# Nicht-invasive Simulation des Energieverbrauchs von Hardware-Komponenten auf Systemebene mit SystemC

Daniel Lorenz
OFFIS – Institut für Informatik
Oldenburg
lorenz@offis.de

Philipp A. Hartmann
OFFIS – Institut für Informatik
Oldenburg
hartmann@offis.de

Kim Grüttner
OFFIS – Institut für Informatik
Oldenburg

Achim Rettberg
Carl v. Ossietzky Universität
Oldenburg

**Kurzfassung**

Nicht zuletzt durch die zunehmend wachsende algorithmische Komplexität heutiger eingebetteter Systeme gewinnt die Betrachtung nicht-funktionaler Eigenschaften dieser Systeme, wie beispielsweise des Energieverbrauchs, immer stärker an Bedeutung. Insbesondere in frühen Entwurfsphasen, lange vor der Fertigstellung der finalen Hardwareplattform, werden daher Methoden und Werkzeuge zur Analyse und Abschätzung der Leistungsaufnahme dringend benötigt.

In dieser Arbeit wird ein Simulationsframework basierend auf SystemC vorgestellt, welche die Anreicherung von bereits existierenden, funktionalen TLM-2.0-Modellen um solche nichtfunktionalen Eigenschaften unterstützt. Um auch eine Betrachtung externer IP-Komponenten zu ermöglichen, kann der hier präsentierte Ansatz ohne Veränderung der einzelnen Blöcke verwendet werden. Der (im Detail) unbekannte interne Zustand (und der daraus resultierende Energieverbrauch) der Komponente wird dabei über eine sogenannte *Power State-Machine* (PSM) angenähert. Die Bestimmung des aktuellen Power States erfolgt über eine separat definierte *Protocol State-Machine* (PrSM), welche durch die nicht-invasive Beobachtung der Interaktion des Hardwareblocks mit seiner Umgebung Rückschlüsse auf das aktuelle Verhalten der Komponente erlaubt. Zur Evaluation der Methodik werden unterschiedliche Komponenten beispielhaft modelliert und analysiert.

## 1. Einleitung

Während die Komplexität der Anwendungen ebenso wie die Leistungsfähigkeit der verfügbaren Berechnungseinheiten in heutigen eingebetteten Systemen stetig zunimmt, kann die Entwicklung der Energiedichte von Akkus und Batterien nicht im gleichen Maße Schritt halten. Dies kann – gerade bei mobilen Systemen – zu einer reduzierten Systemlaufzeit führen, wenn nicht geeignete Maßnahmen zur Energieeinsparung getroffen werden. Da jedoch das Gesamtsystem bereits sehr komplex ist, muss mit der Integration solcher Techniken bereits in frühen Entwurfsphasen begonnen werden, lange bevor die endgültige Hardware-Plattform zur Verfügung steht. Dazu ist es nötig, den erwarteten dynamischen Energieverbrauch des Systems in Abhängigkeit der tatsächlichen Nutzung des Systems zum Beispiel in Simulationsläufen sichtbar und analysierbar zu machen.

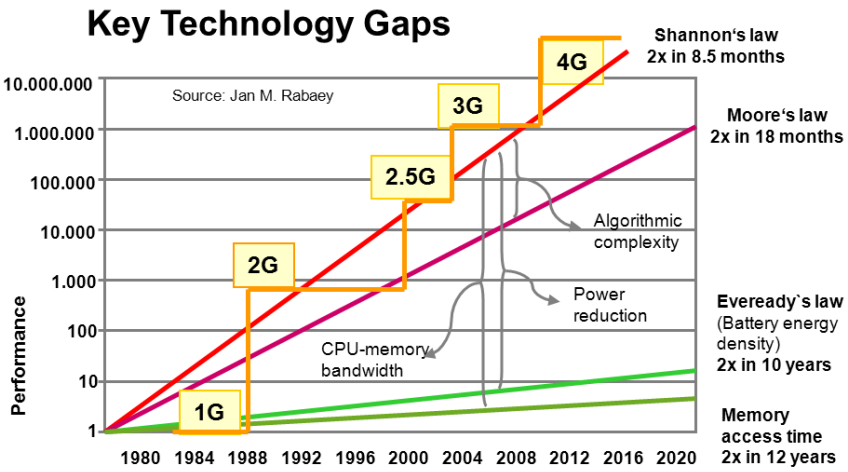


**Abb. 1:** Technologiefortschritt in verschiedenen Bereichen

Dies beinhaltet neben dem Energieverbrauch der verwendeten Prozessorkerne (und ihrer Stromsparmöglichkeiten) auch den Einfluss weiterer Hardwarekomponenten in der Plattform (z.B. Busse, Speicher, dedizierte Hardwarebschleuniger). In zunehmendem Maße werden solche Komponenten heutzutage bei der Entwicklung eines neuen Systems nicht von Grund auf neu entwickelt, sondern entweder hinzugekauft oder aus vorausgegangenen Designs wiederverwendet. Anhand der dann verfügbaren Datenblätter können Systemarchitekten dann den Energieverbrauch in Abhängigkeit der Zieltechnologie statisch abschätzen. Dies berücksichtigt aber keinerlei dynamische Effekte, beispielsweise durch unterschiedliche Nutzungsszenarien, welche verschiedene Lastprofile und dementsprechend Energieanforderungen mit sich bringen.

Um andererseits die komplexen Anwendungen auf heutigen Plattformen zu simulieren, sind abstrakte und hochperformante Simulationsmodelle nötig, welche zum Beispiel frühzeitig die Softwareentwicklung auf der noch nicht vollständig verfügbaren Hardwareplattform ermöglichen. Zu diesem Zweck wird in der Industrie vermehrt die Systembeschreibungssprache SystemC [6] eingesetzt. Abstrakte Modelle nutzen dabei häufig das sogenannte *Transaction-Level Modelling* (TLM) und insbesondere die standardisierte *TLM-2.0*-Erweiterung [8] von SystemC zur Beschreibung von Plattformen die durch Memory Mapped I/O über einen gemeinsamen Bus kommunizieren.

Um die Energieabschätzung des Gesamtsystems bereits auf hohen Abstraktionsebenen zu ermöglichen, wird im Rahmen des europäischen Forschungsprojekts COMPLEX [2] derzeit ein ganzheitlicher Ansatz zur (Energie-)Abschätzung und Exploration komplexer Systeme basierend auf virtuellen Plattformen in SystemC TLM-2.0 entwickelt [5]. Im Rahmen dieser Arbeit wird nun ein Teilkonzept hieraus zur nicht-invasiven Simulation des Energieverbrauchs von Systemkomponenten vorgestellt. *Nicht-invasiv* bedeutet hier, dass der interne Zustand und der darauf basierende Energieverbrauch einer Systemkomponente allein anhand der beobachtbaren Interaktion mit der Außenwelt angenähert und aufgezeichnet werden soll. Dies ist notwendig, um auch bereits existierende, unveränderliche IP-Komponenten zu berücksichtigen.

Abschnitt 2 gibt einen Überblick über bisherige Ansätze zur Modellierung des Energieverbrauchs in SystemC. Die im Rahmen dieser Arbeit entwickelte Methode zur Beschreibung und Simulation sogenannter *Power-State-Machines* (PSMs) wird in Abschnitt 3 vorgestellt. Mit Hilfe der PSM wird der Energieverbrauch (bzw. die Aktivität) einer Komponente für jeden abstrakten Zustand definiert. Um die Übergänge zwischen diesen Zuständen zu detektieren, wird die vorge-

gebene Komponente an ihrer Schnittstelle durch spezielle Adapter, sog. *Protocol-State-Machines* (PrSM), gekapselt und die Kommunikation mit der Umgebung beobachtet. In Abschnitt 4 wird das Energieverhalten unterschiedlicher Komponenten beispielhaft modelliert und simuliert. Abschnitt 5 schließt mit einer Zusammenfassung und gibt einen Ausblick auf zukünftige Arbeiten.

## 2. Verwandte Arbeiten

In der Vergangenheit wurden viele Methoden zur Energieabschätzung auf verschiedenen Ebenen erforscht. Es hat sich gezeigt, dass frühe Designentscheidungen auf Systemebene das größte Potential für Energieeinsparungen besitzen [10]. In [1] wurde ein erster Ansatz vorgestellt, bei dem jede Komponente im System durch eine Power State Machine modelliert wird. Zustandsübergänge werden durch Eingaben von einem Power Manager kontrolliert. Ein Zustand kann anstatt eines bestimmten Energieverbrauchs auch einen maximalen Energieverbrauch haben, dessen Aktivitätslevel durch den Power Manager kontrolliert wird. Kosten für die Zustandsübergänge können durch Verzögerungen modelliert werden.

Da dynamischer Energieverbrauch durch Aktivität entsteht, wird in [10] ein Konzept vorgestellt, bei dem die Aktivität an den Kommunikationsschnittstellen beobachtet wird. Ein eigener Prozess wird durch alle normalen Events der Kommunikationskanäle benachrichtigt und zeichnet die Aktivität auf. Diese Methode ist zwar sehr transparent und erzeugt wenig Overhead, vernachlässigt aber die Aktivität im Innern der Komponenten, die sich durchaus anders verhalten kann als die Aktivität auf den Kommunikationskanälen.

Mit *Powersim* wurde kürzlich in [3] ein weiterer Ansatz vorgestellt. Hier muss der Quelltext der zu untersuchenden Komponenten nicht verändert werden, da ein entsprechend angepasster SystemC-Kernel verwendet wird. Das Power-Modell des Systems wird dabei zu Beginn der Simulation geladen anhand dessen dann bei allen Operationen auf den SystemC-Datentypen eine Aktivitätsabschätzung durchgeführt wird. Aufgrund des modifizierten SystemC-Kernels lässt sich dieser Ansatz nicht in industrielle Entwurfsumgebungen integrieren, die einen eigenen und nicht austauschbaren SystemC-Kernel mitbringen. Zudem ist eine Abschätzung der Kommunikation, insbesondere auf abstrakteren Modellen (z.B. TLM), nur eingeschränkt möglich, weil lediglich der Zuweisungsoperator für die Power-Annotation verwendet werden kann.

Es hat sich gezeigt, dass eine Modellierung mittels Zustandsautomaten mit Adaption an Umgebungsparameter eine gute Möglichkeit ist, den Energieverbrauch abzuschätzen. In [7] wird dieser Ansatz in SystemC/TLM benutzt, um an entsprechenden Stellen im Modul durch einen Methodenaufruf einen Zustandswechsel des Zustandsautomaten herbeizuführen. Dabei wird der Energieverbrauch auch dem Dynamic Voltage and Frequency Scaling (DVFS) angepasst. Der Nachteil ist, dass diese Methode invasiv ist und somit bei IP-Komponenten nicht eingesetzt werden kann.

In [9] wird ein Konzept vorgestellt, bei dem Architektur und Funktionalität voneinander getrennt werden. Das Anwendungsmodell wird auf das Plattformmodell abgebildet, welches ein Modell des Energieverbrauchs beinhaltet. Durch Konfigurationsdateien können verschiedene Varianten ausgewertet werden, die nur die Abbildung des Anwendungsmodells auf das Plattformmodell ändern. Verschiedene Power States können mit einer Komponente verknüpft werden. Im Unterschied zu unserem Ansatz handelt es sich bei dem Anwendungsmodell um einen aktorbasierten Ansatz, der ein dynamisches Datenflussmodell implementiert. Damit funktioniert dieser Ansatz nur für Komponenten, die durch einen solchen Aktor beschrieben wurden. Der in dieser Arbeit vorgestellte Ansatz funktioniert hingegen mit jeder Black-Box IP-Komponenten, die über Memory Mapped

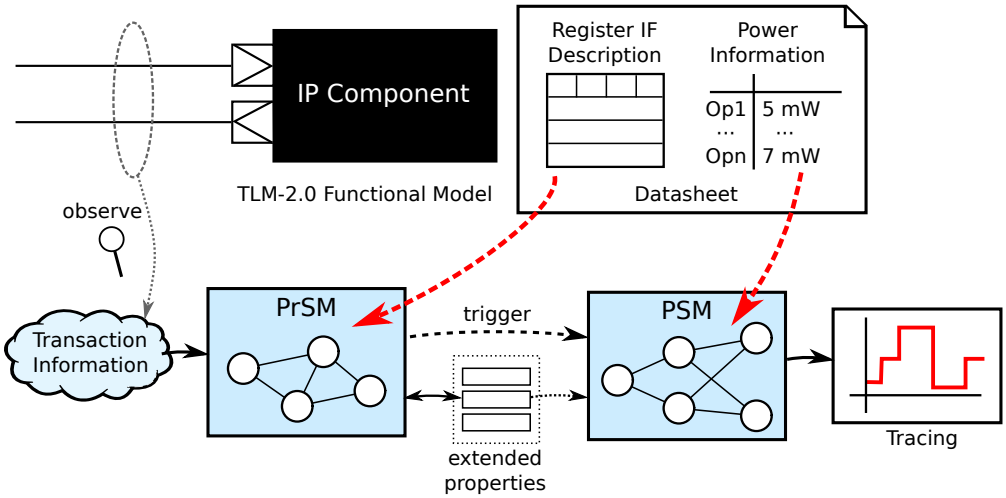


**Abb. 2:** Übersicht über PSM-Ansatz zur nicht-invasiven Energiesimulation

I/O angesprochen werden kann. Mit Hilfe von [4] ließen sich die dynamischen Datenflussmodelle aus [9] mit unserem Ansatz kombinieren.

## 3. Power State-Machines in SystemC

Eine wesentliche Anforderung an den hier vorgestellten Ansatz ist die Abschätzung von sogenannten Black-Box-Komponenten, bei denen nur die Schnittstellen nicht aber der innere Aufbau bekannt ist. Insbesondere kann das Innere der Komponente nicht verändert werden, um den Energieverbrauch bzw. die Aktivität aufzuzeichnen. Daher wird der innere Zustand der IP-Komponente im Folgenden allein durch die Beobachtung der Interaktion mit dieser abstrahiert. Die resultierende Schaltaktivität in den verschiedenen Makrozuständen kann dann aus Datenblättern stammen, aus der Größe des Designs und der funktionalen Komplexität abgeleitet werden (top-down), oder aus Simulationen auf niedrigerer Ebene entnommen werden (bottom-up). Mit letzterer Variante kann ein genaueres Modell aufgebaut werden, vgl. [7].

In Abb. 2 ist ein Überblick des Ansatzes zur nicht-invasiven Integration von Energieinformationen dargestellt. Es wird angenommen, dass die zu untersuchende Black Box-Komponente als funktionales Modell mit TLM-2.0 Interface vorliegt. Die Registerschnittstelle, d.h. die logische Bedeutung der in den globalen Adressraum eingeblendeten Register, der IP-Komponente und das zugehörige Verhalten sind bekannt. Außerdem liegen zu verschiedenen Operationsmodi der Komponente Energieabschätzungen, beispielsweise als mittlere Leistungsaufnahmen bei gegebener Technologie, vor.

Basierend auf diesen Informationen kann nun anhand der Energiecharakteristika das dynamische Power-Modell der Komponente mittels einer PSM erstellt werden. Zur Bestimmung der dynamischen Verlustleistung $P(t)$ als Funktion der Zeit $t$ kann bei gegebener gesamter geschalteter Kapazität $C$, Versorgungsspannung $V_{dd}$ und Taktfrequenz $f$ ausgehend von der momentanen Schaltaktivität $\alpha(t) \in [0, 1]$ die folgende Formel aus der Literatur verwendet werden[1]:

$$P(t) = \frac{1}{2} C V_{dd}^2 f \cdot \alpha(t) \tag{1}$$

[1]Zur Vereinfachung werden hier zunächst statische Werte für Spannung und Frequenz angenommen.

$C$ beschreibt dabei die Gesamtkapazität der Komponente und ist stets konstant und $\alpha(t) = \{x \mid 0 \leq x \leq 1\}$ bezeichnet die Aktivität, die den Anteil der geschalteten Gesamtaktivität angibt. Da die tatsächliche Schaltaktivität $\alpha(t)$ auf Systemebene, und insbesondere bei IP-Komponenten nicht im Detail bekannt ist, wird in der PSM eine diskrete Zustandsmenge $p_1, \ldots, p_n$ zur Unterscheidung charakteristischer Operationsmodi definiert. Ein solcher Zustand $p_i$ ist dabei maßgeblich durch die *mittlere Schaltaktivität* $\alpha_i = \overline{\alpha(p_i)}$ definiert. Mit Hilfe dieser (vom Designer zu wählenden) Parametern $p_i$ und $\alpha_i$ kann dann aus Gleichung 1 die dynamische Leistungsaufnahme der Komponente approximiert werden, wenn der aktuelle Zustand $p(t) \in \{p_1, \ldots, p_n\}$ bekannt ist:

$$P(t) \approx \frac{1}{2} C V_{dd}^2 f \cdot \alpha\left(p(t)\right) \tag{2}$$

Gleichung 2 bildet hier die Ausgabe der PSM und somit die Basis für die Aufzeichnung der Energieaufnahme während der Simulation. Dabei wird angenommen, dass $V_{dd}$ und $f$ durch einen Powermanager dynamisch verändert werden kann. Das so abstrahierte Energieverhalten der Komponente muss nun durch geeignete Stimuli (`psm_event`s) der Übergänge zwischen den Power-Zuständen $p_i$ angeregt werden. Um dies zu erreichen muss der aktuelle funktionale Zustand der Komponente approximiert werden.

Da wir uns hier zunächst auf registerbasierte Schnittstellen beschränken, wird für die Annäherung der inneren Funktionalität diese bekannte, externe Schnittstelle über die Zeit beobachtet und anhand dessen eine *Protocol State Machine* (PrSM) gesteuert. Die wesentliche Aufgabe der PrSM ist die Generierung der Eingaben für die PSM anhand der beobachteten Interaktion der Komponente mit der Außenwelt. Die PrSM extrahiert also die energetisch relevanten Ereignisse, um Kommunikations- und Funktionalitätsartefakte von der rein nicht-funktionalen PSM zu orthogonalisieren. Dies kann zu einer erheblichen Komplexitätsreduktion in der PSM führen, da diese lediglich die verschiedenen und Aktivitätsmodi beschreibt wohingegen die PrSM den internen Zustand und das Verhalten mit der Umwelt behandelt. Des Weiteren bietet die Trennung von PrSM und PSM den Vorteil, dass bei Komponenten mit gleichem Zugriffsprotokoll und unterschiedlichen internen Realisierungen ausschließlich die PSM verändert werden muss und die PrSM wiederverwendet werden kann.

Als Eingabe für die PrSM dienen dabei die beobachtbaren Informationen aus den Transaktionen auf den TLM-2.0-Sockets der Black Box-Komponente. Je nach Registerschnittstelle und Semantik der Lese/Schreib-Zugriffe auf bestimmten Adressen werden Aktivitäten oder sogar komplexe Operationen in der beobachteten Komponente ausgelöst. Dies kann durch Übergänge zwischen Zuständen in der PrSM nachgebildet werden, um die funktionalen Charakteristika zu abstrahieren. Solche Zustandsübergänge in der PrSM können wiederum zu Notifizierungen der PSM führen. Eine detaillierte Beschreibung der Protokollbeobachtung basierend auf TLM-Transaktionen und der Beschreibung der sich daraus ergebenden Übergängen in der PrSM erfolgt in Abschnitt 3.2.

Da einige Zustandswechsel innerhalb der IP-Komponente nicht durch Interaktion mit der Außenwelt detektiert werden können (z.B. dass die Berechnung eines Ergebnisses bereits beendet ist), können in beiden Zustandsautomaten für jeden Zustand optionale Timeouts definiert werden. Das bedeutet, dass der Automat selbstständig in einen definierten Nachfolgezustand wechselt, falls vor Ablauf dieses Timeouts kein expliziter Übergang von außen angeregt wurde.

Zu guter Letzt ist es möglich, explizite Zustands*variablen* ($Z(t)$, *extended properties*) zur flexibleren und eventuell sogar datenabhängigen Zustandsmodellierung hinzuzufügen (vgl. Abb. 2). Diese Zustandsvariablen werden von der PrSM verwaltet und können zum Beispiel zur Speiche-

rung spezifischer Konfigurationsdaten verwendet werden, etwa aktuelle Funktionsauswahl (`MD5` vs. `SHA256` bei einem Hashingmodul). Dies kann die Zahl der expliziten Zustände in PrSM und PSM reduzieren, insbesondere wenn die logische Zustandsfolge unverändert bleibt. Da die Ausgabe der PSM ebenfalls von diesen Attributen abhängen kann, erweitert sich die Berechnung der Aktivität zu $\alpha\left(p(t), Z(t)\right)$ in Gleichung 2. Zustandsübergänge, Timeouts, und Ausgaben der beiden Automaten können abhängig von den Zustandsvariablen sein. Wird zum Beispiel die aktuelle Taktfrequenz des Moduls über ein solches Attribut beschrieben, so kann ein Timeout linear verlängert werden, wenn die Taktfrequenz sinkt.

### 3.1. Integration in das Simulationsmodell

Um nun die abstrakten Automaten aus dem vorigen Abschnitt in einer Gesamtsimulation zu integrieren, wurde eine PSM-Simulationsbibliothek basierend auf SystemC entwickelt. Dabei wird die zu untersuchende Black Box-Komponente zunächst in einen Wrapper mit identischem Interface nach außen gekapselt, sodass diese erweiterte Komponente transparent integriert werden kann.

Für den Wrapper gibt es eine Basisklasse **`psm_wrapper`**`<T>`, der als Templateparameter der Typ der IP-Komponente übergeben wird. Diese Basisklasse stellt die nötige Infrastruktur für die PrSM, PSM, und die Zustandsvariablen zur Verfügung. Zur Beobachtung der Kommunikation verwendet der Wrapper speziell erweiterte TLM-2.0-Sockets, welche die Transaktionen unverändert an die umschlossene IP-Komponente weiterleiten und mit speziellen Beobachtungspunkten angereichert sind (*observable sockets*). Da die Struktur der zu erzeugenden Komponente regulär aus der Struktur der IP-Komponente ableitbar ist, ist eine automatische Generierung möglich:

**Listing 1:** Implementierung eines IP-Wrappers für einen Speicher

```
SC_MODULE(memory) { tlm_target_socket<32> socket; /* ... */ };

struct wrapped_memory : psm_wrapper<memory>
{
  tlm_observable_target_socket<32> socket;
  SC_CTOR(wrapped_memory)
    : PSM_WRAP_SOCKET(socket)
  { /* ... */ }
};
```

In der derzeitigen Implementation werden die Zustände und Übergänge der Automaten explizit im Innern des Wrappers erzeugt. Das Design erlaubt allerdings eine dynamische Generierung basierend auf einer zur Elaborationszeit eingelesenen Konfiguration.

### 3.2. Approximation der Funktionalität durch Protokollbeobachtung

Unter der Voraussetzung, dass die zu untersuchende Komponente mit Hilfe des TLM-2.0 *Base Protocol* [8] mit der Außenwelt kommuniziert, wird eine Reihe von Informationen von den *observable sockets* als Eingabe für die PrSM extrahiert und in einem speziellen **`tlm_transaction_info`**-Objekt zur Auswertung in den Übergangsbedingungen bereitgestellt. Es kann sowohl der Loosely-Timed (LT), als auch der Approximately-Timed (AT) Programmierstil verwendet werden. Die aktuelle Simulationszeit wird ebenso wie ein etwaiger lokaler Versatz (im Sinne des temporal decoupling) an die PrSM weitergeleitet.

Die zur Verfügung stehenden Informationen zu der aktuell aktiven Transaktion beinhalten die folgenden Attribute:

- Generic Payload (R/W, Adresse, Daten, Länge, ...)
- Aktuelle Phase/Timing-Point der Transaktion (TLM-2.0 LT/AT base protocol)
- Verwendete Schnittstelle der Komponente (Socket)
- Rolle der Komponente in der Transaktion (Initiator, Target oder Interconnect)

Zur Modellierung der PrSM kann man die Kommunikation mit der Komponente auf mehreren Protokollebenen betrachten. Zunächst ist die reine Schreib/Lese-Operation im Adressbereich einer Komponente zu nennen (*Transport Layer*, einzelne TLM-2.0 Transaktionen). Bei einer reinen Speicherkomponente ist diese Ebene meist bereits ausreichend um das Energieverhalten zu beschreiben, da keine logische Beziehung über mehrere Transaktionen hinweg besteht.

Bei komplexeren Komponenten (z.B. dedizierte Hardwarebeschleuniger) können Seiteneffekte im Innern der Zielkomponente auch über mehrere Transaktionen hinweg entstehen (*Presentation Layer*). Konzeptionell erweist es sich dann als nützlich, den Adressbereich der Komponente in Kontroll-, Konfigurations- und Datenadressen zu unterscheiden. Dafür wird die semantische Beschreibung der Registerschnittstelle benötigt, um eine Abstraktion der Funktionalität mit Hilfe von geeigneten Zuständen und Zustandsvariablen in einer PrSM zu beschreiben.

*Kontrolldaten* beeinflussen den Protokollfluss und dementsprechend unmittelbar den inneren Zustand der Komponente. Ein Beispiel dafür ist ein explizites Handshake-Register zur Synchronisation mit einem Hardwarebeschleuniger. Transaktionen auf solchen Registern führen in der Regel zu expliziten Zustandsübergängen in der PrSM. Als *Konfigurationsdaten* können solche Register angesehen werden, welche das Verhalten einer Komponente in nachfolgenden Zuständen oder über einen längeren Zeitraum beeinflussen. Ein Beispiel könnte die Auswahl des zu verwendenden Algorithmus für einen Verschlüsselungs-Koprozessor sein. Die Werte von Konfigurationsregistern erfordern oft die Einführung (und Beobachtung) einer entsprechenden Zustandsvariable in der PrSM. *Nutzdaten* wiederum werden zwar von der IP-Komponente verarbeitet, haben aber keinen Einfluss auf das Protokoll oder den internen (beobachtbaren) Kontrollfluss der Komponente. Dies könnten beispielweise die zu verschlüsselnden Daten des o.g. Beschleunigers oder schlicht (lokale) Speicher beinhalten. Obwohl die konkret verwendeten Daten auch einen Einfluss auf den Aktivität einer Komponente haben können, wird dies hier vernachlässigt.

Die Zustandsübergänge in den Automaten werden mit Bedingungen verknüpft. Diese Bedingungen werten die Eingabedaten des Automaten aus, indem sie diese an ein vom Benutzer definiertes Funktionsobjekt weiterleiten. Dabei können entweder einfache C-Funktionen genutzt werden oder komplexe Funktoren, die einen inneren Zustand haben und parametrisiert werden können. Listing 2 zeigt einen solchen Funktor, der die aktuelle Transaktion auf eine bestimmte Startadresse prüft. Komplexere Bedingungen können aus solchen Basisfunktionen durch logische Verknüpfungen definiert werden. Ist nun eine Bedingung erfüllt, so führt die PrSM einen Zustandswechsel durch und kann zusätzlich eine Notifizierung an die PSM weiterleiten. Diese reagiert darauf, indem sie einen Zustandswechsel durchführt, falls es einen passenden Zustandsübergang gibt.

## 4. Evaluation

Zur Evaluation des zuvor vorgestellten Konzepts wird die nicht-invasive PSM-Methodik in diesem Abschnitt auf zwei IP-Komponenten angewendet. Dabei handelt es sich um ein Speichermodul und ein FFT-Modul. Beide Komponenten sind TLM-2.0 Targets.

**Tabelle 1:** Verarbeitungszeit und Aktivität des Speichers beim Lesen und Schreiben

| | Schreiben | Lesen |
|---|---|---|
| Aktivität | 1 | 0.71 |
| Verarbeitungszeit | $\frac{20\,\mathrm{ns}}{\mathrm{Wort}}$ | $\frac{20\,\mathrm{ns}}{\mathrm{Wort}}$ |

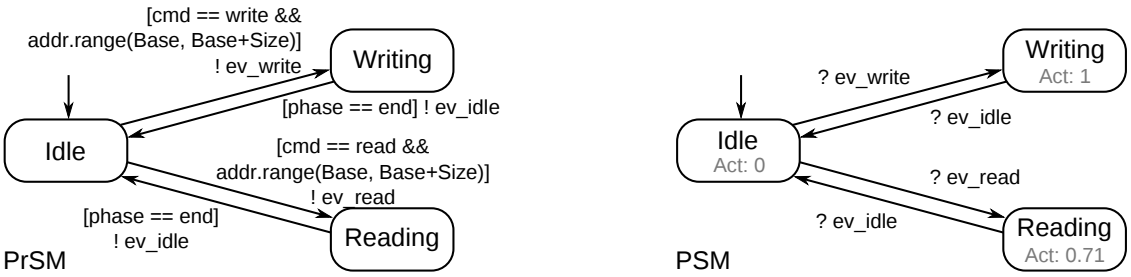


**Abb. 3:** PrSM und PSM des Speichermoduls

Die für das Speichermodell angegebenen Verarbeitungszeiten und Aktivitäten sind in Tabelle 1 angegeben und wurden folgendermaßen ermittelt. Es wurde ein SystemC-Speichermodell nach Gatterebene synthetisiert. Dieses wurde mit mehreren Schreib- und Leseoperationen simuliert und basierend darauf eine Abschätzung des Energieverbrauchs gemacht. Daraus ergab sich bei einer Spannung von $1\,\mathrm{V}$ und einer Frequenz von $50\,\mathrm{MHz}$ im Mittel ein Energieverbrauch von $0{,}7\,\mu\mathrm{W}$ beim Schreiben und $0{,}5\,\mu\mathrm{W}$ beim Lesen. Dabei wurde von der statischen Verlustleistung und der durch den Takt verursachten dynamischen Verlustleistung abstrahiert. Die Verarbeitungszeit pro Word dauert bei bei beiden Operationen einen Takt, wird also als $20\,\mathrm{ns}$ angenommen, obwohl die effektive Verarbeitungszeit natürlich deutlich geringer ist. Die Werte für den Energieverbrauch werden nun abstrahiert von Spannung, Frequenz und Kapazität und ergeben die jeweilige Aktivität. Da die genaue Kapazität nicht bekannt ist, wird für den höchsten Energieverbrauch angenommen, dass dort die gesamte Kapazität geschaltet wird. Damit ergibt sich für die Schreibvorgänge eine Aktivität von 1 und für die Lesevorgänge eine Aktivität von 0,71.

Die PrSM und PSM des Speichers sind in Abb. 3 dargestellt. Die Bedingungen der für die Aktivierung eines Übergangs der PrSM sind in `[...]` dargestellt. Die PrSM aktiviert Zustandsübergänge in der PSM mittels Events (abstrakte Kommandos). Das Auslösen eines Events ist mit `!<event-name>` bezeichnet. Die Sensitivität auf ein Event in der PSM ist mit `?<event-name>` bezeichnet. In der PrSM wird beim Schreiben innerhalb des Adressbereichs des Speichers in den Zustand `Writing` gewechselt und das Event `ev_write` an die PSM gesendet. Daraufhin wechselt die PSM vom Zustand `Idle` nach `Writing`. Mit Beendigung der Kommunikationsphase wechselt

**Listing 2:** Test auf Zugriff auf eine dedizierte Adresse

```
struct match_address {
  sc_dt::uint64 address;
  explicit match_address( sc_dt::uint64 a = 0x0 ) : address(a) {}
  bool operator()( tlm_transaction_info & info
                 , psm_property_list &     /* unused */  )
  { return info.address() == address; }
};
PSM_REGISTER_FUNCTOR(match_address);
```

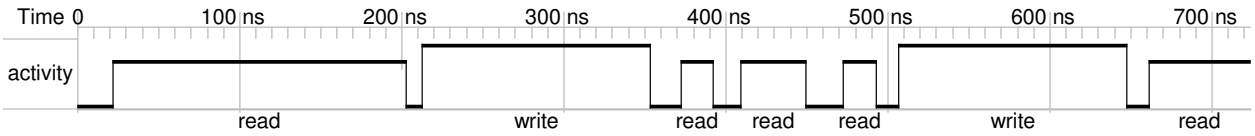


**Abb. 4:** Aktivitätaufzeichnung der Simulation des Speichermoduls

**Tabelle 2:** Registerinterface der FFT-Komponente

| relative Adresse | 0x00 | 0x04 | [0x08, 0x27] | [0x28, 0x47] |
|---|---|---|---|---|
| Zugriff | Wr | Rd | Wr | Rd |
| Funktion | Kontrolle | Kontrolle | Daten | Daten |
| Bedeutung | Operanden gültig | Ergebnis gültig? | Operanden | Ergebnis |

die PrSM zurück in den `Idle`-Zustand, was die PSM synchron nachvollzieht. Für Leseoperationen aus dem Speicher erfolgt das Zusammenspiel von PrSM und PSM analog.

Abb. 4 zeigt einen Aktivitätstrace der zur Speicher-Komponente gehörenden PSM. Dieser Trace wurde durch die Simulation von Speicher-IP und Power-Modell zusammen mit einem TLM-Initiator (z.B. Prozessor) für verschieden Schreib- und Leseoperationen mit einer unterschiedlichen Anzahl an Worten pro Zugriff erstellt.

Die FFT-Komponente berechnet eine diskrete Fourier-Transformation der Größe 8 und weist im Gegensatz zur Speicher-Komponente ein Interface mit Registern aus allen in Abschnitt 3 unterschiedenen Kategorien auf. Die Interpretation des Registerinterfaces ist in Tabelle 2 gezeigt. Die Wortbreite beträgt vier Byte und die Adressierung des Registerinterfaces erfolgt in Bytegranularität.

Die Aufgabe der PrSM zu der FFT-Komponente ist die Interpretation des Zugriffsprotokolls auf dem Registerinterface der Komponente. Diese erfolgt durch eine Beobachtung der Aktivitäten auf den in Tabelle 2 angegebenen Adressbereichen in Kombination mit den Phasen der zugehörigen Transaktionen. Das in diesem Beispiel angenommene Zugriffsprotokoll zur Benutzung der FFT-Komponente sieht folgendermaßen aus:

1. Schreibe alle acht Operanden sukzessive in den Adressbereich Base+0x08 bis Base+0x24.
2. Schreibe Wert 1 an Adresse Base+0x00, um den Beginn der Berechnung zu triggern.
3. Frage Adresse Base+0x04 ab, ob das Ergebnis vorliegt (Wert == 1).
4. Lese Ergebnis aus dem Adressbereich Base+0x28 bis Base+0x44.

Abb. 5 zeigt die zu dem Zugriffsprotokoll gehörige PrSM, sowie die von ihr getriggerte PSM. Der Zustandsübergang von `Calculate` nach `Result calculated` ist hier ein "Timeout"-Übergang, der nach einer Verzögerung von 11 Takten (`delay(11)`) erfolgt. Dabei sei angenommen, dass die

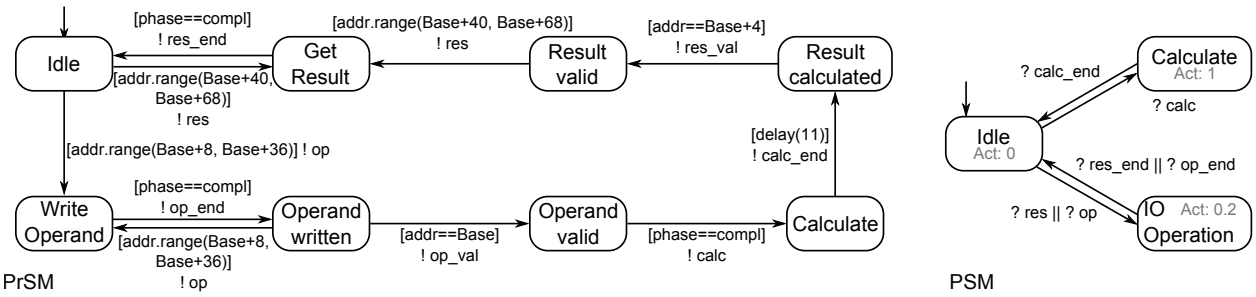


**Abb. 5:** PrSM und PSM der FFT-Komponente

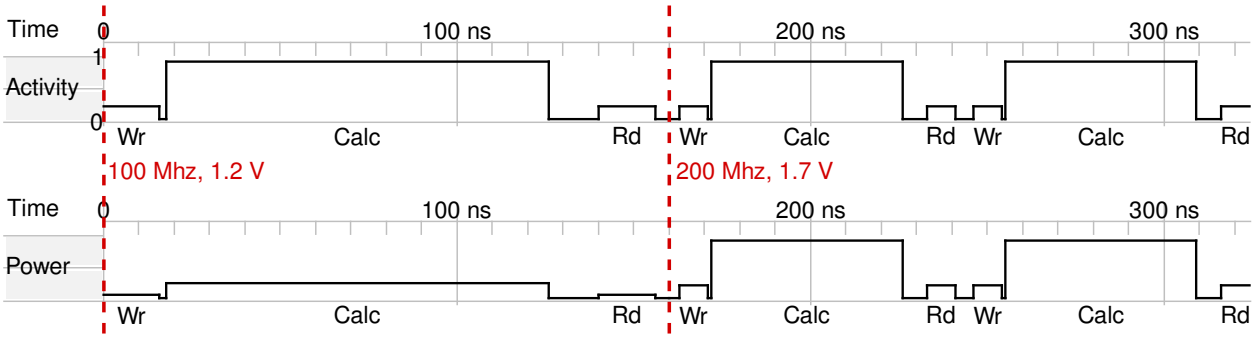


**Abb. 6:** Aktivität und Leistungsaufnahme der FFT-Komponente mit DVFS

FFT-Komponente nach 11 Takten ein Ergebnis liefert. Die Zeit zwischen dem Zustandsübergang berechnet sich dynamisch aus der Taktfrequenz $f$ mit der die FFT betrieben wird: $\mathrm{timeout} = 1/f \cdot \mathrm{delay}$. Bei den in der PSM annotierten Aktivitäten handelt es sich um experimentell ermittelte mittlere Aktivitäten der zugrundeliegenden FFT-Komponente. Diese wurden nach dem gleichen Verfahren berechnet, wie bereits oben beim Speicher beschrieben.

Abb. 6 zeigt den Trace eines exemplarischen Aktivitätsprofils der FFT-Komponente. Für die FFT-Komponente ist der Effekt des Umschaltens von Versorgungsspannung und Taktfrequenz (DFVS) bei $160\,\mathrm{ns}$ gezeigt. Hier ist zu sehen, dass sich bei einer Verdoppelung der Taktfrequenz die Aktivitätsdauer halbiert. In der $P(t)$ Darstellung ergibt sich die dynamische Verlustleistung nach Gleichung 2.

## 5. Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein Konzept und eine Implementation einer nicht-invasiven Simulation des Energieverbrauchs von Hardware-Komponenten auf Systemebene vorgestellt. Basierend auf zwei kommunizierenden Zustandsautomaten (Protokoll- und Power State Machine) wird durch Beobachtung der Interaktion des funktionalen Modells der Komponente über ihr TLM-2.0 Interface der innere Zustand und darauf basierend die Schaltaktivität approximiert. Die Protokoll State Machine abstrahiert den komplexen Zustandsraum des inneren Zustands der IP-Komponente. Diese triggert die Power State Machine, die den aktuellen Operationsmodus in Verbindung mit der aktuellen Aktivität anzeigt. Zustandsübergänge können nicht nur durch Eingabedaten sondern auch durch Timeouts ausgelöst werden.

Zur Erweiterung bzw. Vereinfachung der Zustandsautomaten dienen Zustandsvariablen, die über einen längeren Zeitraum gültige Konfigurationsdaten speichern. Eine spezielle Zustandsvariable wird für die aktuelle Frequenz eingesetzt, wodurch sich die Länge der Timeouts proportional ändern kann. Zusätzlich erlaubt die hier vorgestellte Methodik und Umsetzung eine dynamische Anpassung der Energieverbrauchsausgabe an Veränderungen der Frequenz und Versorgungsspannung durch DFVS.

Die Anwendbarkeit wurde an zwei Beispielen (Speicher-Modell, FFT-Beschleuniger) gezeigt. Zusätzlich wurden die Ergebnisse für zufällige Stimuli-Daten zu diesen Beispielen erläutert.

In zukünftigen Arbeiten soll die erreichbare Genauigkeit durch einen Vergleich mit Simulationen auf niedrigeren Abstraktionsebenen evaluiert werden. Des Weiteren ist eine Erweiterung um hierarchische PSMs vorgesehen, um eine einfachere Integration von dynamischem Powermanagement zu ermöglichen.

## Literatur


[1] Benini, L., R. Hodgson und P. Siegel: *System-level power estimation and optimization*. In: *International Symposium on Low Power Electronics and Design*, ISLPED'98, S. 173–178, Monterey, CA, USA, Aug. 1998. ACM, ISBN 1-58113-059-7.

[2] COMPLEX (ICT FP7 247999): *COdesign and power Management in PLatform-based design space EXploration*. Projekt-Webseite. `http://complex.offis.de`.

[3] Giammarini, M., S. Orcioni und M. Conti: *Powersim: Power Estimation with SystemC*. In: *Solutions on Embedded Systems*, Bd. 81 d. Reihe *Lecture Notes in Electrical Engineering*, S. 285–300. Springer Netherlands, ISBN 978-94-007-0638-5.

[4] Gladigau, J., A. Gerstlauer, C. Haubelt, M. Streubühr und J. Teich: *A system-level synthesis approach from formal application models to generic bus-based MPSoCs*. In: *2010 International Conference on Embedded Computer Systems (SAMOS)*, S. 118 –125, Juli 2010.

[5] Grüttner, K., K. Hylla, S. Rosinger und W. Nebel: *Towards an ESL framework for timing and power aware rapid prototyping of HW/SW systems*. In: *Forum on Specification and Design Languages*, FDL'2010, S. 1–6, Sep. 2010.

[6] IEEE Computer Society: *IEEE Standard SystemC® Language Reference Manual*. IEEE Std 1666™-2005, März 2006, ISBN 0-7381-4870-9. `http://standards.ieee.org/getieee/1666/`.

[7] Lebreton, H. und P. Vivet: *Power Modeling in SystemC at Transaction Level, Application to a DVFS Architecture*. In: *IEEE Annual Symposium on VLSI*, ISVLSI'08, S. 463–466. IEEE Computer Society, Apr. 2008.

[8] Open SystemC Initiative: *OSCI TLM-2.0 Language Reference Manual*. Version JA32, Juli 2009. `http://systemc.org/`.

[9] Streubühr, M., R. Rosales, R. Hasholzner, C. Haubelt und J. Teich: *ESL Power and Performance Estimation for Heterogeneous MPSoCs Using SystemC*. In: *Forum on Specification and Design Languages*, S. 202–209, Oldenburg, Germany, Sep. 2011.

[10] Walravens, C., Y. Vanderperren und W. Dehaene: *ActivaSC: A highly efficient and non-intrusive extension for activity-based analysis of SystemC models*. In: *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, S. 172–177, Juli 2009.