In *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*.
By Jürgen Ruf (Editor), GIITGGMM-Workshop, Shaker Verlag, Aachen, Februar 25-27, 2002

# Modellierung Rekonfigurierbarer Systemarchitekturen

Christian Haubelt, Jürgen Teich
DATE, University of Paderborn
Paderborn, Germany
{haubelt, teich}@date.upb.de

Kai Richter, Rolf Ernst
IDA, Technical University of Braunschweig
Braunschweig, Germany
{richter, ernst}@ida.ing.tu-bs.de

*Dieser Artikel stellt ein Modell zur Spezifikation rekonfigurierbarer Systeme vor. Die Spezifikation beruht hierbei auf einem graphenorientierten Ansatz und erlaubt die Modellierung des gewünschten Verhaltens sowie Ressourcen- und Abbildungsbeschränkungen. Durch diesen Ansatz wird es möglich, Begriffe wie Allokation und Bindung auch für rekonfigurierbare Systeme formal zu definieren.*

## 1 Einleitung

Nicht nur die steigende Komplexität, sondern auch die geänderten Anforderungen an heutige Systeme verlangt nach neuen Methoden und Modellen zur Analyse und Optimierung eingebetteter Systeme. So gibt es kaum noch Systeme, die ohne programmierbare Komponenten auskommen. Ein starker Zuwachs ist ebenfalls im Bereich der Systeme, die auf rekonfigurierbarer Hardware basieren, zu verzeichnen.

Eine Möglichkeit, die Modellierung von Systemen trotz ihrer Komplexität zu erleichtern, liegt in der Hierarchisierung der Modelle. Verschiedene Modelle wurden im Laufe der Zeit um Hierarchie erweitert. So z. B. Petri Netze [4] oder Datenflussgraphen [1]. Auch wurden neue Modelle, die bereits Hierarchisierungsmöglichkeiten beinhalten, entwickelt (siehe z. B. [3] und [9]).

Allen Modellen ist gemein, dass sie lediglich die Modellierung des Verhaltens eines Systems erlauben. Dies mag noch für Betrachtungen ausreichen, in denen eine beschränkte Zielarchitektur zur Verfügung steht. Dies reicht aber leider nicht aus, sofern komplexere Ressourcenbeschränkungen berücksichtigt werden müssen. Noch gravierender werden die Auswirkungen, wenn die Zielarchitektur rekonfigurierbare Komponenten beinhalten kann.

In diesem Bericht stellen wir ein hierarchisches Modell zur Spezifikation von rekonfigurierbaren Systemen vor. Das Modell unterstützt dabei die Formulierung von Ressourcenbeschränkungen, inklusive rekonfigurierbaren Ressourcen, sowie Bindungsbeschränkungen von Operationen. In Abschnitt 2 werden zunächst die theoretischen Grundlagen des hierarchischen Modells beschrieben, bevor in Abschnitt 3 das eigentliche Modell vorgestellt wird. Dort werden auch kleinere Beispiele des Modells diskutiert.

## 2 Hierarchische Graphen

Der in [2] beschriebene Ansatz zur Systemsynthese basiert auf einem graphentheoretischen Modell, dem sog. *Spezifikationsgraphen*. Der Spezifikationsgraph besteht aus einem Problemgraphen, der das

gewünschte Verhalten spezifiziert, einem Architekturgraphen, der die zur Verfügung stehende Architekturvielfalt modelliert, und Abbildungskanten, die die Implementierbarkeit von Knoten des Problemgraphen auf Knoten des Architekturgraphen anzeigen. Während hierarchische Strukturen für die Zielarchitektur auf Grund der Mehrstufigkeit in diesem Modell darstellbar sind, gilt dies leider nicht bei der Modellierung des gewünschten Verhaltens. Aber gerade die Hierarchisierungsmöglichkeit auch im Problemgraphen ist essentiell für die Modellierung von rekonfigurierbaren Systemen. Dieser Abschnitt definiert ein hierarchisches Graphenmodell, welches später verwendet wird, um rekonfigurierbare Systeme zu modellieren.

In hierarchischen Graphen können einzelne Knoten durch Subgraphen verfeinert werden. Besteht zusätzlich die Möglichkeit, einen einzelnen Knoten durch alternative Subgraphen zu ersetzen, so werden die Subgraphen als *Cluster* und die zu ersetzenden Knoten als *Interfaces* bezeichnet. Die Definitionen 1 bis 3 geben den grundlegenden Aufbau eines hierarchischen Graphen dieser Form wieder. Aus Platzgründen werden hier nicht alle Details definiert. Eine vollständige Beschreibung befindet sich in [5, 6].

**Definition 1 (Cluster)**
*Ein* Cluster $\gamma(I, O, V, E, \Psi)$ *enthält einen gerichteten, nichthierarchischen Graphen* $G = (V_\mathrm{G}, E_\mathrm{G})$*, wobei*

> $V$ *und* $\Psi$ *eine Partitionierung der Menge der Knoten* $V_\mathrm{G}$ *des Graphen* $G$ *bilden und* $E$ *entspricht der Menge der Kanten* $E_\mathrm{G}$*.*

*Weiterhin sind*

> $I = \{i_1, i_2, \ldots i_{N_I}\}$ *die Menge der Eingänge,*
>
> $O = \{o_1, o_2, \ldots o_{N_O}\}$ *die Menge der Ausgänge,*
>
> $V = \{v_1, v_2, \ldots, v_{N_V}\}$ *die Menge der nichthierarchischen Knoten, die sog.* Blattknoten*,*
>
> $E$ *die Menge der Kanten und*
>
> $\Psi = \{\psi_1, \psi_2, \ldots, \psi_{N_\Psi}\}$ *die Menge der hierarchischen Knoten, die sog.* Interfaces *nach Definition 2.*

*Sei* $I_\Psi$ *die Vereinigung aller Mengen der Eingänge der Interfaces nach Definition 2 und* $O_\Psi$ *die Vereinigung aller Mengen der Ausgänge der Interfaces nach Definition 2.*

*Sei weiterhin die Vereinigung der Ein- und Ausgänge* $(I \cup O)$ *die Menge der sog.* Ports*. Ports bilden die Anschlüsse eines Clusters zum Einbetten in eine Hierarchie.*

*Für die Elemente* $e \in E$ *gilt:*

$$\begin{aligned} E \quad \subseteq \quad & (I \times V) \cup (I \times I_\Psi) \cup (V \times I_\Psi) \cup (V \cup V) \cup \\ & (O_\Psi \times V) \cup (O_\Psi \times O) \cup (V \times O) \end{aligned}$$

*Die Abbildung der Ports des Clusters* $(I \cup O)$ *und der Ports der Interfaces* $(I_\Psi \cup O_\Psi)$ *muss hierbei eindeutig sein.* □

Während die Blattknoten nicht weiter verfeinert werden können, dienen die sog. *Interfaces* als Schnittstellenbeschreibung zum Einbetten von Subgraphen (Clustern). Hierbei ist ein Interface wie folgt definiert:

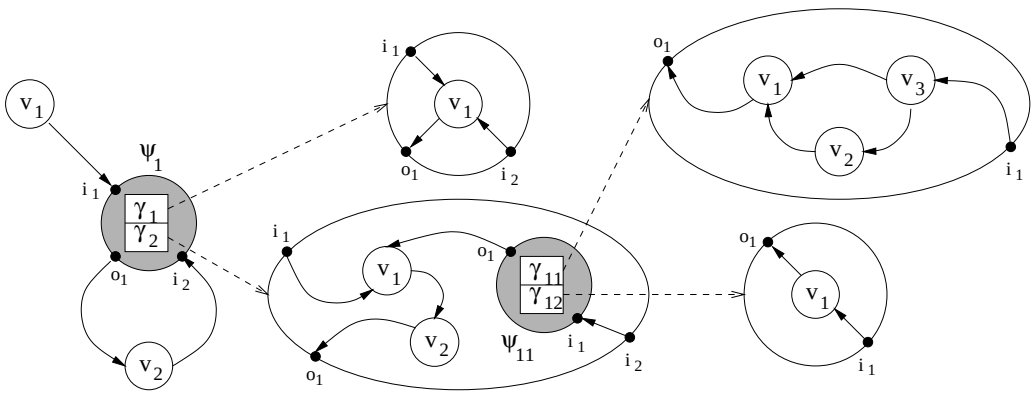


Abbildung 1: Beispiel für einen hierarchischen Graphen

**Definition 2 (Interface)**
*Ein* Interface $\psi(I, O, \Gamma, \Phi)$ *ist ein 4-Tupel* $(I, O, \Gamma, \Phi)$*, wobei*

$I = \{i_1, i_2, \ldots i_{N_I}\}$ *die Menge der Eingänge,*

$O = \{o_1, o_2, \ldots o_{N_O}\}$ *die Menge der Ausgänge,*

$\Gamma = \{\gamma_1, \gamma_2, \ldots \gamma_{N_\Gamma}\}$ *die Menge der assoziierten Cluster und*

$\Phi : I_\Gamma \cup O_\Gamma \mapsto I \cup O$ *eine Funktion ist, die den Ports der assoziierten Cluster* $\gamma \in \Gamma$ *einen Port des Interfaces* $\psi$ *zuordnet. Hierin bedeuten*

$I_\Gamma$ *die Vereinigung aller Mengen der Eingänge der Cluster* $\gamma \in \Gamma$

$O_\Gamma$ *die Vereinigung aller Mengen der Ausgänge der Cluster* $\gamma \in \Gamma$

*Diese Funktion wird im Folgenden als* Port-Mapping *bezeichnet.*

□

Die beiden rekursiven Definitionen 1 und 2 zusammen ermöglichen somit den Aufbau einer Hierarchie. Im Folgenden wird ein Cluster $\gamma$, dessen Interfacemenge leer ist ($\gamma.\Psi = \emptyset$) als *Blatt-Cluster* bezeichnet. Mit den oben genannten Definitionen ist es möglich, den Begriff des *hierarchischen Graphen* wir folgt zu definieren:

**Definition 3 (Hierarchischer Graph)**
*Ein* hierarchischer Graph $G$ *ist ein Cluster* $\gamma$ *nach Definition 1 mit der Bedingung, dass die Ein- und Ausgangsmengen leer sind (*$G.I = G.O = \emptyset$*).* □

**Beispiel 1** Abbildung 1 zeigt einen hierarchischen Graphen G nach Definition 3. Der Graph ergibt sich zu:

$$\begin{aligned} \mathrm{G} &= (\mathrm{I}, \mathrm{O}, \mathrm{V}, \mathrm{E}, \Psi) \\ &= (\emptyset, \emptyset, \{\mathrm{v}_1, \mathrm{v}_2\}, \{(\mathrm{v}_1, \psi_1.\mathrm{i}_1), (\mathrm{v}_2, \psi_1.\mathrm{i}_2), (\psi_1.\mathrm{o}_1, \mathrm{v}_2)\}, \{\psi_1\}) \end{aligned}$$

Die einzige hierarchische Komponente im Graphen G ist somit das Interface $\psi_1$, welches nach Definition 2 wie folgt gegeben ist:

$$\begin{aligned} \psi_1 &= (\mathrm{I}_{\psi_1}, \mathrm{O}_{\psi_1}, \Gamma_{\psi_1}, \Phi_{\psi_1}) \\ &= (\{\mathrm{i}_1, \mathrm{i}_2\}, \{\mathrm{o}_1\}, \{\gamma_1, \gamma_2\}, \\ &\quad \{\Phi(\gamma_x.\mathrm{i}_y) = \mathrm{i}_y, \Phi(\gamma_x.\mathrm{o}_1) = \mathrm{o}_1 \mid x = 1, 2; y = 1, 2\}) \end{aligned}$$

Die beiden Cluster $\gamma_1$ und $\gamma_2$ sind nach Definition 1 gegeben durch:

$$\begin{aligned}
\gamma_1 &= (\mathrm{I}_{\gamma_1}, \mathrm{O}_{\gamma_1}, \mathrm{V}_{\gamma_1}, \mathrm{E}_{\gamma_1}, \Psi_{\gamma_1}) \\
&= (\{\mathrm{i}_1, \mathrm{i}_2\}, \{\mathrm{o}_2\}, \{\mathrm{v}_1\}, \{(\mathrm{i}_1, \mathrm{v}_1), (\mathrm{i}_2, \mathrm{v}_1), (\mathrm{v}_1, \mathrm{o}_1)\}, \emptyset) \\
\gamma_2 &= (\mathrm{I}_{\gamma_2}, \mathrm{O}_{\gamma_2}, \mathrm{V}_{\gamma_2}, \mathrm{E}_{\gamma_2}, \Psi_{\gamma_2}) \\
&= (\{\mathrm{i}_1, \mathrm{i}_2\}, \{\mathrm{o}_2\}, \{\mathrm{v}_1, \mathrm{v}_2\}, \{(\mathrm{i}_1, \mathrm{v}_1), (\mathrm{v}_2, \mathrm{o}_1), (\mathrm{v}_1, \mathrm{v}_2), \\
&\quad (\psi_1.\mathrm{o}1, \mathrm{v}_1), (\mathrm{i}_2, \psi_1.\mathrm{i}_1)\}, \{\psi_{11}\})
\end{aligned}$$

Die weiteren Elemente können Abbildung 1 entnommen werden. □

Ist zu einem Interface $\psi$ ein zugehöriger Cluster $\gamma \in \psi.\Gamma$ ausgewählt, so ist es möglich das Interface $\psi$ durch $\gamma$ zu ersetzen. Es ist also möglich, die Hierarchie aufzulösen. Das Ergebnis ist ein nicht-hierarchischer Graph. Für alle folgenden Modelle gilt, dass zu jedem Zeitpunkt exakt ein Cluster pro Interface selektiert ist. Die Cluster stellen somit alternative Ersetzungen des Interfaces dar.

# 3 Spezifikation hierarchischer Systeme

Nachdem im vorangegangenen Abschnitt hierarchische Graphen definiert wurden, können diese nun zur Modellierung von eingebetteten Systemen verwendet werden. In Anlehnung an [2] wird ein Modell entwickelt, welches die Verhaltensspezifikation von Systemen dahingehend erweitert, dass eine detaillierte Modellierung von Architektureigenschaften und Abbildungsalternativen ermöglicht wird.

## 3.1 Hierarchische Architekturmodellierung

Ein Architekturgraph ist die Abstraktion von einer möglichen Zielarchitektur. Die einzelnen Bestandteile dieses Graphen können, müssen aber nicht zwingend zur Implementierung verwendet werden. Hier soll ein Ansatz gewählt werden, der das Konzept der hierarchischen Graphen zur Modellierung der Architekturvielfalt nutzt. Einen *hierarchischen Architekturgraphen* erhält man, wenn man, wie in Abschnitt 2 beschrieben, Knoten durch Subgraphen ersetzen kann.

Die Benutzung von hierarchischen Komponenten kann aus unterschiedlichen Gründen erfolgen:

1. Modellierung alternativer Kommunikationsstrukturen.
2. Modellierung unterschiedlicher Versionen/Konfigurationen funktionaler Ressourcen.

Der erste Punkt bezieht sich auf die eigentliche Struktur der zu verwendenden Hardware, der zweite Punkt auf die zu verwendenden Module.

**Beispiel 2** Abbildung 2 zeigt drei unterschiedliche Architekturgraphen mit jeweils vier funktionalen Ressourcen ($\mathrm{v}_1$ bis $\mathrm{v}_4$). Beim Architekturgraphen aus Abbildung 2a) erfolgt die Kommunikation der Ressourcen über einen Shared Bus $\mathrm{c}_1$. Abbildung 2b) zeigt eine Ring-Kommunikation dieser Ressourcen. Bei diesen beiden Architekturgraphen ist die zu verwendende Kommunikationsstruktur fest vorgegeben. In Abbildung 2c) sind beide Alternativen in ein und demselben Architekturgraphen dargestellt, und können die hierarchische Busressource, die von den vier funktionalen Ressourcen verwendet wird, ersetzen. Das letzte Beispiel modelliert somit das Wissen über Alternativen und somit den direkten Vergleich dieser in einem Modell. Von den beiden Alternativen kann immer nur genau eine zu jedem Zeitpunkt selektiert sein. Würde es sich bei dieser Kommunikationsstruktur um unterschiedliche Konfigurationen eines FPICs handeln, so könnte die Struktur der Kommunikations sogar zur Laufzeit verändert werden. □

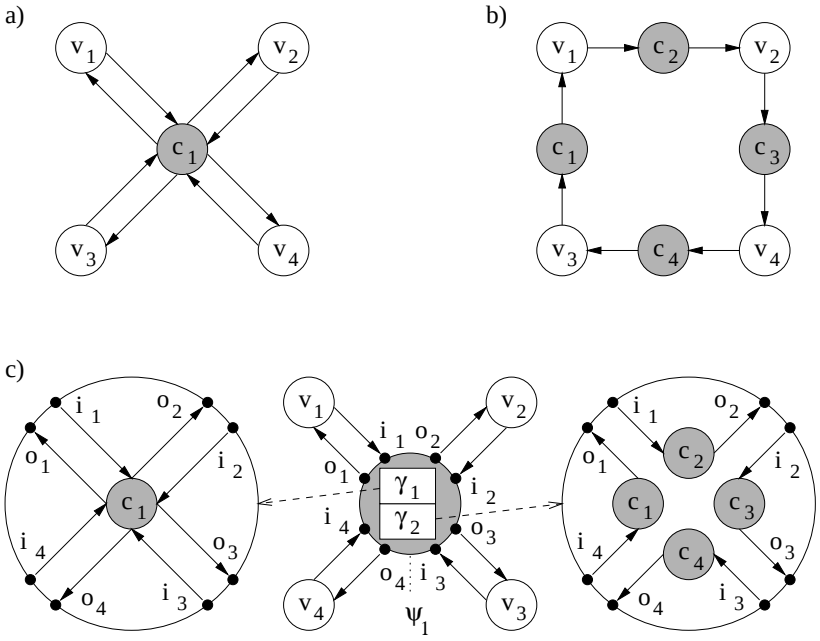


Abbildung 2: Varianten c) einer Kommunikationsstruktur als Stern a) oder als Ring b).

Neben der Modellierung von Alternativen zur Realisierung von Kommunikationsknoten, können auch funktionale Ressourcen durch Subgraphen implementiert werden.

**Beispiel 3** Abbildung 3 zeigt ein Beispiel für ein funktionales Modul im Architekturgraphen. Der Entwickler ist sich in diesem Fall darüber im Klaren, dass er einen Mikrocontroller der Familie MC6833X einsetzen möchte. Was er hingegen noch nicht entscheiden konnte, ob die Timing-Funktionen des GPT (**G**eneral **P**urpose **T**imer) des MC68331 ausreichend sind oder doch die wesentlich flexibleren Funktionen der TPU (**T**ime **P**rocessor **U**nit) des MC68332 benötigt werden. In beiden Fällen wird auf die Funktionalität der CPU32, des EBI (**E**xternal **B**us **I**nterface) und QSM (**Q**ueued **S**erial **M**odule) zurückgegriffen. Die einzelnen Subkomponenten sind über einen IMB (**I**nter **M**odule **B**us) mit einander verbunden.

Zwar hätte man direkt den Austausch des GPT-Moduls gegen das TPU-Modul modellieren können. Hierdurch hätte man jedoch den physikalischen Zusammenhang mit den restlichen Subkomponenten verloren. Auch die zusätzlich modellierte Subkomponente des RAM hätte man so nicht direkt erfassen können. □

Mit den oben gezeigten Beispielen kann der sog. *hierarchische Architekturgraph* definiert werden.

**Definition 4 (Hierarchischer Architekturgraph)**
*Ein* hierarchischer Architekturgraph $G_{\mathrm{A}}$ *ist ein bipartiter hierarchischer Graph nach Definition 3, wobei in jedem Cluster* $\gamma$ *die Menge der Blattknoten* $V_\gamma$ *durch die Mengen* $V_{\mathrm{f},\gamma}$ *und* $V_{\mathrm{c},\gamma}$ *partitioniert wird. Die Elemente* $v \in V_{\mathrm{f},\gamma}$ *bzw.* $v \in V_{\mathrm{c},\gamma}$ *modellieren hierbei funktionale bzw. Kommunikationsressourcen.*

*Den Elementen der Menge aller Knoten* $V_{\mathrm{all}}(G_{\mathrm{A}})$ *des hierarchischen Architekturgraphen* $G_{\mathrm{A}}$ *wird eine Menge von (nicht weiter spezifizierten) Attributen zugeordnet.* □

Die zuletzt genannten Attribute werden erst zu einem späteren Zeitpunkt in der Analyse und Synthese zur Optimierung verwendet.

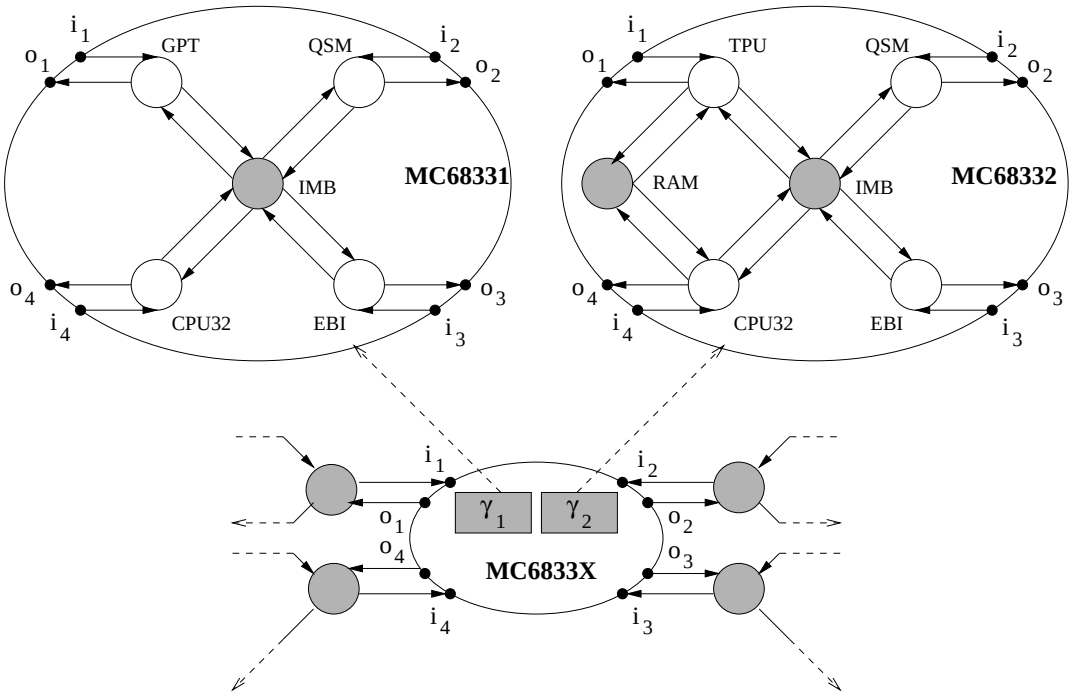


Abbildung 3: Entwurfsalternativen einer funktionalen Ressource (Mikrocontroller)

Damit der hierarchische Architekturgraph einen bipartiten hierarchischen Graphen bildet, wurden die Mengen der Blattknoten in den Clustern durch eine Bipartition ersetzt. Im Fall des hierarchischen Architekturgraphen ist die Bipartitheit vorausgesetzt, d. h. per definitionem sind immer funktionale Ressourcen mit Kommunikationsressourcen verbunden, und umgekehrt. Der hierarchische Architekturgraph wird im Folgenden einfach als Architekturgraph bezeichnet.

**Beispiel 4** Zur Verdeutlichung der vorherigen Definitionen wird noch einmal Abbildung 2c) betrachtet. Der hierarchische Architekturgraph $\mathrm{G_A}$ aus Abbildung 2c) ist nach Definition 4 gegeben durch:

$$\mathrm{G_A} = (\mathrm{I}, \mathrm{O}, \mathrm{V}, \mathrm{E}, \Psi) \quad \text{mit} \quad \mathrm{I} = \mathrm{O} = \emptyset, \mathrm{V} = \mathrm{V_f} \cup \mathrm{V_c}$$

mit

$$\begin{aligned}
\mathrm{V_f} &= \{\mathrm{v}_1, \mathrm{v}_2, \mathrm{v}_3, \mathrm{v}_4\} \\
\mathrm{V_c} &= \emptyset \\
\mathrm{E} &= \{(\mathrm{v}_1, \psi_1.\mathrm{i}_1), (\psi_1.\mathrm{o}_1, \mathrm{v}_1), (\mathrm{v}_2, \psi_1.\mathrm{i}_2), (\psi_1.\mathrm{o}_2, \mathrm{v}_2), \\
&\quad (\mathrm{v}_3, \psi_1.\mathrm{i}_3), (\psi_1.\mathrm{o}_3, \mathrm{v}_3), (\mathrm{v}_4, \psi_1.\mathrm{i}_4), (\psi_1.\mathrm{o}_4, \mathrm{v}_4)\} \\
\Psi &= \{\psi_1\}
\end{aligned}$$

Das einzige Interface $\psi_1$ ist nach Definition 2 gegeben durch:

$$\psi_1 = (\mathrm{I}_{\psi_1}, \mathrm{O}_{\psi_1}, \Gamma_{\psi_1}, \Phi_{\psi_1}) \text{, wobei}$$

$$\begin{aligned}
\mathrm{I}_{\psi_1} &= \{\psi_1.\mathrm{i}_1, \psi_1, \mathrm{i}_2, \psi_1.\mathrm{i}_3, \psi_1.\mathrm{i}_4\} \\
\mathrm{O}_{\psi_1} &= \{\psi_1.\mathrm{o}_1, \psi_1, \mathrm{o}_2, \psi_1.\mathrm{o}_3, \psi_1.\mathrm{o}_4\} \\
\Gamma_{\psi_1} &= \{\gamma_1, \gamma_2\} \\
\Phi_{\psi_1} &= \{(\gamma_1.\mathrm{i}_1, \psi_1.\mathrm{i}_1), (\gamma_2.\mathrm{i}_1, \psi_1.\mathrm{i}_1), (\gamma_1.\mathrm{i}_2, \psi_1.\mathrm{i}_2), (\gamma_2.\mathrm{i}_2, \psi_1.\mathrm{i}_2), (\gamma_1.\mathrm{i}_3, \psi_1.\mathrm{i}_3), (\gamma_2.\mathrm{i}_3, \psi_1.\mathrm{i}_3), \\
&\quad (\gamma_1.\mathrm{i}_4, \psi_1.\mathrm{i}_4), (\gamma_2.\mathrm{i}_4, \psi_1.\mathrm{i}_4), (\gamma_1.\mathrm{o}_1, \psi_1.\mathrm{o}_1), (\gamma_2.\mathrm{o}_1, \psi_1.\mathrm{o}_1), (\gamma_1.\mathrm{o}_2, \psi_1.\mathrm{o}_2), \\
&\quad (\gamma_2.\mathrm{o}_2, \psi_1.\mathrm{o}_2), (\gamma_1.\mathrm{o}_3, \psi_1.\mathrm{o}_3), (\gamma_2.\mathrm{o}_3, \psi_1.\mathrm{o}_3), (\gamma_1.\mathrm{o}_4, \psi_1.\mathrm{o}_4), (\gamma_2.\mathrm{o}_4, \psi_1.\mathrm{o}_4)\}
\end{aligned}$$

□

## 3.2 Der Spezifikationsgraph

Blickle et al. [2] definieren ein Modell zur Spezifikation von Systemen mit azyklischem Verhalten unter Ressourcenbeschränkungen. Dieses Modell soll hier auf hierarchische Prozessgraphen und hierarchische Architekturgraphen erweitert werden, den sog. *hierarchischen Spezifikationsgraphen*. Als Problemgraphmodell dient hier das ausführlich dokumentierte SPI-Modell (**S**ystem **P**roperty **I**nterval) [9]. Bei dem SPI-Modell handelt es sich um ein hierarchisches Prozessgraphmodell, welches in der Lage ist sowohl reaktives als auch transformatives Verhalten zu modellieren.

**Definition 5 (Hierarchischer Spezifikationsgraph)**
*Ein* hierarchischer Spezifikationsgraph $G_{\mathrm{spec}} = (G_{\mathrm{P}}, G_{\mathrm{A}}, E_{\mathrm{M}})$ *besteht aus:*

- *dem Problemgraphen $G_{\mathrm{P}}$ (z. B. der SPI-Prozessgraph), der das gewünschte Verhalten spezifiziert,*
- *dem nach Definition 4 gegebenen hierarchischen Architekturgraphen $G_{\mathrm{A}}$, der die Architekturvielfalt modelliert, sowie*
- *einer Menge von Abbildungskanten $E_{\mathrm{M}} \subseteq V_{\mathrm{l}}(G_{\mathrm{P}}) \times V_{\mathrm{l}}(G_{\mathrm{A}})$.*

*Die Abbildungskanten $e \in E_{\mathrm{M}}$ stellen eine benutzerdefinierte Abbildungsrelation dar, die die Implementierbarkeit von Blattknoten $v_{\mathrm{P}} \in V_{\mathrm{l}}(G_{\mathrm{P}})$ des Problemgraphen auf den Blattknoten $v_{\mathrm{A}} \in V_{\mathrm{l}}(G_{\mathrm{A}})$ verdeutlicht. Den Abbildungskanten sind eine Menge von Attributen zugeordnet.* □

Wird als Problemgraph das SPI-Modell verwendet, so ist es nicht erlaubt, Prozesse $p \in P_{\mathrm{SPI}}$ auf Busressourcen ($v \in V_{\mathrm{c}}$) des Architekturgraphen abzubilden. Weiterhin können dann die Attribute der Abbildungskanten im Allgemeinen vom Prozess-Modus des zugehörigen Blattknotens des SPI-Modells abhängig sein.

**Beispiel 5** Abbildung 4 zeigt einen Spezifikationsgraphen nach Definition 5. Der Problemgraph (links in Abb. 4) ist hier durch einen SPI-Graphen mit zwei Prozessen ($\mathrm{P}_1$, $\mathrm{P}_2$), drei Kanälen ($\mathrm{C}_1$, $\mathrm{C}_2$, $\mathrm{C}_3$) und einem Interface $\psi_1$ gegeben. Das Interface $\psi_1$ ist wiederum durch zwei Cluster $\gamma_1$ und $\gamma_2$ spezifiziert.

Der Architekturgraph in Abbildung 4 besteht aus drei funktionalen Komponenten ($\mathrm{ASIC}$, $\mathrm{FPGA1}$, $\mathrm{FPGA2}$) und drei Busressourcen ($\mathrm{Bus1}$, $\mathrm{Bus2}$, $\mathrm{Bus3}$). Hierbei ist bei $\mathrm{FPGA2}$ noch nicht klar, welches Design $\mathrm{FPGA2}_1$ oder $\mathrm{FPGA2}_2$ bei der Implementierung zum Einsatz kommt, d. h. $\mathrm{FPGA2}$ wird als ein Interface modelliert. Dies ist durch zwei mögliche Cluster $\mathrm{FPGA2}_1$ und $\mathrm{FPGA2}_2$ angedeutet. Des weiteren sind den Ressourcen Kostenattribute in Form von Intervallen zugeordnet. Diese Attribute geben die Kosten an, die eine Komponente bei ihrer Allokation verursacht.

Weiterhin zeigt Abbildung 4 die Abbildungskanten, die die mögliche Implementierbarkeit von den Blattknoten des Problemgraphen auf die Blattknoten des Architekturgraphen wiedergeben. Den Abbildungskanten sind Ausführungszeiten in Form von Intervallen zugeordnet. Diese geben die Ausführungszeiten eines Knotens des Problemgraphen auf dem zugehörigen Ressource des Architekturgraphen an. Auf die Modellierung von Bindungskosten wurde hier der Übersichtlichkeit halber verzichtet. □

Der in Abbildung 4 gezeigte Spezifikationsgraph modelliert ein rekonfigurierbares System: Auf Grund der Abbildungskanten kann Cluster $\gamma_1$ nur ausgeführt werden, sofern das $\mathrm{FPGA2}$ mit dem Design $\mathrm{FPGA2}_2$ konfiguriert wurde. Das selbe gilt für Cluster $\gamma_2$ und Design $\mathrm{FPGA2}_1$. Da die einzelnen

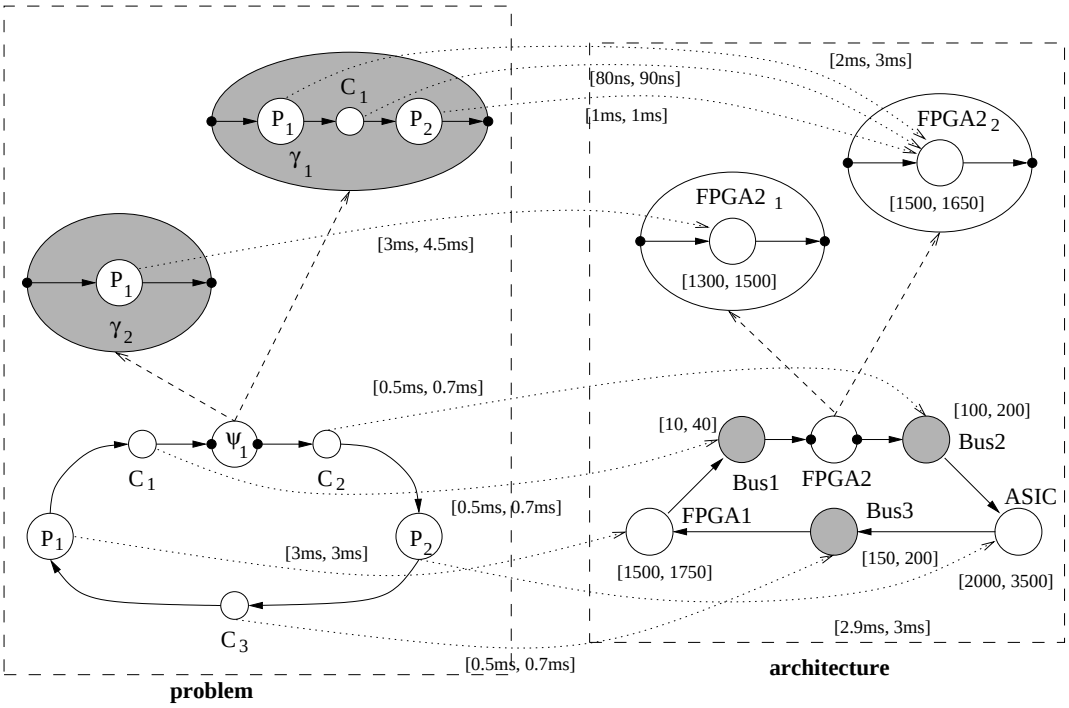


Abbildung 4: Spezifikationsgraph nach Definition 5

Cluster alternative Ersetzungen eines Interfaces darstellen, bedeutet dies, dass das FPGA zur Laufzeit rekonfiguriert werden muss, sofern eine alternative Funktionalität von Knoten $\psi_1$ gefordert wird. In [5] konnte das Synthesemodell nach Blickle et al. [2] auf zeitbehaftete Allokation und Bindung erweitert werden.

Hierin beschreibt eine zeitbehaftete Allokation $\alpha(v,t)$ die Verwendung des Knotens $v$ zum Zeitpunkt $t$ in einer Implementierung. Die zeitbehaftete Bindung $\beta(v,t)$ zeigt an, auf welcher Ressource die Operation $v$ zum Zeitpunkt $t$ ausgeführt wird [6, 7].

Entscheidend hierbei ist, dass die Allokation $\alpha(v,t)$ nicht nur auf die Ressourcen des Architekturgraphen beschränkt ist. Auch die Auswahl von Operationen im Problemgraphen kann hiermit modelliert werden. Wie oben angedeutet bedeutet dies, dass die Änderung der Allokation im Architekturgraphen zur Laufzeit auf Grund einer Allokationsänderung im Problemgraphen das Verhalten eines rekonfigurierbaren Systems modelliert. Auch lassen sich durch Allokationsänderungen im Problemgraphen mit Hilfe dieses Modells adaptive Systeme modellieren. Dies ist offensichtlich, da das Anpassen des Verhaltens auf geänderte Anforderung der Definition eines adaptiven Systems genügen.

# 4 Zusammenfassung und Ausblick

In dieser Kurzfassung wurde ein Modell zur Spezifikation von rekonfigurierbaren und adaptiven Systemen vorgestellt. Dieses Modell ist durch einen hierarchischen Spezifikationsgraphen gegeben. Mit Hilfe des Spezifikationsgraphen ist es möglich, das gewünschte Verhalten, Ressourcenbeschränkungen und Abbildungsmöglichkeiten zu modellieren. Durch die hierarchischen Knoten im Spezifikationsgraphen und deren alternativen Ersetzungen ist es möglich, eine geänderte Auswahl von Operationen im Problemgraphen bzw. von Ressourcen im Architekturgraphen über der Zeit darzustellen. Es wurde erläutert, dass diese Möglichkeit die Grundlage für die Modellierung rekonfigurierbarer Systeme bildet.

In weiterführenden Arbeiten wurde bereits die Anwendbarkeit dieses Modells zur effizienten Entwurfsraumexploration gezeigt [5]. Eine zentrale Frage im Zusammenhang mit diesem Modell ist die der Ablaufplanung. Was bedeutet es die Bearbeitung eines Clusters zu unterbrechen und in welchem Zustand befindet sich der Cluster, wenn er wieder zur Ausführung kommt? Erste Ansätze im Bereich der Ablaufplanung hierarchischer Systeme bietet [1] und [8]. Diese Arbeiten beschränken sich jedoch auf die Ablaufplanung hierarchischer Datenflussgraphen ohne Ressourcenbeschränkungen.

# Literatur


[1] B. Bhattacharya and S. Bhattacharyya. Quasi-static Scheduling of Reconfigurable Dataflow Graphs for DSP Systems. In *Proc. of the International Conference on Rapid System Prototyping*, pages 84–89, Paris, France, June 2000.

[2] T. Blickle, J. Teich, and L. Thiele. System-Level Synthesis Using Evolutionary Algorithms. In Rajesh Gupta, editor, *Design Automation for Embedded Systems*, number 3, pages 23–62. Kluwer Academic Publishers, Boston, January 1998.

[3] Karam S. Chatha and Ranga Vemuri. MAGELLAN: Multiway Hardware-Software Partitioning and Scheduling for Latency Minimization of Hierarchical Control-Dataflow Task Graphs. In *Proc. CODES'01, Ninth International Symposium on Hardware/Software Codesign*, Copenhagen, Denmark, April 2001.

[4] Luis Alejandro Cortés, Petru Eles, and Zebo Peng. Hierarchical Modeling and Verification of Embedded Systems. In *Proc. Euromicro Symposium on Digital Systems Design*, Warsaw, Poland, September 2001.

[5] Christian Haubelt. Modelle und Verfahren zur Hardware/Software-Partitionierung hierarchischer Prozessgraphen. Technical report, Datentechnik, Electrical Engineering Department, University of Paderborn, April 2001. Technical-Report No 01/01.

[6] Christian Haubelt, Jürgen Teich, Kai Richter, and Rolf Ernst. Flexibility/Cost-Tradeoffs in Platform-Based Design. In *SAMOS – Systems, Architectures, Modeling, and Simulation*, 2002. Will be published in Lecture Notes in Computer Science (LNCS), Vol. 2268, Springer.

[7] Christian Haubelt, Jürgen Teich, Kai Richter, and Rolf Ernst. System Design for Flexibility. In *Proc. Design, Automation and Test in Europe (DATE'02)*, Paris, France, March 2002.

[8] Paul Pop, Petru Eles, Traian Pop, and Zebo Peng. An Approach to Incremental Design of Distributed Embedded Systems. In *Proc. 38th IEEE/ACM Design Automation Conference (DAC)*, Las Vegas, U.S.A., June 2001.

[9] K. Richter, D. Ziegenbein, R. Ernst, L. Thiele, and J. Teich. Representation of Function Variants for Embedded System Optimization and Synthesis. In *Proc. 36th Design Automation Conference (DAC'99)*, New Orleans, June 1999.