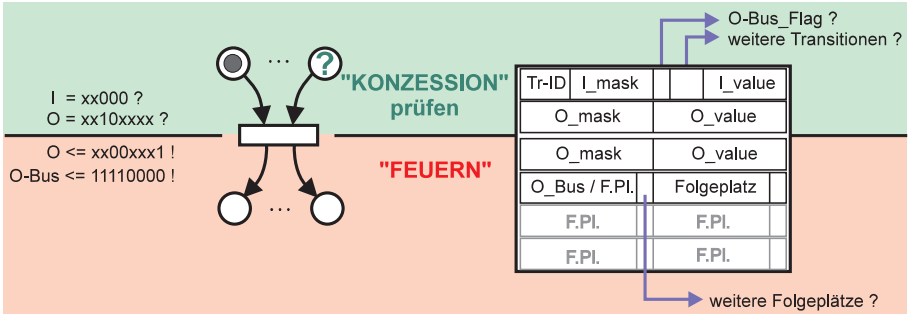

Abbildung 3. Kodierung einer Transition.

einzelnen Datenwörtern. Die ersten beiden Datenwörter enthalten die Informationen zur Prüfung der Konzession der Transition. Bei der Konzessionsprüfung können die Werte aller Ausgangs- und Eingangssignale bitweise auf die Werte “0”, “1” oder “x” (don’t care), dargestellt durch 2 Bit als Maske und Wert, überprüft werden. Weitere Datenbits enthalten Informationen wie eine Identifizierungsnummer der Transition, das Vorhandensein einer optionalen Adresse für den Ausgangs-Port und eine relative Sprungadresse zu weiteren Folgetransitionen. Die restlichen Datenwörter eines Transitionsblocks werden nur bei Feuerung der Transition ausgeführt. Sie enthalten Maske und Wert zur Veränderung der Ausgangssignalwerte, eine optionale Adresse für den Ausgangs-Port und die Speicheradressen der Folgeplätze. Die Anzahl der Folgeplätze ist frei programmierbar (max. 6), der letzte Folgeplatz wird durch ein Indikatorbit erkannt. Synchronisierende Transitionen, das sind Transitionen mit mehr als einem Eingangsplatz, stellen einen Sonderfall dar und werden mit Hilfe der Identifizierungsnummer der Transition (TrID) auf Konzession überprüft, um ein mehrmaliges Feuern zu verhindern.

Die Ausführung der Petri-Netze erfolgt nach einem festgelegten Bearbeitungsalgorithmus im Controller, wie in Abb. 4 dargestellt. Ausgehend von der aktuellen Markierung wird die Folgemarkierung berechnet, d.h. alle Token, die eine Transition konzessionieren, werden durch Feuerung der entsprechenden Transition verschoben. Dem Prinzip der lokalen Feuerung von Transitionen entspricht eine Betrachtung der Netzdynamik aus Sichtweise der Token, die sich in einem Platz aufhalten. Der hierarchisch aufgebaute Bearbeitungsalgorithmus ist aufgeteilt in Platzzyklen und Transitionszyklen. Zur Berechnung der Folgemarkierung werden zunächst alle markierten Plätze betrachtet. Jeder Platz wird dabei als Eingangsplatz seiner Folgetransitionen gesehen, die nacheinander auf Konzession geprüft werden. Konflikte werden durch die Anordnung der Folgetransitionsblöcke im Speicher gelöst, indem die erste aktivierte Transition gefeuert wird und damit die anderen in Konflikt stehenden Folgetransitionen mit niedrigerer Priorität ihre Konzession verlieren. Erst wenn die Folgemarkierung berechnet wurde, werden die neuen Ausgangssignalwerte an die Systemumgebung freigegeben. Dadurch erfolgt der Bearbeitungsalgorithmus zwar intern sequentiell, wird aber extern als nebenläufig wahrgenommen. Unterscheidet sich die Folgemarkierung von der Ausgangsmarkierung, so wird automatisch die nächste Folgemarkierung berechnet. Ist dies nicht der Fall, werden neue Eingangssignalwerte abgefragt und damit ein neuer Berechnungszyklus gestartet. Ein impliziter Wartezustand tritt ein,

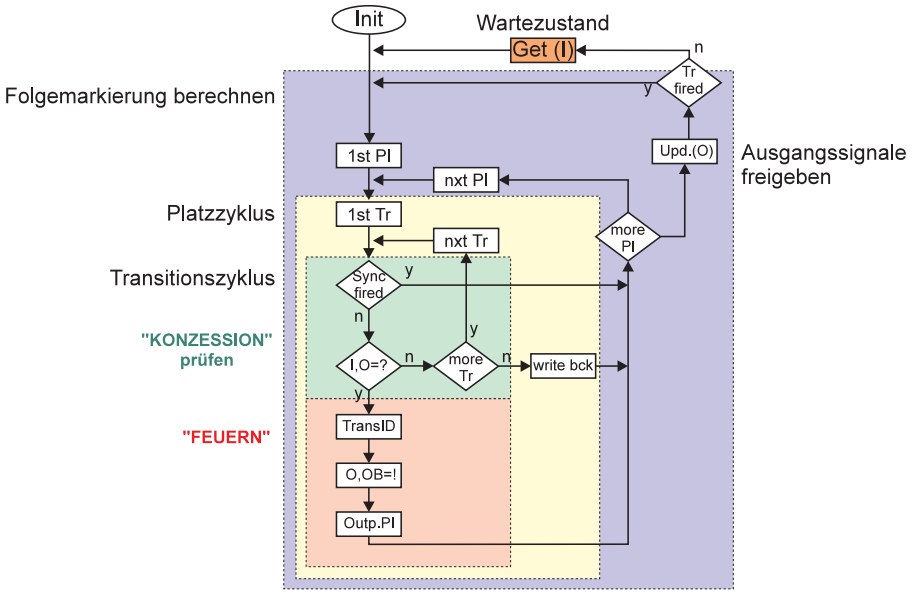

Abbildung 4. Bearbeitungsalgorithmus der PNDU.

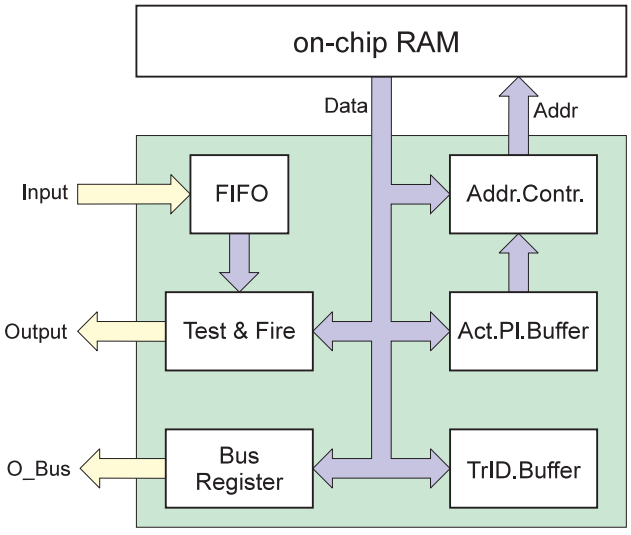

Abbildung 5. Vereinfachtes Blockschaltbild der PNDU.

wenn keine neuen Eingangssignalwerte vorhanden bzw. zwischengespeichert sind.

4 Entwurf der PNDU

Die in Abschnitt 3 hinsichtlich Transitionskodierung und Bearbeitungsalgorithmus vorgestellte PNDU wurde mit VHDL als synchrone Schaltung beschrieben und auf eine Alcatel 0,35 μm Standardzellentechnologie abgebildet. Abbildung 5 zeigt ein vereinfachtes Blockschaltbild der PNDU, die Systemsteuerung ist hier nicht abgebildet. Ziel der Umsetzung war es, pro Taktzyklus einen Speicherzugriff zu erreichen. Der Speicher wurde durch ein Makro als on-chip RAM vorgesehen. Das Adressierungsmodul (Addr.Contr.) verwaltet sowohl relative als absolute Sprünge des Adresszeigers. Die aktuellen Token werden durch Adresspointer auf den jeweiligen Platz in einem Speicher (Act.Pl.Buffer) verwaltet. Bei synchronisierenden Transitionen wird nach dem ersten Feuern die entsprechende Transitionskennung in einem weiteren Speicher (TrID.Buffer) abgelegt, um mehrmaliges Feuern dieser Transition aus Sichtweise der anderen Token zu verhindern. Dem Ausgangs-Port wird optional nach jeder Feuerung einer Transition ein neuer Wert zugewiesen (Bus Register). Die Konzessionsprüfung der Ausgangs- und Eingangssignalwerte wird zusammen mit der Zuweisung neuer Ausgangssignalwerte bei der Feuerung im “Test & Fire” Modul durchgeführt. Die Eingangssignalwerte werden nach einer Synchronisation auf den Systemtakt in einem “FIFO” zwischengespeichert. Dadurch wird erreicht, daß mehrmalige Änderungen dieser Werte während eines Bearbeitungszyklus gespeichert und zu einem späteren Zeitpunkt in den Bearbeitungszyklus eingebracht werden.

Befindet sich die PNDU in dem inaktiven Wartezustand des Bearbeitungsalgorithmus, so werden bei der synchronen Schaltung die Eingangssignale ständig aktiv abgefragt um eine erneute Wertänderung festzustellen. Bei einer asynchronen Schaltung (Berkel et al., 1999) kann eine derartige Abfrage ohne Schaltungsaktivität umgesetzt werden und ermöglicht dadurch eine ereignisgesteuerte Reaktivierung der PNDU. Die Eingangssignale der PNDU sind selbst auch asynchron und müssen bei der synchronen Schaltung synchronisiert werden, um metastabile Zustände zu verhindern. Bei asynchronen Schaltungsansätzen gibt es hierfür spezielle Gatter, sog. Mutex-Elemente, bei denen das Ausgangssignal solange inaktiv bleibt, bis sich intern der metastabile Zustand aufgelöst hat. Auch die unterschiedliche Bedeutung der einzelnen Transitionsworte und damit verbunden die unter-

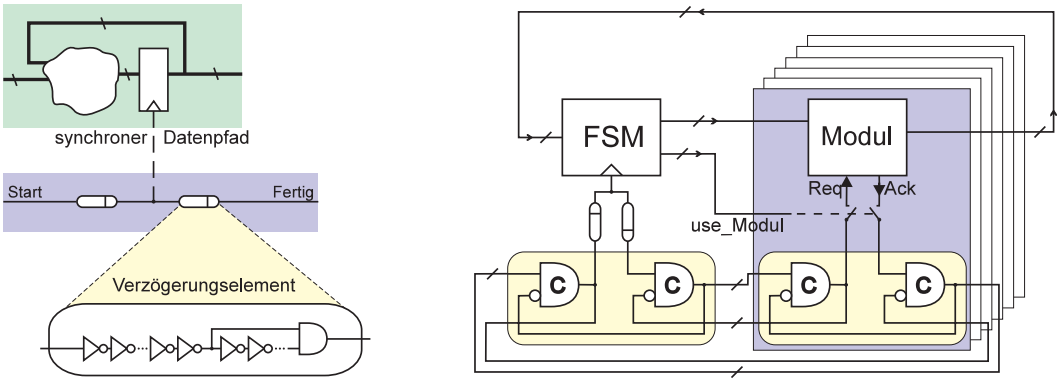

Abbildung 6. (a) Datenpfad mit Verzögerungsleitung und **(b)** selbstgetaktete Systemsteuerung.

schiedlich lange dauernden Bearbeitungsschritte werden in der synchronen Schaltung nicht berücksichtigt. Speziell für die PNDU sind zwei Vorteile einer asynchronen Schaltungsumsetzung von Bedeutung. Erstens wirkt sich der implizite Ruhezustand mit ereignisgesteuerter Reaktivierung günstig in Hinblick auf Low-Power bei batteriebetriebenen Geräten aus. Zweitens werden durch das Fehlen eines globalen Taktes geringere Spannungsspitzen erreicht, was sich positiv auf das Elektromagnetische Verhalten der Schaltung auswirkt.

Die PNDU wurde mit der gleichen Spezifikation auch als asynchrone bzw. selbstgetaktete Schaltung entworfen. Zielsetzung war hierbei in erster Linie nicht die Vorteile asynchroner Schaltungen zu beweisen, sonder vielmehr die Umsetzung einer selbstgetakteten Schaltung mit den gängigen Entwurfswerkzeugen aus dem synchronen Schaltungsentwurf.

Spezielle Lösungen werden hier nur kurz vorgestellt, wie sie in der selbstgetakteten PNDU umgesetzt wurden. Anstelle des globalen Taktes werden asynchrone Schaltungen mit lokalen Handshakes (Start- und Fertigsignalen) betrieben. Bei den Modulen, siehe Abb. 6a), wurde der Datenpfad mit VHDL auf RTL beschrieben und daraus die kritischen Pfade extrahiert. Die Steuersignale Start und Fertig werden durch spezielle Verzögerungsleitungen auf die Laufzeit des kritischen Pfades angepaßt (bundled data), wobei die Verzögerungszeit im Steuerpfad mit einer Sicherheit von 50% eingestellt wurde. Aufgrund der Verwendung des 4-Phasen-Protokolls konnten asymmetrische Verzögerungselemente eingesetzt werden. Die Systemsteuerung ist bei der PNDU eine zentrale Steuerung mit der Intension unnötige Modulaufrufe zu vermeiden. Mit einem gängigen Verfahren (Cortadella et al., 1997), das auf der Synthese von Signalübergangsgraphen (STG) beruht, konnte diese Systemsteuerung aufgrund ihrer Komplexität nicht realisiert werden. Die gewählte Lösung ist in Abb. 6b) abgebildet. Sie basiert auf einer Endlichen Zustandsmaschine (FSM) im Zusammenspiel mit einer 2-stufigen Ringpipeline. Die FSM übernimmt die Entscheidung, welches Modul aktiviert wird und erzeugt die entsprechenden Steuersignal. Da jedes Modul intern mit unterschiedlicher Verzögerungszeit arbeitet, ergeben sich daraus unterschiedliche interne Taktraten und Speicherzugriffsfrequenzen. Diese Systemsteuerung ist sehr flexibel und ermöglicht dadurch kurzfristige Anpassungen im Entwurfsprozeß. Die höherer Zahl interner Zustände verglichen mit der synchronen Systemsteuerung wirkt sich jedoch auf die Geschwindigkeit aus.

Die PNDU wurde als synchrone und selbstgetaktete VHDL-Beschreibung als Gatternetzliste auf Alcatel 0,35 μm Standardzellentechnologie synthetisiert. Zum Vergleich der beiden Ansätze wurde die minimale Parameterkonfiguration gewählt. Die PNDU hat hier 5 Eingangssignale, 8 Ausgangssignale und maximal 4 Token. Die Speichergröße von 256×16 Bit RAM ermöglicht eine maximale Netzgröße von 64 Transitionen. In dieser Konfiguration wird eine nicht aktivierte Transition nach 1–2 Taktzyklen erkannt. Zum Feuern der Transition werden 4–6 Taktzyklen benötigt. Die maximale Taktfrequenz bei der synchronen PNDU liegt bei f_{sync}=35–50 MHz je nach Optimierungsziel bei der Synthese. Die selbstgetaktete Schaltung erreichte eine mittlere Speicherzugriffsfrequenz von f_{st}=26 MHz, wobei sie im Betrieb zwischen 15–30 MHz schwankt. Der Flächenbedarf der synchronen PNDU liegt zwischen A_{sync}=0,403–0,452 mm^2 je nach Optimierungsziel. Die selbstgetaktete Schaltung benötigt A_{st}=0,446 mm^2. Diese Flächenwerte verstehen sich inklusive on-chip RAM, welches mit A_{RAM}=0,270 mm^2 bereits den größten Anteil der Schaltung einnimmt.

Zusammenfassung

Interpretierte Petri-Netze eignen sich zur Darstellung von Kontrollanwendungen und deren Steuerung für diskrete ereignisgesteuerte Systeme. Die hier vorgestellte PNDU ist eine neuartige schaltungstechnische Implementierung von Petri-Netzen. Die Konzeption dieses programmierbaren Petri-Netz Controllers wurde hinsichtlich Kodierung und Bearbeitungsalgorithmus vorgestellt. Das Prinzip der lokalen Feuerung ermöglicht die Einbindung eines RAM Speichers mit geringer Wortbreite. Die Netzgröße ist variabel und nicht a priori durch den Bearbeitungsalgorithmus festgelegt. Die PNDU wurde als synchrone und selbstgetaktete Schaltung mit VHDL beschrieben und auf eine Alcatel

0,35 μm Standardzellentechnologie abgebildet. Die selbstgetaktete PNDU wurde mit gängigen Synthesewerkzeugen für synchrone Schaltungen entworfen. Die Flächenwerte beider Entwürfe waren nahezu identisch, die maximale Taktfrequenz der synchronen Schaltung konnte bei der selbstgetakteten Implementierung nicht erreicht werden. Die selbstgetaktete PNDU hat im Wartezustand keine Schaltungsaktivität und wird ereignisgesteuert reaktiviert. Ein Prototyp einer synchronen 8-Bit PNDU wurde als Chip gefertigt und getestet (Bulach, 2001).

Literatur

Berkel, C. H. (Kees) van, Josephs, M. B., and Nowick, S. M.: Scanning the Technology: Applications of Asynchronous Circuits, Proceedings of the IEEE, 87, 2, 223–233, February 1999.

Bulach, S.: The Design and ASIC Realization of a Custom Petri Net Based Programmable Controller, Dissertation Universität Ulm, 2001.

Cortadella, J., Kishinevski, M., Kondratyev, A., Lavagno, L., and Yakovlev, A.: Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers, IEICE Transactions on Information and Systems, E80-D, 3, 315–325, March 1997.

Crockett, D., Desrochers, A., DiCesare, F., and Ward, T.: Implementation of a Petri Net Controller for a Machining Workstation, Proceedings of the IEEE International Conference on Robotics and Automation, 1861–1867, 1987.

Hartenstein, R. W., Hirschbiel, A., and Weber, M.: Patil Array, Report, CS Department, Universität Kaiserslautern, 1987.

Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, 1–3, Springer Verlag, 1997.

Kamakura, T., Shimoda, T., Dohi, Y., and Murakoshi, H.: Implementation of a Large Petri Net by a Group of Petri Net Controller, Proceedings of the IECON'97, 23rd International Conference on Industrial Electronics, Control and Instrumentation, New Orleans, Louisiana, USA, 1210–1215, (1997-11), 1997.

Murakoshi, H. and Dohi, Y.: Petri Net Based High Speed Programmable Controller by ASIC Memory, Proceedings of the 29th SICE Annual Conference, Tokyo, Japan, 697–700, (1990–07), 1990.

Murata, T., Komoda, N., Matsumoto, K., and Haruna, K.: A Petri Net-Based Controller for Flexible and Maintainable Sequence Control and its Application in Factory Automation, IEEE Transactions on Industrial Electronics, IE-33, 1, 1–8, February 1986.

Murata, T.: Petri Nets: Properties, Analysis and Applications, Proceedings of the IEEE, 77(4), 541–580, 1989.

Patil, S.: Circuit Implementation of Petri Nets, Computational Structures Group Memo 73, MIT Project MAC, Cambridge Massachusetts, 1972.

Patil, S.: An Asynchronous Logic Array, Computational Structures Group Memo 111, MIT Project MAC, Cambridge Massachusetts, 1975.

Petri, C. A.: Kommunikation mit Automaten, Dissertation Universität Bonn, 1962.

Valette, R., Courviosier, M., Bigou, JM., and Albukerque, J.: A Petri Net Based Programmable Logic Controller, Proceedings of the 1st International IFIP Conference on Computer Applications in Production and Engineering, (Warman, E. A. (Ed.)), 103–116, North Holland, 1983.