IP²

Anwendungsspezifische Entwurfsmethodik für flexible SoC in zukünftigen Mobilfunksystemen
Intellectual Property Prinzipien für konfigurierbare Basisband SoCs in Internet-Protokoll basierten Mobilfunknetzen

BMBF-Projekt • Förderkennzeichen 01 M 3059 F

VHDL-Modelle für zeitbehaftete Petri-Netze

Zielstellung

Zeitbehaftete Petri-Netze sind ein Hilfsmittel zur Beschreibung des Informations- und Steuerflusses nebenläufiger dynamischer Systeme. Sie sind geeignet für die Modellierung kommunizierender Prozesse in Software-Anwendungen und auch für die Modellierung asynchron kommunizierender Hardwarekomponenten.
Durch die Anwendung von Petri-Netzen können sowohl qualitative als auch quantitative Aussagen hinsichtlich der Leistung und Zuverlässigkeit eines Systems gewonnen werden.

Für Anwendungen im Embedded Systems Design soll ein Lösungsansatz gezeigt werden, wie Petri-Netze als formale Top-Level-System-Spezifikationen in den Hardware-Software-Co-Design-Flow integriert werden können.

Design-Flow

Ausgangspunkt ist ein zuvor erstellter und analysierter Petri-Netz-Graph, d.h. die Topologie ist bekannt.

Die Umsetzung des Petri-Netz-Graphen in ein VHDL-Modell ermöglicht die dynamische Verifikation des Netzes mittels Simulation für Schritte der Entwurfsverfeinerung.

Zudem bieten synthesefähige VHDL-Modelle die Option einer Implementierung in Hardware.
Die Synthese solcher VHDL-Spezifikationen führt zu Hardwarestrukturen aus Komponenten die über Handshakingmechanismen asynchron miteinander kommunizierenden.

Ausgangspunkt

Zeitbewertungen in Petri-Netzen

		Zeitbewertung, z.B	Interpretation, z.B.
Lösungsvarianten	Marken Stellen	⇨ Gültigkeit von Marken	⇨ verlustbehaftete Kommunikation
	Kanten	⇨ Durchlässigkeit	⇨ Signallaufzeiten Reaktionszeiten
Unser Ansatz	Transitionen	⇨ Schaltdauer	⇨ Abstraktion für Echtzeit-Prozesse

Darstellung von Petri-Netzen

Darstellungsform	Modellierungssicht	Simulationsprinzipien, z.B.
mathematische Form (Vektoren & Netzmatrix)	⇨ imperativ	⇨ zeit-gesteuert
Erreichbarkeitsgraph	⇨ imperativ	⇨ zeit-gesteuert
gerichteter Graph (Petri-Netz-Graph)	⇨ reaktiv objektorientiert	⇨ ereignis-gesteuert **VHDL**

Modellierung

Struktur & Verhalten

3-Komponenten-Modell

entity	**place**
parameter (generics)	maximum capacity initial marking number of related arcs
output port (state record)	number of tokens ∈ INTEGER empty capacity ∈ INTEGER

entity	**transition**
parameter (generics)	delay time timing realisation {internal, external} number of related arcs
output port (state record)	action ∈ {idle, start, active, completed}

entity	**arc**
parameter (generics)	weight
output port (state record)	action ∈ {passive, ready, active} weight ∈ INTEGER

Konfliktauflösung

Typische Probleme bei der Analyse von Petri-Netzen sind Konflikte, wenn z.B. Marken von einer Stelle durch mehrere Transitionen genutzt werden können oder mehrere Transitionen Marken an eine kapazitätsbegrenzte Stelle übergeben können.

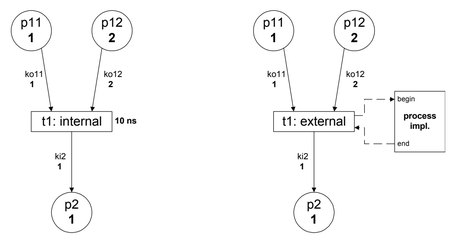
Für Petri-Netze mit zeitbehafteten Transitionen kommt als weiterer Aspekt hinzu, dass für eine aktive Transition abzusichern ist, dass die zu Beginn der Aktivität als aufnahmebereit identifizierte Stelle nicht inzwischen durch eine andere Transition belegt worden ist.

In unserer Lösung gibt es daher verschiedene Modelle für drei Level der Konfliktäuflösung zur Auswahl:
- keine Konfliktauflösung
- Konfliktauflösung
- Konfliktauflösung und Kapazitätsreservierung

Die Konfliktauflösung kann wahlweise stochastisch oder deterministisch erfolgen.

Realisierung der Zeitbewertung

Für die Simulation der Transitionsschaltzeit sind zwei Varianten implementiert. Die Zeitverzögerung kann einfach als internes wait-Statement eingeführt sein. Alternativ kann diese Verzögerung extern berechnet werden. In diesem Falle erzeugt das Transitions-Entity einen Start-Impuls für einen externen Prozess und wartet auf dessen Terminierung. Auf diese Weise kann das Zeitverhalten stochastisch oder deterministisch modelliert werden.



Im Prozess der Entwurfsverfeinerung eröffnet die externe Variante die Nutzung dieser Option als Interface für die Verbindung von Steuer- und Datenfluß eines Systems.

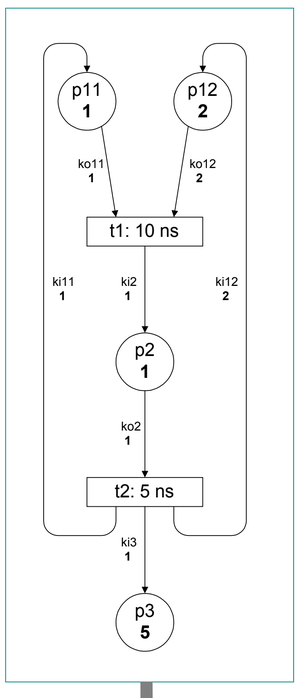
Hierarchische Netze

Zusätzlich zur Möglichkeit der Beschreibung von flachen PN-Graphen-Strukturen, ist das Konzept nutzbar für die Beschreibung hierarchischer Petri-Netze. Sub-Netze können als Instanzen in hierarchischen Petri-Netzen verwendet werden.

Simulation

Demo-Beispiel

In diesem Beispiel ist die Überführung eines Petri-Netz-Graphen in eine VHDL-Netzliste gezeigt.
Stellen, Transitionen und Kanten sind in der Strukturbeschreibung als Komponenten instanziiert und über die Ports miteinander verbunden.. Deren Zustände sind als globale Signale definiert.
Das Beispiel hat eine kollisionsfreie Topologie, daher sind Entities ohne Konfliktbehandlung referenziert. Die Verzögerungszeit der Prozesse wird in den Transitionen intern realisert.



Transformation

```
architecture structure of fdl_pn is

    for all : place use entity work.e_place_simple(simu);
    for all : transition use entity work.e_transition_simple(simu);
    for all : arc use entity work.e_arc_simple(simu);

    signal p11, p12, p2, p3 : t_place_state;
    signal t1, t2 : t_trans_state;
    signal ki11, ki12, ko11, ko12, ki2, ko2, ki3 : t_arc_state;

    ...

begin
    .
    .
    .

    c_p11 : place
        generic map  (g_capacity => 1, g_no_arcs => 2, g_initial_marking => 0)
        port map     (p_arcs(1) => ki11, p_arcs(2) => ko11, p_state => p11, p_reset => reset );

    c_p12 : place
        generic map  (g_capacity => 2, g_no_arcs => 2, g_initial_marking => 0)
        port map     (p_arcs (1) => ki12, p_arcs (2) => ko12, p_state => p12, p_reset => reset );

    c_ko11 : arc
        generic map  (g_weight => -1 )
        port map     (p_place => p11, p_trans => t1, p_state => ko11, p_reset => reset );

    c_ko12 : arc
        generic map  (g_weight => -2 )
        port map     (p_place => p12, p_trans => t1, p_state => ko12, p_reset => reset );

    c_t1 : transition
        generic map  (g_delay => 10 ns, g_timing => internal, g_no_arcs => 3 )
        port map     (p_arcs (1) => ko11, p_arcs (2) => ko12, p_arcs (3) => ki2, p_state => t1, p_reset => reset );

    c_ki2 : arc
        generic map  (g_weight => 1 )
        port map     (p_place => p2, p_trans => t1, p_state => ki2, p_reset => reset );

    c_p2 : place
        generic map   (g_capacity => 1, g_no_arcs => 2, g_initial_marking => 1)
        port map      (p_arcs (1) => ki2, p_arcs (2) => ko2, p_state => p2, p_reset => reset );

    .
    .
    .
end structure;
```

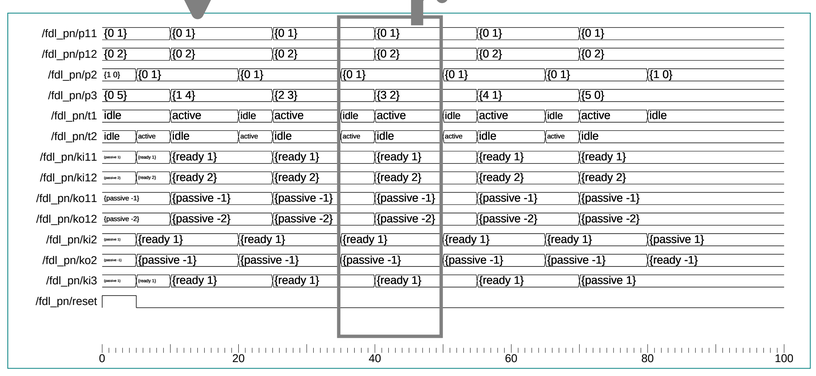
Im unteren Waveform sind die Resultate eine Simulation gezeigt, bei der die Stelle p2 mit einer Marke und die anderen Stellen mit keiner Marke initialisiert wurden.
Es ist zu sehen, dass die Transitionen t1 und t2 abwechselnd aktiv sind und sich die Anzahl der Marken in Stelle p3 beginnend bei Null Marken (Markierung {number of tokes:0, empty capacity: 5}) kontinuierlich erhöht, bis die Kapazitätsgrenze erreicht ist (Markierung {5 0}).

Da nur die Transitionen zeitbehaftet sind, erfolgen die meisten Zustandsänderungen ohne reale Zeitverzögerung. Diese Änderungen sind daher in den Waveforms nicht sichtbar. In der Tabelle ist daher dargestellt, welche Zustandssequenzen in den delta-delay-Schritten im VHDL-Simulator erfolgen: für die Kanten die Abfolge {passive → ready → active → passive} und für die Transitionen: {idle → start → active → completed → idle}.

ns	delta	/p11	/p12	/p2	/p3	/t1	/t2	/ki11	/ki12	/ko11	/ko12	/ki2	/ko2	/ki3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
35	+16	0 1	0 2	0 1	2 3	idle	active	ready 1	ready 2	passive -1	passive -2	ready 1	passive -1	ready 1
40	+1	0 1	0 2	0 1	2 3	idle	complete	ready 1	ready 2	passive -1	passive -2	ready 1	passive -1	ready 1
40	+2	0 1	0 2	0 1	2 3	idle	idle	ready 1	ready 2	passive -1	passive -2	ready 1	passive -1	ready 1
40	+3	0 1	0 2	0 1	2 3	idle	idle	active 1	active 2	passive -1	passive -2	ready 1	passive -1	active 1
40	+4	0 1	0 2	0 1	2 3	idle	idle	ready 1	ready 2	passive -1	passive -2	ready 1	passive -1	ready 1
40	+5	1 0	2 0	0 1	3 2	idle	idle	ready 1	ready 2	passive -1	passive -2	ready 1	passive -1	ready 1
40	+7	1 0	2 0	0 1	3 2	idle	idle	passive 1	passive 2	ready -1	ready -2	ready 1	passive -1	ready 1
40	+10	1 0	2 0	0 1	3 2	start	idle	passive 1	passive 2	ready -1	ready -2	ready 1	passive -1	ready 1
40	+11	1 0	2 0	0 1	3 2	active	idle	passive 1	passive 2	ready -1	ready -2	ready 1	passive -1	ready 1
40	+12	1 0	2 0	0 1	3 2	active	idle	passive 1	passive 2	active -1	active -2	ready 1	passive -1	ready 1
40	+13	1 0	2 0	0 1	3 2	active	idle	passive 1	passive 2	ready -1	ready -2	ready 1	passive -1	ready 1
40	+14	0 1	0 2	0 1	3 2	active	idle	passive 1	passive 2	ready -1	ready -2	ready 1	passive -1	ready 1
40	+16	0 1	0 2	0 1	3 2	active	idle	ready 1	ready 2	passive -1	passive -2	ready 1	passive -1	ready 1
50	+1	0 1	0 2	0 1	3 2	complete	idle	ready 1	ready 2	passive -1	passive -2	ready 1	passive -1	ready 1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

dynamische Verifikation

Zoom



TECHNISCHE UNIVERSITÄT DRESDEN

Heike Garbe, Olaf Schäfer, Hans-Joachim Jentschel

Institut für Verkehrsinformationssysteme
Arbeitsgruppe Entwurf nachrichtentechnischer Systeme