Gefördert durch:

Bundesministerium
für Wirtschaft
und Klimaschutz

aufgrund eines Beschlusses
des Deutschen Bundestages

KI Embedded

KI-Grundlagenentwicklung mit Leitanwendungen
Virtuelle Sensorik und Brennstoffzellenregelung

Abschlussbericht

Zuwendungsempfänger	PLS Programmierbare Logik & Systeme GmbH
Förderkennzeichen	19I21043F
Projektlaufzeit	01.09.2021 – 31.12.2024

Das diesem Bericht zugrunde liegende Vorhaben wurde mit Mitteln des Bundesministeriums für Wirtschaft und Klimaschutz gefördert.

Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

Autoren

Heiko Rießland (PLS Programmierbare Logik & Systeme GmbH)
Andreas Gajda (PLS Programmierbare Logik & Systeme GmbH)
Fridolin Fräulin (PLS Programmierbare Logik & Systeme GmbH)

Im Folgenden wird die Abkürzung PLS GmbH verwendet.

Inhalt

I. Kurze Darstellung 3
1 Aufgabenstellung 3
2 Voraussetzungen, unter denen das Vorhaben durchgeführt wurde 4
3 Planung und Ablauf des Vorhabens 4
3.1 Projektaufbau 5
3.2 Veränderungen gegenüber der ursprünglichen Planung 6
4 Wissenschaftlicher und technischer Stand, an den angeknüpft wurde 6
Debug- Trace- und Test-Werkzeuge für KI-Hardware-Beschleuniger DFA 6
5 Zusammenarbeit mit anderen Stellen 7
II. Eingehende Darstellung 7
1 Verwendung der Zuwendung und des erzielten Ergebnisses 7
1.1 AP2: Hardwarebeschleunigung 8
1.2 AP 3: Methoden und Werkzeuge zur Anwendungsvalidierung 19
1.3 AP 4: Use-Case ICE – Smart Monitoring mit KI-basierten, virtuellen Sensoren AP 5 Use-Case Fuel Cell – Modellbasierte Regelung des Brennstoffzellenstapels 33
1.4 AP 6: Projektmanagement 33
2 Wichtigste Positionen des zahlenmäßigen Nachweises 33
3 Notwendigkeit und Angemessenheit der geleisteten Arbeit 33
4 Voraussichtlicher Nutzen, insbesondere der Verwertbarkeit des Ergebnisses im Sinne des fortgeschriebenen Verwertungsplans 34
5 Während der Durchführung des Vorhabens dem ZE bekannt gewordener Fortschritt auf dem Gebiet des Vorhabens bei anderen Stellen 34
6 Erfolgte und geplante Veröffentlichungen des Ergebnisses 35
7 Literaturverzeichnis 35
8 Abbildungsverzeichnis 36

I. Kurze Darstellung

Das Projekt KI Embedded verfolgte das Ziel, neuartige KI-Verfahren zur Modellierung und Regelung komplexer dynamischer Systeme für den Einsatz auf eingebetteten Systemen zu entwickeln. Im Fokus stand dabei die Berücksichtigung der spezifischen Anforderungen ressourcenbeschränkter eingebetteter Hardwareumgebungen (englisch Embedded Systems). Um die Entwicklung und Integration dieser Verfahren systematisch zu unterstützen, wurde durch die Projektpartner itemis, OFFIS, Bosch und PLS GmbH eine durchgängige Werkzeugkette bestehend aus Editor, Compiler, Simulationsumgebung und Debugger konzipiert und prototypisch implementiert. Diese Infrastruktur ermöglicht eine effiziente Entwicklung, Validierung und Übertragung KI-basierter Regelungen auf eingebettete Plattformen. Die Machbarkeit wurde exemplarisch anhand eines Hardwarebeschleunigers demonstriert, auf dem ein Regelungssystem für einen Brennstoffzellenstapel bzw. virtuelle Sensorik experimentell realisiert wurden. Das Projekt leistet damit einen wichtigen Beitrag zur breiten industriellen Nutzbarmachung KI-gestützter Regelungsverfahren im Bereich eingebetteter Systeme.

1 Aufgabenstellung

Der vom Projektpartner Bosch bereits vor Projektbeginn entwickelte Hardwarebeschleuniger Data Flow Architecture (DFA) stellt einen innovativen Ansatz im Bereich spezialisierter Rechenbeschleuniger dar. Im Unterschied zu herkömmlichen, sequenziell arbeitenden Prozessorarchitekturen basiert die DFA auf einem massiv parallelen Berechnungsmodell, das es erlaubt, komplexe Algorithmen in Form von Datenflussgraphen effizient und gleichzeitig abzuarbeiten. Diese Architektur verspricht erhebliche Vorteile hinsichtlich Rechengeschwindigkeit, Energieeffizienz und notwendiger Chipfläche, insbesondere für KI- und Regelungsanwendungen auf eingebetteten Systemen. Sie erfüllt darüber hinaus die strengen Sicherheitsanforderungen (Safety) der Automobilindustrie.

Um die Integration und praktische Nutzbarmachung der DFA in reale Entwicklungsprozesse zu ermöglichen, hat die PLS GmbH im Rahmen des Projekts wesentliche Beiträge zu einer Werkzeugkette für die DFA-Konfiguration geleistet. Im Mittelpunkt stand dabei die Erforschung und prototypische Umsetzung neuartiger Konzepte für das Debugging, die Laufzeitbeobachtung und den Test der DFA-basierten Systeme. Angestrebt wurde ein leistungsfähiger Debugger-Prototyp, der eine systematische Analyse und Kontrolle der DFA sowohl auf niedriger hardwarenaher Ebene (z. B. Register- und Signale) als auch auf höheren Abstraktionsebenen (z. B. grafische Visualisierung der Konfiguration und Datenflüsse) ermöglicht. Dadurch wird nicht nur die Nachvollziehbarkeit der internen Abläufe verbessert, sondern auch die Fehlersuche und Optimierung im Entwicklungsprozess erheblich erleichtert.

Zur Unterstützung der Interoperabilität innerhalb der gesamten Werkzeugkette wurde darüber hinaus ein offenes Datenformat konzipiert und implementiert. Dieses Format dient der konsistenten und flexiblen Weitergabe von Laufzeit- und Debug-Informationen an andere Werkzeuge der Projektpartner sowie an externe Systeme. So wird der reibungslose Informationsaustausch innerhalb des Entwicklungsprozesses sichergestellt und die langfristige Anschlussfähigkeit an zukünftige Werkzeuge und gewährleistet.

2 Voraussetzungen, unter denen das Vorhaben durchgeführt wurde

Die Automobilindustrie steht vor einem tiefgreifenden Wandel. Der Klimawandel erfordert den Umstieg auf neue, nachhaltige Energieträger – etwa die Brennstoffzelle als mobiler Energieversorger. Gleichzeitig führt der Markteintritt neuer Hersteller zu einem verschärften Wettbewerb und steigenden Kostendruck. Dies verstärkt das Bestreben, bestehende Technologien zu optimieren und zu kostensenkenden Maßnahmen zu greifen, etwa durch die Reduktion der eingesetzten Sensorik.

Langfristig ist jedoch ein nahezu vollständiger Technologiewandel erforderlich. In vielen Bereichen gilt Wasserstoff als zentrale Zukunftstechnologie – insbesondere im Nutzfahrzeugsektor, wo Brennstoffzellensysteme bereits heute wirtschaftlich konkurrenzfähig sind. Die effiziente Modellierung und Regelung solcher Systeme stellen jedoch eine besondere Herausforderung dar: Die zugrunde liegenden physikalischen Prozesse sind komplex, wirken über längere Zeiträume hinweg zusammen und sind häufig schwer direkt zu messen oder zu instrumentieren.

Diese hohe Systemkomplexität bei gleichzeitig hohem Genauigkeitsanspruch macht den Einsatz von KI-Verfahren besonders vielversprechend. Solche Methoden haben sich bereits bei der Modellierung und Steuerung konventioneller Antriebssysteme erfolgreich bewährt. Um jedoch den Übergang von der Vorentwicklung zur seriennahen Umsetzung zu beschleunigen, genügt die bloße Entwicklung innovativer Algorithmen nicht. Es müssen auch die entsprechenden technischen Voraussetzungen geschaffen werden, um diese Verfahren unter industriellen Rahmenbedingungen nutzbar zu machen.

Dazu sind leistungsfähige Entwicklungswerkzeuge erforderlich, die eine konsistente, nachvollziehbare und zeiteffiziente Umsetzung unterstützen – von der Modellierung über die Simulation bis hin zur Umsetzung auf Zielhardware. Nur mit einer solchen Werkzeugbasis kann der Transfer innovativer KI-Lösungen in die industrielle Anwendung gelingen.

3 Planung und Ablauf des Vorhabens

Die Planung im Projekt KI Embedded orientierte sich an den Voraussetzungen zu Projektbeginn. Der (KI-)Hardwarebeschleuniger DFA war zu Planungsbeginn vom Projektpartner Bosch entwickelt und bereits in einem ersten Vorserien-Chip der Firma STMicroelectronics integriert. Im Verlaufe des Projektes erfolgte eine Weiterentwicklung der DFA Hardware und eine Integration in Vorserien-Chips weiterer Halbleiterhersteller (NXP, Renesas). Für die anderen Projektpartner zeichnete sich ein größerer Bedarf an Modellierungs- und Regelungsverfahren für Brennstoffzellen-Fahrzeuge ab, so dass dieser Anwendungsfall als Beispielsystem ausgewählt wurde.

Das Projekt war auf eine Laufzeit von 34 Monaten konzipiert. Es startete am 1. September 2021. Durch Personalengpässe und Verzögerungen in der Lieferung des Brennstoffzellenstapels wurde eine kostenneutrale Verlängerung des Projekts notwendig. Nach Verlängerung endete es fristgemäß zum 31.12.2024.

3.1 Projektaufbau

Bei der Projektplanung wurden sechs Arbeitspakete (AP) definiert welche die logische Abfolge der Entwicklung definieren

- AP1 Modellierung und Werkzeuge
- AP2 Hardwarebeschleunigung
- AP3 Methoden und Werkzeuge zur Anwendungsvalidierung
- AP4 Use-Case ICE – Smart Monitoring mit KI-basierten, virtuellen Sensoren
- AP5 Use-Case Fuel Cell – KI-basierte Regelung
- AP6 Projektmanagement

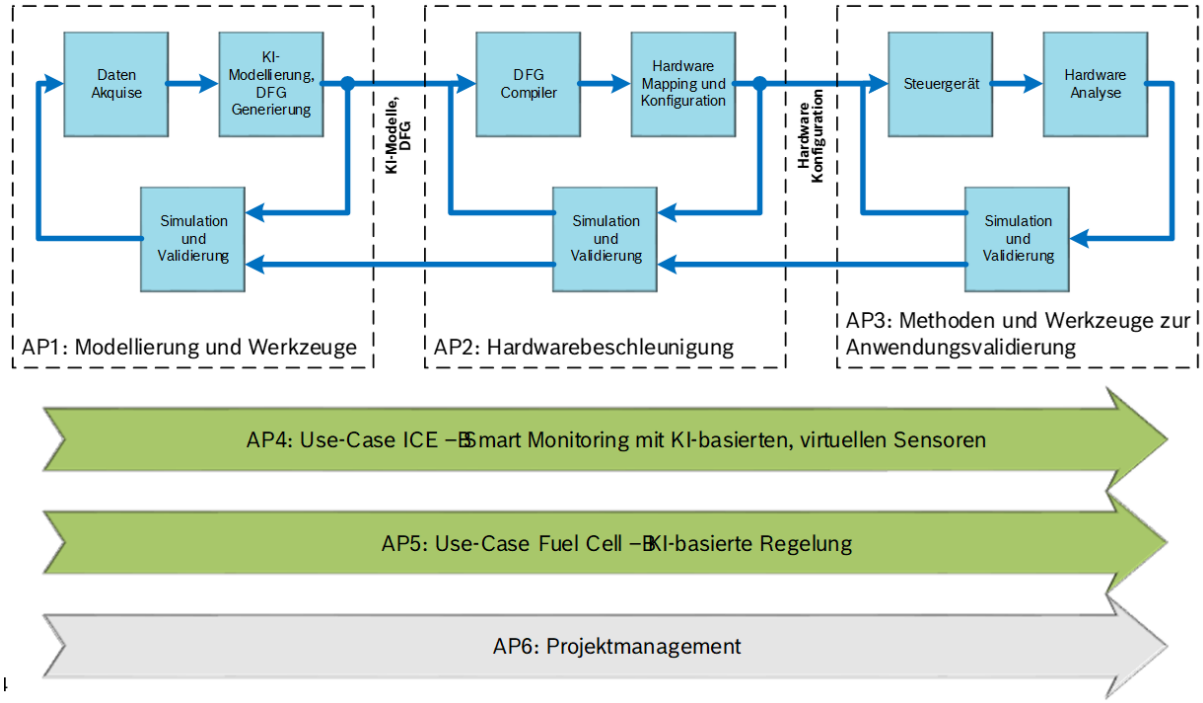


Abbildung 1: Gliederung des Projekts KI Embedded anhand von Arbeitspaketen

Die Beiträge der PLS GmbH lagen im Wesentlichen in den Arbeitspaketen AP2 und AP3 sowie den projektbegleitenden Anwendungspaketen AP4 bis AP6. Das Arbeitspaket 3 "Methoden und Werkzeuge zur Anwendungsvalidierung" wurde von der PLS GmbH geleitet. Da sich die Beiträge der Partner über mehrere Arbeitspakete erstrecken ist eine Darstellung der inhaltlichen Zuordnung der Beiträge übersichtlicher, so wie sie in Abbildung 2 gezeigt wird.

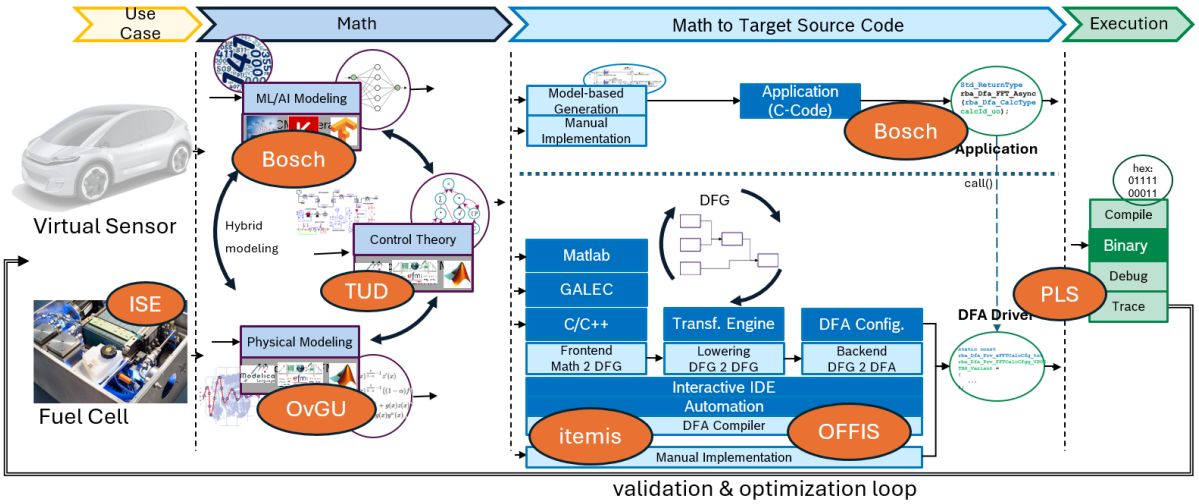


Abbildung 2: Bestandteile des Projekts KI Embedded und Zuordnung der Partner zu den Hauptinhalten ihrer Beiträge

3.2 Veränderungen gegenüber der ursprünglichen Planung

Während der Projektlaufzeit ergaben sich bei den anderen Projektpartnern durch Lieferschwierigkeiten, Personalmangel und Entwicklungen im Bereich der KI-Verfahren Umstände, die zu leichten Abweichungen vom ursprünglichen Plan geführt haben. Diesen Abweichungen wurde in einer aktualisierten Vorhabenbeschreibung Rechnung getragen, welche zusammen mit einem kostenneutralen Verlängerungsantrag umgesetzt wurde. Die PLS GmbH hat sich dem kostenneutralen Verlängerungsantrag angeschlossen, da die Bearbeitung der PLS Unterarbeitspakete in den Arbeitspaketen AP4 und AP5 ohne die Ergebnisse der entsprechenden Projektpartner nur bedingt und ggf. nur mit erhöhten Kosten möglich gewesen wäre.

Über die ursprüngliche Planung hinaus wurde vom Projektpartner Bosch ein "System-C" – Modell der DFA sowie eine prototypische Treibersoftware zur Verfügung gestellt. Da die aktuell verfügbare Hardware noch keine Tracing-Mechanismen enthält, wurde vereinbart, dass Trace-Daten auf Basis des "System-C" - Modells gewonnen werden. Entsprechende Mechanismen sind dort verfügbar.

4 Wissenschaftlicher und technischer Stand, an den angeknüpft wurde

Debug- Trace- und Test-Werkzeuge für KI-Hardware-Beschleuniger DFA

Der vom Projektpartner Bosch entwickelte KI-Hardware Beschleuniger DFA ist ein Datenflussprozessor. Diese Technologie ist aus der Forschung bekannt, schafft jedoch erst durch spezielle Anpassungen den Sprung in das industrielle Umfeld. DFA ist insbesondere auf tief eingebettete Systeme zugeschnitten und zeichnet sich aufgrund des geringen Flächenbedarfs durch minimale Kosten, hohe Energiesparsamkeit und die Unterstützung von Sicherheitsanforderungen (Safety) aus. Da die DataFlow-Architecture noch in der Entwicklung begriffen ist, gab es nur sehr wenig Werkzeugunterstützung für die Entwicklung.

Aktuelle Werkzeuge zur Visualisierung, Testung und Fehlersuche (Debugger) in herkömmlichen Rechnerarchitekturen gehen von einer überwiegend sequentiellen Abarbeitung von fest definierten Maschinenbefehlen aus, die auch mehrstufig aus einer Modell- oder Programmiersprache generiert werden. Zusätzlich zu den Maschinenbefehlen werden symbolische Informationen generiert, welche die Debugger für die Rücktransformation auf die Programmiersprache verwenden. Zusätzlich extrahieren Debugger mittels Trace-Hardware Informationen über den tatsächlich sequentiell erfolgten Kontroll- und Datenfluss. Die Steuerung, Visualisierung, Fehlersuche und Testung neuartiger KI-Hardware (Datenfluss-Architektur) und den darauf ausgeführten parallelen Algorithmen ist mit den auf sequentieller Abarbeitung basierenden Ansätzen aber nicht umsetzbar.

Für einen breiten industriellen Einsatz und somit den Erfolg des Hardwarebeschleunigers DFA sind aber entsprechende Werkzeuge von zentraler Bedeutung. Die Testung und Validierung der zu entwickelnden Algorithmen für die neuartige KI-Hardware DFA erfordern eine Visualisierung und Steuerung für parallele, dynamisch konfigurierbare Hardware mittels Prä- und Post-Analyse des Systems. Über die reine Darstellung der statischen Hardware-Zustände hinaus, ist es erforderlich, die Konfiguration mit dem ursprünglichen dynamischen Modell abgleichen zu können und auch die Datenflüsse angemessen zu visualisieren. Die dafür zu entwickelnden Debug-Werkzeuge müssen dabei die massive Parallelität des Systems berücksichtigen und neue Visualisierungsmöglichkeiten für die Zustandsbeobachtung und -veränderung entwickeln.

5 Zusammenarbeit mit anderen Stellen

Zur Koordinierung der Entwicklungen für die Werkzeugkette im Rahmen der Arbeitspakete 2 und 3 gab es eine wöchentliche Telefonkonferenz der Projektpartner Robert Bosch AG, Itemis AG, OFFIS e.V. und PLS GmbH. Bei Bedarf gab es weitere zumeist bilaterale Zusammenarbeit in ad-hoc Online-Meetings. Die Gesamtkoordination des Projektes wurde durch monatliche Telefonkonferenzen der Projektleiter und durch halbjährige Präsenztreffen der am Projekt beteiligten Personen an verschiedenen Projektstandorten sichergestellt.

Die Koordination der durch PLS eingebundenen FE-Fremdleistungen (GWT-TUD GmbH) erfolgte durch monatliche Telefonkonferenzen im Vertragszeitraum.

II. Eingehende Darstellung

1 Verwendung der Zuwendung und des erzielten Ergebnisses

Die PLS GmbH hat im Rahmen des Projektes Methoden zur Fehlersuche, Testung und Beobachtung des Datenflussprozessors DFA des Projektpartners Bosch untersucht und Lösungskonzepte erarbeitet.

Wesentliche Schwerpunkte waren dabei Untersuchungen zu notwendigen Funktionen innerhalb einer integrierten SoC-Debug-Umgebung zur Fehlersuche und Algorithmus-Testung sowie zur Unterstützung der Ablaufsteuerung und Visualisierung von internen Werten und Zuständen der DFA, die Konzeption einer Schnittstelle zur Übernahme von Konfigurationsdaten für die DFA sowie die Entwicklung von Filter-Algorithmen zur Separierung von zur Laufzeit des Datenflussprozessors gewonnenen Informationen aus aufgezeichneten Trace-Daten eines SoC-Gesamtsystems (alle AP2).

Der Anteil der PLS GmbH im AP3 umfasste die Entwicklung von Algorithmen zur analytischen und visuellen Aufbereitung der Laufzeitinformationen zur Fehlersuche und Testung des abgearbeiteten Modells sowie zu dessen Beobachtung sowie Untersuchungen zur Steuerung und Überwachung des gesamten SoC für Fehlersuche und Test der Interaktion des Datenflussprozessors mit den umgebenden Chip-Komponenten.

Das im Projekt erzielte Ergebnis ermöglicht die Schaffung neuer Werkzeuge zur Fehlersuche, Testung und Validierung von Software, die KI-Hardwarebeschleuniger in zukünftigen System-on-Chip (SoC‘s) nutzt. Die zu entwickelnden Methoden und Werkzeuge erhöhen Anwendbarkeit und Attraktivität solcher KI-Beschleuniger-Hardware durch effizientere Algorithmen-Entwicklung und Testbarkeit und erschließen das Potential neuen Anwendergruppen basierend auf den konzeptionell neuen Produkteigenschaften.

1.1 AP2: Hardwarebeschleunigung

Grundlegende Funktionen zur Konfiguration, zum Debuggen und zur Laufzeitbeobachtung des KI-Hardwarebeschleunigers

Bereitstellung von Basis-Debug-Funktionen für KI-Hardwarebeschleuniger

In diesem Teilarbeitspaket wurde der grundlegende Debug-Zugang (Konfiguration, Beobachtbarkeit und Steuerung) zum ersten verfügbaren System-on-Chip (SoC) im Allgemeinen und dem eingebetteten KI-Hardwarebeschleuniger im Besonderen geschaffen.

Ziel der Debug-Funktionen ist es, das Verhalten der Data Flow Architecture (DFA) in der frühen Phase der Algorithmen-Entwicklung zu verstehen. Dies wird durch Einzelschritt-Debugging und das Beobachten der DFA im angehaltenen Zustand realisiert. Aus dem Quelltext erzeugter, sequentiell ausgeführter Maschinencode wie bei herkömmlichen controllerbasierten Hardwarebeschleunigern existiert in diesem Kontext nicht, da die DFA eine andere Struktur aufweist. Anstelle des Quelltexts tritt eine textuelle Beschreibung eines Algorithmus in einer speziellen Programmiersprache wie z. B. Guarded Algorithmic Language for Embedded Control (GALEC). Dieser Text wird in mehreren Schritten übersetzt und optimiert, um Adress-/Wertepaare zu erzeugen, die den Registerinhalt der Basisblöcke der DFA darstellen. Dies wird als DFA-Konfiguration bezeichnet und ersetzt im Vergleich zu traditionellen Computerarchitekturen den Binärcode. Nach einer Ressourcenprüfung kann diese Konfiguration auf eine konkrete Hardware (HW) abgebildet werden. Als Zwischenformat zwischen Algorithmus und DFA-Konfiguration wurde von den Projektpartnern itemis AG und OFFIS ein Data-Flow-Graph (DFG)-Modell entwickelt, das von anderen Teilen der Werkzeugkette verwendet wird.

Der Debug-Zugang umfasste im ersten Schritt das Lesen und Schreiben von Registern und Speicher, sowie die Zustandskontrolle des SoC und des Beschleunigers. Die integrierte DFA-Einheit ermöglicht der Zielsoftware ausschließlich schreibenden Zugriff auf ihre Basisblöcke über das Advanced eXtensible Interface (AXI). Für Debugging-Zwecke ist ein Backdoor-Protokoll über den Advanced Peripheral Bus (APB) implementiert, welches Lese- und Schreibzugriffe erlaubt. Entsprechende Algorithmen wurden entwickelt, die für den Nutzer transparent, die Bedienung der in Hardware realisierten Schutzfunktionen zur Absicherung eines unbefugten Zugriffes realisieren.

Für den Zugriff auf die Register und Speicher des Beschleunigers wurde eine Registerdatenbank erstellt und eine Software-API entwickelt deren Funktionalität und praktische Verwendbarkeit an einer prototypischen Implementierung durch Regressionstests abgesichert wird. Komplementär wurde für das Nutzer-Interface eine GUI-Komponente "DFA-View" entwickelt, die eine manuelle Überprüfung der Register und Speicherinhalte ermöglicht. Da die DFA über keine zentrale Start-/Stopp-Möglichkeit verfügt mussten andere Lösungen gefunden werden. Zwei Mechanismen ermöglichen eine Art Ausführungs-Steuerung. Die Load/Store-Basisblöcke benötigen ein Trigger-Signal, um aktiv zu werden. Dies erlaubt einen definierten Start eines vorkonfigurierten Algorithmus. Ein asynchrones Anhalten der DFA ist durch das Setzen in den Debug-Modus möglich. Darüber hinaus ermöglicht ein Schrittzählermechanismus die Ausführung von 1 bis n Taktzyklen.

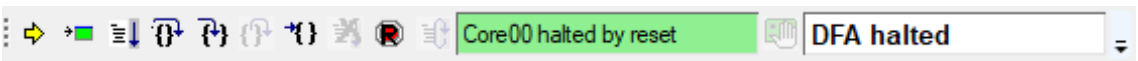


Abbildung 1 Prototypische Toolbar für den Core00 eines die DFA integrierenden SoC und die DFA. Die Elemente zur Ausführungssteuerung auf der linken Seite sind jeweils dem aktiven Core zugeordnet, das im Moment gerade die DFA ist.

Diese Basis-Debug-Funktionen wurden sowohl über ein Nutzerinterface, als auch programmatisch über eine Software-API zur Verfügung gestellt. Diese ermöglicht es, den Debugger durch externe Programme – z. B. Python-Skripts – zu steuern. Alle (API- und Nutzer-) Schnittstellen wurden in fachlicher Abstimmung mit den Projektpartnern konzipiert und entwickelt.

Definition und Implementierung einer Schnittstelle zur Übernahme von Konfigurationsdaten für den KI-Hardwarebeschleuniger

Für die Konfiguration des KI-Hardwarebeschleunigers wurde eine Schnittstelle in Form eines textbasierten und maschinenlesbaren Datenaustauschformates im Austausch mit den Projektpartnern entwickelt. Dieses kann sowohl vom Debugger verwendet werden und ist auch zur Generierung einer DFA-Konfigurationsbibliothek durch die vorgelagerten Werkzeuge nutzbar. Für den Prozess der Quellcodeerzeugung werden die Registerkonfiguration und die Operanden benötigt, um statische und dynamische Konfigurationsvektoren zu generieren. Die Schnittstelle ist in einer XSD-XML-Schemadatei beschrieben.

```
<?xml version="1.0" encoding="utf-8"?>
<DFAConfiguration>
  <AlgoName>FC Sigmoid Configuration</AlgoName>
  <SupportedDfaArchitecture>bosch.com:dfa_lib:dfa_core:1.0</SupportedDfaArchitecture>
  <BaseBlock id="bb_load_1_4_2x32_s2s1.instance0.split0.set0.input0">
    <Name>bb_load_1_4_2x32_s2s1</Name>
    <BaseBlockID>0</BaseBlockID>
    <TargetPortID>0</TargetPortID>
    <SplitID>0</SplitID>
    <SetID>0</SetID>
    <Register>
      <Name>control_split_combine</Name>
      <Value>0</Value>
      <Field>
        <Name>sel_split_combine</Name>
        <Value>sel_split_combine_split</Value>
      </Field>
    </Register>
    ...
  </BaseBlock>
</DFAConfiguration>
```

Abbildung 2 Auszug aus einer XML-basierten DFA-Konfiguration

Die Registerkonfiguration selbst ist ein XML-basiertes Format mit UTF-8 Codierung. Sie erlaubt das Einbinden von Drittanbieterinformationen durch die Einbindung einer Schemadatei „ModelExtension_Schema.xsd“ in das DFA-Konfigurationsschema. Dies kann genutzt werden, um ModelExtension-Elemente hinzuzufügen, die wiederum die für den Debugger erforderlichen Back-Tracing-Informationen enthalten können. Es wurden Import- und Export-Schnittstellen entwickelt und das Datenformat so dokumentiert, dass der Test und die iterative Weiterentwicklung des Formats gemeinsam mit den Projektpartnern möglich wurden. Die Funktionalität und praktische Verwendbarkeit der prototypischen Implementierung wurden durch entsprechende Regressionstests sichergestellt.

Das Trace-Konzept für die DFA-Hardware basiert auf der Nutzung des AMBA Trace Bus (ATB), der die Übertragung beliebiger Daten sowie die Bereitstellung synchronisierter Zeitstempel ermöglicht. Der Fokus des Konzepts liegt auf der Wiederverwendung von Debug-Komponenten zur Erzeugung und benutzerdefinierten Formatierung der Trace-Daten.

Zur Unterstützung des Debuggings soll die DFA um ein Breakpoint-Konzept erweitert werden. Diese Breakpoints geben Signale aus, wenn bestimmte, auf den Zuständen von Basisblöcken beruhende Bedingungen erfüllt sind. Wird eine solche Bedingung erfüllt, sendet der Breakpoint ein Signal, das den DFA in den Debug-Modus versetzt, d. h., die Ausführung aller Blöcke wird angehalten.

Dasselbe Signal dient auch dazu, Trace-Daten für jeden Basisblock auszugeben. Die Daten werden als 64-Bit-Werte fester Länge geschrieben, wobei der Benutzer konfigurieren kann, an welcher Bitposition sich die eigentlichen Informationen innerhalb des 64-Bit-Werts befinden. Die Daten verschiedener Basisblöcke können in einem einzigen 64-Bit-Wert kombiniert werden, indem unterschiedliche Bitpositionen verwendet werden.

Es ist zu erwarten, dass das erzeugte Trace-Datenvolumen die Übertragungsrate des ATB deutlich übersteigt. In diesem Fall kann der Benutzer konfigurieren, ob der DFA gestoppt werden soll, bis alle Daten geschrieben sind, oder ob Daten verworfen werden dürfen.

Potenzial für weiterführende Analysen

Angesichts der Informationsmenge pro Basisblock und der gleichzeitigen Ausführung mehrerer Basisblöcke ist davon auszugehen, dass die erzeugten Trace-Daten die Bandbreite des ATB deutlich übersteigen. Daher bleiben im Wesentlichen zwei Optionen:

a. Die Beobachtung wird auf wenige Basisblöcke beschränkt, um ein Anhalten des DFA zu vermeiden, oder
b. es werden alle relevanten Daten beobachtet, wobei ein Anhalten des DFA in Kauf genommen wird. Beide Ansätze haben ihre Berechtigung, abhängig vom Analyseziel.

Zeit- und Leistungsanalyse

Analysen mit Fokus auf Timing und Leistung können sich auf wenige, für die Analyse wesentliche Basisblöcke beschränken. Beispielsweise kann für die Analyse eines bestimmten Basisblocks das zugehörige Register sowie seine Vorgängerblöcke beobachtet werden. Alternativ kann für eine umfassende Algorithmusanalyse das Verhalten der Lade- und Speicherblöcke untersucht werden.

Weitere Performance-Analysen können sich auf den Durchsatz des DFA konzentrieren. Dies wäre möglich, indem ausschließlich die durch den DFC-Block ausgelösten Interrupts beobachtet werden, die jeweils ein einzelnes Datenpaket erzeugen. Die Daten könnten in einer Zeitachse dargestellt werden, synchronisiert mit anderen Tracedaten (z. B. Program-Trace), um Ursachen für Leerlaufzeiten des DFA zu identifizieren.

Funktionale Analyse

Der Interrupt, der am Ende der Ausführung ausgelöst wird, kann auch zur Überprüfung der korrekten Funktionsweise des DFA genutzt werden. Ist dieses Ereignis in den Trace-Daten nicht enthalten, kann davon ausgegangen werden, dass der DFA oder der konfigurierte Algorithmus nicht korrekt arbeitet. Diese Analyse wird zuverlässiger, wenn zusätzlich ein dediziertes Startereignis beobachtet wird, z. B. das Schreiben in das „start_control“-Register der Ladeblöcke.

Datenfluss- und Abhängigkeitsanalyse

Die Analyse von Daten, deren Veränderungen und Abhängigkeiten erfordert eine vollständige Sicht auf sämtliche Zwischenresultate jedes Basisblocks. Dies erzeugt jedoch ein sehr großes Datenvolumen, was wiederum dazu führt, dass der DFA angehalten wird. Das bedeutet, das Laufzeitverhalten wird beeinflusst.

Da Timing-Analysen und Datenflussanalysen in der Regel nicht gleichzeitig durchgeführt werden, ist diese Einschränkung weniger kritisch. Es muss jedoch sichergestellt werden, dass die durch das Anhalten verursachte Verzögerung so gering bleibt, dass die Ausführung für den Benutzer in akzeptabler Zeit abgeschlossen werden kann.

System-C- Modell

Über die ursprüngliche Planung hinaus wurde vom Projektpartner Bosch ein "System-C" - Kommandozeilen-Modell der DFA sowie eine prototypische Treibersoftware zur Verfügung gestellt. Da die aktuell verfügbare DFA-Hardware noch keine Tracing-Mechanismen enthält, wurde vereinbart, dass Trace-Daten auf Basis des "System-C" - Modells gewonnen werden. Entsprechende Mechanismen sind dort verfügbar und können zur vorgezogenen Bewertung von DFA-Algorithmen unabhängig von verfügbarer Hardwareimplementierung verwendet werden. Das System wurde hinsichtlich der bereitgestellten Informationen und möglicher Folgeanalysen untersucht. Es bietet zwei Datenquellen zur Analyse der Qualität und Quantität potenzieller Trace-Daten: ein Value Change Dump (VCD) der Signale sowie eine Log-Funktion für die instrumentierten Basisblöcke.

Value Change Dump

Der Value Change Dump liefert Ereignisse von Interrupt- und internen Signalleitungen in numerischem Format, das für weiterführende Analysen interpretiert werden kann. Die Ausgabe erfolgt im ASCII-Format und wird in einer Datei gespeichert, deren Name beim Start der Simulation angegeben werden muss.

Das Format enthält die Kennung der Signalleitung, Zeitinformationen zu Ereignissen sowie den neuen Wert zum Zeitpunkt des Ereignisses. Die Zeitinformationen basieren auf einer internen Zeitreferenz, die zur Simulation des Taktsystems des DFA-Cores verwendet wird. Dies ist für die nachfolgende Korrelation der Ereignisse mit den Log-Ausgaben wesentlich.

Die Werte hängen von der beobachteten Signalleitung ab und bestehen meist aus alternierenden Null-/Eins-Werten. Da in der Regel nur ein einziger Durchlauf des zu testenden Algorithmus beobachtet wird, werden meist nur wenige Ereignisse aufgezeichnet – hauptsächlich die vom DFA-Core ausgelösten Interrupt-Signale, die das Ende einer Analyse anzeigen.

Zeit- und Performanceanalyse

Die Analyse des Zeitabstands zwischen Ereignissen erlaubt die Bestimmung der logischen Laufzeit einer Analyse, sofern ein explizites Start- und Endsignal existiert.

Der VCD enthält in der Regel das Interrupt-Signal, das von der Analyse genutzt wird, um dem Interrupt-Handler das Analyseende zu signalisieren. Dies hängt jedoch von der Konfiguration ab – ein Analysewerkzeug muss daher wissen, ob dieses Ereignis erwartet werden kann.

Der Beginn einer Analyse ist in den Ergebnissen nicht explizit dargestellt. Möglicherweise kann das Reset-Ereignis als Startpunkt verwendet werden, da die DFA-Treiberbibliothek einen Reset ausführt, sobald eine neue Konfiguration in die DFA-Komponente geschrieben wird. Da jedoch kein Reset erfolgt, wenn derselbe Algorithmus mit neuen Daten verwendet wird, ist das Reset-Ereignis kein verlässlicher Indikator für den Analysebeginn.

Es ist nicht möglich, aus der Datei die Laufzeit bzw. Performance einzelner Basisblöcke zu ermitteln.

Funktionale Analyse

Wenn im Vorfeld, z. B. durch Analyse der DFA-Konfiguration, bekannt ist, ob ein Signal am Ende der Analyse zu erwarten ist, kann ein Analysetool durch gezielte Suche dieses Ereignisses auf die korrekte Funktion des DFA und/oder des Algorithmus schließen.

Da die Daten derzeit nur in eine Datei geschrieben werden, ist nur eine nachträgliche (post-mortem) Analyse möglich. Durch Bereitstellung einer alternativen Analyseeinheit zur Laufzeit wäre es denkbar, die Ausgaben parallel zur Algorithmus-Ausführung zu analysieren.

Software-Log

Die Log-Ausgabe besteht aus frei formatiertem Text an die Standardausgabe. Diese kann umgeleitet werden, z. B. in eine Datei zur nachträglichen Aufbereitung oder direkt an ein Tool zur gleichzeitigen und/oder nachgelagerten Analyse. Die Basisblöcke und Speicher-Interfaces sind instrumentiert und verwenden ein festes Format zur Protokollierung von Ein- und Ausgabedaten. Da die Ausgabe jedoch an die Standardausgabe erfolgt, können frei formatierte Nachrichten anderer Komponenten (z. B. Ergebnisdarstellung) eingestreut sein.

```
[INFO·]:·442500·ps·000130:·HW32_W·-·AXI-W·{.address_u32=0x003030c0,.data=0x00000101},¶

[INFO·]:·00445·ns·000133:·dfa.i_architecture_004.i_BB_Ex_AddSub__2_2__s1s3_1·-·Access·on·
targetSocketId:·0,·currentSplitId:·0,·currentSetId:·0,·currentRegisterID:·4,·Value·(hex):·bfc8dfc5,·
Value·(i):·3217612741,·Value·(f):·-1.569329e+00·-·success¶

[INFO·]:·00445·ns·000133:·dfa.i_architecture_004.i_BB_Ex_AddSub__2_2__s1s3_0·-·Access·on·
targetSocketId:·0,·currentSplitId:·0,·currentSetId:·0,·currentRegisterID:·4,·Value·(hex):·bfc8dfc5,·
Value·(i):·3217612741,·Value·(f):·-1.569329e+00·-·success¶

[INFO·]:·447500·ps·000136:·dfa.i_architecture_004.i_BB_Ex_Mac__3_2__s2s6_0·-·Access·on·
targetSocketId:·0,·currentSplitId:·0,·currentSetId:·3,·currentRegisterID:·1,·Value·(hex):·1,·Value·
(i):·1,·Value·(f):·1.401298e-45·-·success¶

[INFO·]:·447500·ps·000136:·dfa.i_architecture_004.i_BB_Load__1_4_2x32__s2s1_0·-·Read_done:·splitId·
=·0,·setId·=·0,·with·current·offset:·1,·addr:·4·(0x4),·data:·i:·3209679282·(0xbf4fd1b2),·data:·f:·-
8.117934e-01¶
```

Abbildung 3 Auszug aus einer Log-Datei des System-C - Modells

Das System-C - Modell verwendet nur begrenzte Parallelität, sodass Logmeldungen stets vollständig ausgegeben werden, bevor die nächste Nachricht erscheint. Die interne Speicherinterface-Implementierung verwendet ein festes Format für Ausgaben. Es werden Lese- und Schreibzugriffe an den DFA protokolliert, jedoch keine Lese-/Schreibzugriffe vom DFA in den Hauptspeicher. Es wäre möglich, dem System-C – Modell Instrumentierungscode für Zugriffe auf den Hauptspeicher hinzuzufügen, um so auch Eingabedaten und Ausgabedaten beobachtbar zu machen. Die Analyse führte zu folgenden Erkenntnissen. Das Log-Format ist formal genug, um mit regulären Ausdrücken geparst zu werden. Das Log-Format enthält ausreichend Informationen, um den Datenflussgraphen zu konstruieren. Der Datenfluss wird im gesamten Graphen erfasst. Der extrahierte Datenfluss lässt sich auf den Hardware-Beschleuniger abbilden. Auf Grundlage dieser Analyse wurde ein Trace-Aufzeichnungsschema entwickelt, das die Struktur und den Datenfluss des Hardware-Beschleunigers unabhängig von der Quelle der Trace-Daten widerspiegelt. Die gewonnenen Erkenntnisse und insbesondere die Analysen sind auch für ein später verfügbares Hardware-Trace verfügbar.

Konfigurationsanalyse

Mit Kenntnis der Adressen von Basisblöcken und zugehörigen Registern ist es möglich, die Konfiguration der Analyse allein anhand der beobachteten Schreibzugriffe auf die DFA-Komponente zu rekonstruieren. Dieses Hintergrundwissen kann aus den IP-XACT-Daten sowie zielsystemspezifischen Speicherabbildungen gewonnen werden. Aus den Speicherzugriffen lässt sich der Registerinhalt der angesprochenen Basisblöcke rekonstruieren. Und anhand dieser Registerinhalte kann auch die interne Belegung der Interconnect-Matrix abgeleitet werden. Diese Informationen ermöglichen den Aufbau eines Datenabhängigkeitsgraphen auf Basis der verwendeten Basisblöcke. Mit weiterem Wissen über die Funktionsweise der einzelnen Basisblöcke ließ sich zudem zeigen, dass sich auch der ursprünglich ausgeführte Algorithmus rekonstruieren lässt.

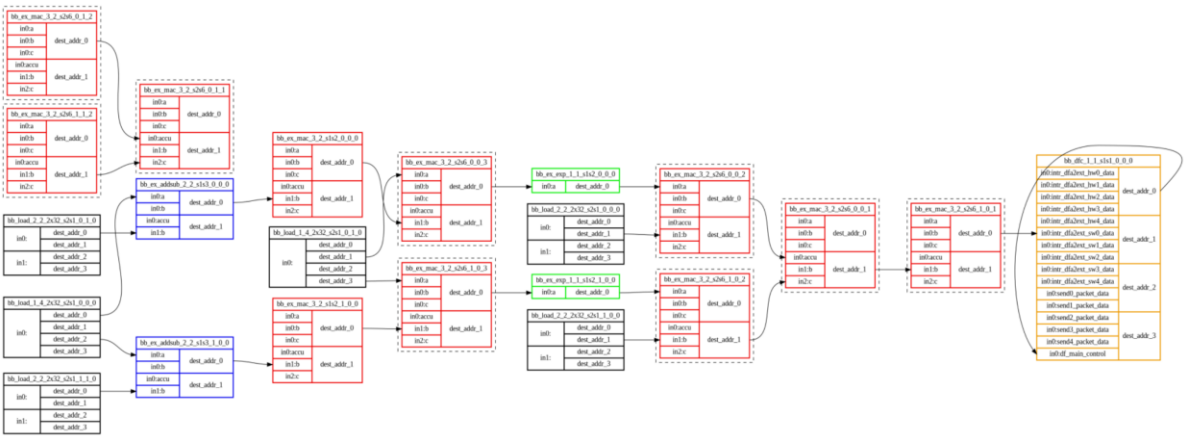


Abbildung 4 Rekonstruierter Datenabhängigkeitsgraph aus dem Software-Log

Im aktuellen Entwicklungsstand ist die Konfiguration in der Regel statisch und kann durch Auslesen der Register des DFA vor der Ausführung ermittelt werden. Der DFA bietet jedoch das Potenzial zur dynamischen Rekonfiguration, sodass in zukünftigen Analysen möglicherweise unterschiedliche Konfigurationen für die aufgezeichneten Trace-Daten rekonstruiert werden müssen. Besonders in einem solchen Fall kann die Rückerkennung der Konfiguration aus dem Trace von großem Wert sein.

Zeit- und Performanceanalysen

Die detaillierten Zeitstempel für jeden Zugriff auf Basisblöcke – sowohl für Eingabe- als auch Ausgabedaten – ermöglichen Performance-Analysen auf mehreren Ebenen: Basisblock-Operation, Sub-Algorithmus, Algorithmus und Speicherbus.

Die Analyse der Zeitstempel von Ein- und Ausgaben kann genutzt werden, um statistische Kennwerte wie minimale, maximale und durchschnittliche Verarbeitungszeit zu berechnen – ausgehend von a) dem Zeitpunkt der Verfügbarkeit der ersten benötigten Eingabedaten und b) dem Zeitpunkt der letzten benötigten Eingabedaten. Diese Statistiken ermöglichen es, Engpässe im Algorithmus oder mögliche Schwächen in der Implementierung eines Basisblocks zu identifizieren.

Mehrere im DFA-Benutzerhandbuch beschriebene Beispiele basieren auf komplexen Algorithmen, die sich aus Verkettungen mehrerer Sub-Algorithmen zusammensetzen – vergleichbar mit mathematischen Formeln, die aus Teilformeln oder mehreren Rechenschritten bestehen. Durch das Filtern der Ein- und Ausgabenachrichten von Basisblöcken, die die Argumente und Ergebnisse eines Sub-Algorithmus darstellen, kann der Fokus gezielt auf diese Teilschritte gelegt werden, um deren spezifische Performanceeigenschaften im Kontext des Gesamtalgorithmus zu analysieren.

Die Gesamtleistung eines Algorithmus hängt unter anderem von der Reihenfolge der Konfiguration der Blöcke ab. Da die DFA-Basisblöcke parallel und unabhängig voneinander arbeiten, können Teile der Analyse bereits ausgeführt werden, während andere noch konfiguriert werden. Ausgehend von der bekannten Konfiguration (z. B. aus vorangegangenen Analysen oder Benutzereingaben) kann durch die Analyse von Datenabhängigkeiten und der Verfügbarkeit von Eingabedaten eine Optimierung des Konfigurationsalgorithmus erfolgen, um Latenzen zu minimieren und den Durchsatz zu maximieren.

Die Analyse der Zeitstempel von Lese- und Schreiboperationen kann zur Identifikation von Performance-Engpässen im Zusammenhang mit dem Speicherbus dienen. Staus oder Blockierungen bei Speicherzugriffen können Rückstaueffekte (Backpressure) im Algorithmus verursachen und zu Verzögerungen im Gesamtprozess führen.

Die Kombination der Datenabhängigkeiten pro Basisblock ermöglicht eine Technik namens Slicing, um die Herkunft eines bestimmten Ausgabewerts zu analysieren. Dabei wird aus dem Abhängigkeitsgraph ein Graph konstruiert, der Kanten von Eingabedaten zu Ausgabedaten enthält. Der Benutzer kann einen bestimmten Ausgabewert angeben, und die Analyse rekonstruiert alle Zwischenberechnungen und aus dem Speicher gelesenen Eingabedaten, um die Eingabewerte oder Operationen zu finden, die zum untersuchten Ergebnis geführt haben.

Trace-Aufzeichnungsschema

Aufgrund von Weiterentwicklungen durch den Projektpartner Bosch hat sich das Format der Logdateien im Projektverlauf geändert. Daher wurde eine Versionsnummer für das Logformat eingeführt, die zu Beginn der Ausführung ausgegeben wird. Ein Parser-basierter Ansatz mit regulären Ausdrücken wurde entwickelt, um die Logausgaben zu verarbeiten und daraus eine interne Repräsentation zu erzeugen.

Das Konzept des DFA-Tracings lässt sich als ein Strom von Einträgen (Records) beschreiben. Einträge sind Datensammlungen, deren Inhalt vom Typ des Basisblocks abhängt, der die Daten erzeugt hat. Einige Blöcke können verschiedene Arten von Einträgen erzeugen, z. B. bei der Ein- oder Ausgabe von Daten an einem Port. Ein Eintrag enthält alle notwendigen Informationen, um die enthaltenen Daten physisch und logisch im Datenflussgraphen einzuordnen.

Eine Aufzeichnung (Recording) kann durch einen eindeutigen Namen identifiziert werden. Derzeit ist keine Versionsnummerierung vorgesehen. Eine Aufzeichnung besteht aus einer zeitlich (schwach) geordneten Sequenz von Einträgen. Es wird empfohlen, Zeitstempel als Nanosekunden-Zähler zu definieren, beginnend mit dem Start des Hardware-Beschleunigers. Zukünftige Hardware-Trace-Einheiten könnten jedoch andere Mechanismen verwenden, etwa einen Taktzähler auf Basis einer festen Frequenz. Die einzige harte Anforderung ist derzeit die schwache zeitliche Ordnung der Einträge anhand eines 64-Bit-Zeitstempels (unsigned).

Folgende Eintragstypen (Records) wurden identifiziert: Load, Store, Event, Register Access, In-Port, Out-Port, Memory Access, Memory Read, Memory Write. Die Einträge können verschiedene Datentypen enthalten, Wird ein bestimmter Datentyp verwendet, sollte dessen Verwendung konsistent zu anderen Einträgen desselben Typs erfolgen.

Datentypen

Einer der beiden Datentypen, die in allen Einträgen gemeinsam vorkommen, ist der Zeitstempel. Er sollte als 64-Bit unsigned Integer vorliegen und entweder einen Taktzähler oder einen Nanosekundenwert in Bezug auf einen implizit bekannten Offset repräsentieren. Eine

Implementierung sollte berücksichtigen, dass dieser Wert bei sehr großem Offset während der Aufzeichnung überlaufen kann. In diesem Fall sollte der Zählerwert entweder implizit auf mehr als 64 Bit erweitert oder auf einen kleineren Offset zurückgesetzt ("rebased") werden, da andernfalls die schwache zeitliche Ordnung der Einträge verletzt würde.

Die Schleifenebene (Loop Level) beschreibt immer die innerste der möglichen verschachtelten Schleifenebenen. Aufgrund des beschränkten Wertebereichs in den Registerdefinitionen reicht eine Darstellung als 32-Bit unsigned Integer aus.

Obwohl alle bekannten Hardwareversionen des DFA auf 32-Bit breite Adressbusse ausgelegt sind, ist absehbar, dass zukünftig 64-Bit breite Adressbusse verwendet werden. Daher wurde als Adresstyp ein 64-Bit unsigned Integer gewählt. Bei der Umwandlung in ein Stringformat sollte eine hexadezimale Schreibweise bevorzugt werden, wobei führende Nullen weggelassen werden.

Der Block-Identifier sollte eine eindeutige Repräsentation der identifizierenden Merkmale eines Basisblocks sein: Typ, Instanz, Split, Set Wird der Identifier als String dargestellt, wird folgendes Format empfohlen:

<type>.instance<instance>.split<split>.set<set>

Dabei bezeichnet <type> den Typ des Basisblocks, z. B. „bb_ex_addsub_2_1_s1s3“, „bb_ex_mac_3_2_s1s2“ usw. Die Werte <instance>, <split> und <set> sind dezimale Integer, die an den String angehängt werden, um ihre Bedeutung zu verdeutlichen. Diese Stringdarstellung kann in Parsern verwendet werden, um die enthaltenen Informationen wieder in die binäre Form zurückzuwandeln. Ein vollständiges Beispiel wäre z. B.:

bb_ex_mac_3_2_s1s2.instance0.split0.set0

Die Ganzzahldarstellung des Block-Identifiers kann gebildet werden, indem für die einzelnen Felder feste Bitbereiche reserviert und in definierter Reihenfolge zusammengesetzt werden – ähnlich der Struktur der Stringdarstellung. Der Typ kann dabei durch eine Enumeration abgebildet werden, wobei jedem Blocktyp ein fortlaufender Wert zugewiesen wird.

Das folgende Beispiel beschreibt die gewählte Kodierung für die DFA-Version 4.7.0.1, basierend auf den ermittelten Maximalwerten für Split, Set, Instanz und Typ:

8	7	6	5	4	3	2	1	0
set			split	instance	identifier			

Abbildung 5 Bitposition und -breite der Identifikator-Typen

Die erste Zeile stellt die Bitposition und die zweite Zeile den zugewiesenen Identifikator-Typ dar.

Der Port-Identifier wird in In-Port- und Out-Port-Einträgen verwendet und kann entweder als unsigned Integer oder als Zeichenkette dargestellt werden. Da maximal vier Ports pro Basisblock verwendet werden, ist ein 32-Bit unsigned Integer ausreichend, auch für zukünftige Versionen der DFA-Architektur. Der Port-Typ (ein- oder ausgehend) ergibt sich aus dem Typ des Eintrags, sodass keine zusätzliche Kennzeichnung erforderlich ist. Wird der Port-Identifier als Zeichenkette übertragen, sollte der Typ des Ports im Format enthalten sein, zusammen mit der Portnummer:

<Typ><Identifikator>

Beispielsweise: "in0", "out2" usw.

Dies dient der besseren Lesbarkeit durch Menschen, so dass alle notwendigen Informationen ohne zusätzliche Quellen direkt sichtbar und die Zeichenketten durchsuch- und sortierbar sind.

Der Core-Identifier wird bei Memory Read- und Memory Write-Einträgen verwendet, die Speicherzugriffe eines Anwendungskerns auf den DFA-Speicher darstellen. Der Identifier sollte sich an der Darstellung innerhalb der Anwendung orientieren. Wird ein Integer verwendet, sollte eine Zuordnung zur Zeichenkettenrepräsentation möglich sein. Die String-Darstellung sollte dem in der Anwendung verwendeten Namen des Kerns entsprechen – idealerweise dem, der auch in den verwendeten Tools verfügbar ist. Der Name sollte über mehrere Aufzeichnungen hinweg konsistent bleiben.

Der Register-Identifier als 32-Bit unsigned Integer wird aus der Zugriffsadresse berechnet: Es ist die Adresse abzüglich des Offsets des DFA-Adressbereichsanfangs, um 2 Bits nach rechts verschoben:

RI(Register) := ((Adress of Register) – (Adress of first DFA register)) >> 2

Da alle Register wortweise ausgerichtet sind (die letzten zwei Bits der Adresse sind null), ist diese Berechnung umkehrbar. Somit besteht eine direkte Zuordnung zwischen Identifier und Speicheradresse des Registers. Wird eine Zeichenketten-Darstellung verwendet, ist der Name des Registers aus der IP-XACT-Beschreibung zu verwenden. Da Blöcke desselben Typs identische Registerbezeichner besitzen, ist der Name nicht global eindeutig.

Der Event-Typ identifiziert die Art des Ereignisses in einem Eintrag. Er wird durch den Event-Identifier ergänzt, der einzelnen Ereignisse derselben Art unterscheidet. Der Event-Typ ist aufzählbar (enumerierbar) und kann als unsigned Integer oder Zeichenkette dargestellt werden. Bekannte Typen sind z. B.: Interrupt, Reset und Interne Signale.

Ein Event-Identifier dient zur Unterscheidung von Ereignissen derselben Art, z. B. Interrupt-Leitungen, Signalleitungen, Trigger oder Watchpoints. Es handelt sich um einen 32-Bit unsigned Integer. Eine benannte Zeichenketten-Darstellung ist ebenfalls möglich.

Eintragstypen

Ein Load-Eintrag steht für einen Lesezugriff eines „Load“-Basisblocks auf eine Speicheradresse des Zielsystems und enthält: Zeitstempel, Schleifenebene, Speicheradresse, Block-Identifier, Gelesene Daten. Der Zeitstempel beschreibt den Zeitpunkt, an dem der Lesezugriff abgeschlossen wurde. Wird ein Zugriff nicht erfolgreich durchgeführt, wird kein Load-Eintrag erzeugt – was als Indikator für einen Fehlschlag gilt.

Ein Store-Eintrag steht für einen Schreibzugriff eines „Store“-Basisblocks auf eine Speicheradresse des Zielsystems und enthält: Zeitstempel, Schleifenebene, Speicheradresse, Block-Identifier, Geschriebene Daten. Obwohl die Daten bereits vor Abschluss des Schreibvorgangs bekannt sind, wird der Eintrag nur bei erfolgreichem Abschluss erzeugt. Ein fehlender Store-Eintrag deutet auf einen fehlgeschlagenen Zugriff hin.

Ein Registerzugriff-Eintrag (Register Access) dokumentiert Schreibzugriffe auf Konfigurationsregister. Er wird durch die Zuordnung der Registeradresse zum Register-Identifier erzeugt und enthält: Zeitstempel, Register-Identifier, Block-Identifier, Geschriebene Daten. In den Log-Dateien überschneiden sich Registerzugriffe teilweise mit In-Port-Meldungen. Die Unterscheidung erfolgt über die Register-Identifier. Zukünftige DFA-Versionen könnten jedoch Konfigurationsregister auch für Out-Port-Pakete verwenden, was zu Konflikten führen könnte.

Ein In-Port-Eintrag wird erzeugt, wenn ein neuer Datenwert am In-Port eines Basisblocks ankommt. Dieser Eintrag ist möglicherweise in Hardware-Traces nicht verfügbar, da dort nur Out-Port-Einträge vorgesehen sind. Er stammt aus den SystemC-Log-Dateien und enthält: Zeitstempel, Schleifenebene, Port-Identifier, Block-Identifier, Empfangene Daten.

Ein Out-Port-Eintrag wird erzeugt, wenn ein neuer Datenwert am Out-Port eines Basisblocks ausgegeben wird und enthält: Zeitstempel, Schleifenebene, Port-Identifier, Block-Identifier, Übertragene Daten. Wird ein Wert auf mehreren Out-Ports ausgegeben, wird für jeden Port ein Eintrag erzeugt. Bei mehreren verbundenen In-Ports wird nur ein Eintrag erzeugt.

Ein Memory-Access-Eintrag wird erzeugt, wenn ein Datenzugriff erfolgt, der nicht durch einen Anwendungskern, sondern z. B. durch DMA innerhalb des DFA erfolgt und enthält: Zeitstempel, Zugriffsadresse, Daten. Dieser Eintrag ist nicht für Blocktransfers gedacht, sondern für Einzelübertragungen. Informationen zu Blocktransfers liegen nicht vor, da sie im SystemC-Log nicht erscheinen. Eine denkbare Struktur für zukünftige Blocktransfers wäre: Zeitstempel, Quelladresse, Zieladresse, Datenlänge, Datenblock (als Datenvektor), welche die Länge der Blockübertragung und die Daten selbst als reinen Datenvektor angibt.

Ein Memory Read - Eintrag beschreibt einen Datenzugriff vom DFA-Speicher auf einen Anwendungskern und enthält: Zeitstempel, Zugriffsadresse, Core-Identifier und Daten.

Ein Memory Write – Eintrag beschreibt einen Datenzugriff vom Anwendungskern auf den DFA-Speicher und enthält: Zeitstempel, Zugriffsadresse, Core-Identifier und Daten

Event-Einträge transportieren Informationen über seltene Ereignisse wie Interrupts, Reset, Signale etc. Zwei Elemente identifizieren das Ereignis: Typ und Identifier. Der Event-Eintrag enthält: Zeitstempel, Schleifenebene, Event-Typ, Event-Identifier, Block-Identifier. Optional kann ein Datenfeld enthalten sein, das z. B. einen neuen Zustand transportiert.

Serialisierung

Begleitend zu den verschiedenen Möglichkeiten, auf DFA-Aufzeichnungen zuzugreifen und diese darzustellen, wurde ein Format zur Serialisierung entwickelt. Diese erfüllt mehrere Zwecke: Kommunikation zwischen Werkzeugen, Eingabe für Tests, Persistente Speicherung von Ergebnissen und Unterstützung beim Debugging. Das Format basiert auf XML und ist in einem Schema beschrieben. Aktuell können zwei Arten von Trace-Aufzeichnungen importiert werden: System-C – Logdateien und DFA-Aufzeichnungen im entwickelten XML-Format. Eine Aufzeichnung kann außerdem im XML-Format exportiert werden – der Prototyp fungiert damit effektiv als Konverter von System-C - Logdateien in das XML-format.

Erstellung eines Demonstrators

Im Zuge der Analyse der Logdateien wurde ein Demonstrator erstellt, der die oben beschriebenen Daten aus den vorhandenen Logdateien extrahiert. Der Demonstrator konnte zeigen, dass folgende Analysen möglich sind:

- Rekonstruktion der Konfiguration aus den in Konfigurationsregister geschriebenen Daten
- Extraktion von Zwischenzeitstempeln sowie Ein- und Ausgabedaten der Basisblöcke
- Erstellung einer Zeitachse der Datenereignisse
- Erzeugung eines Datenabhängigkeitsgraphen für einen bestimmten Ausgabewert (Slice)

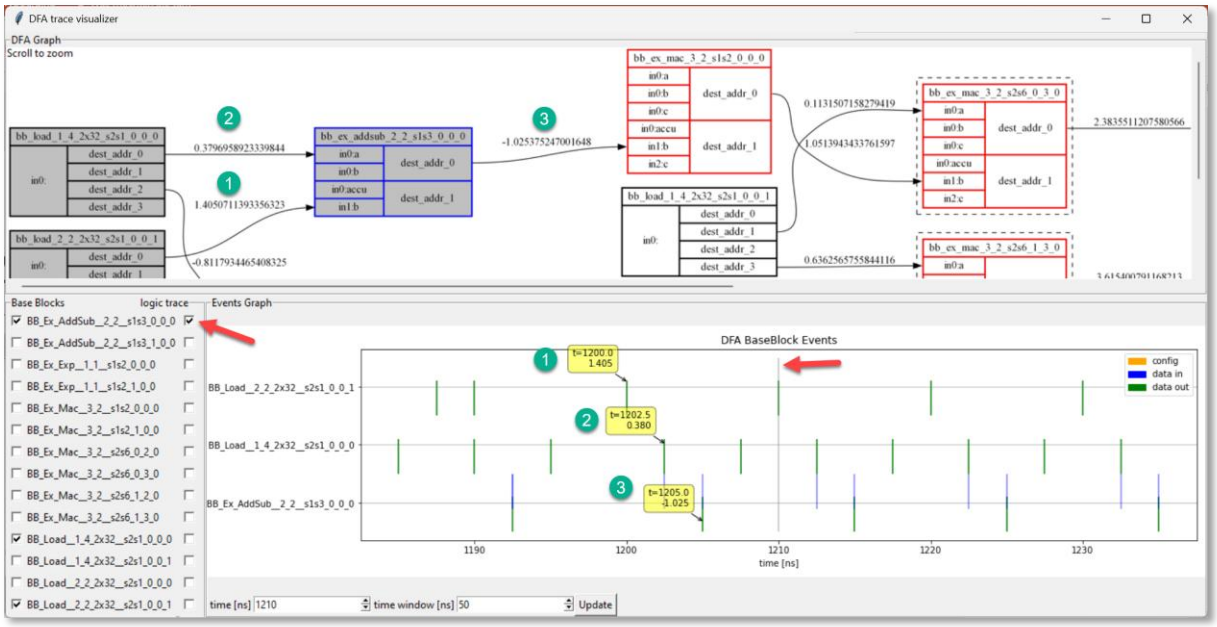


Abbildung 6 Grafische Darstellung der Konfiguration und der Zeitachse der Datenereignisse, extrahiert aus den Konfigurationsdaten

1.2 AP 3: Methoden und Werkzeuge zur Anwendungsvalidierung

Definition und prototypische Implementierung von Methoden und Schnittstellen zur Bereitstellung von Laufzeitdaten des KI-Hardwarebeschleunigers für die Validierung durch die Modellierungs-Werkzeuge

Definition und Implementierung von Schnittstellen zur Übernahme von Modelldaten für den Debugger zur Testung von Algorithmen und Modellen und zur Steuerung von anderen Werkzeugen beispielsweise zur Visualisierung des Modellstatus

Im Rahmen des Projekts wurden verschiedene Werkzeuge von unterschiedlichen Projektpartnern entwickelt, dennoch sollte eine gemeinsame Nutzung möglich sein. Dafür müssen die Werkzeuge über Schnittstellen verfügen, die eine solche Integration erlauben.

Zu den derzeit verfügbaren Werkzeug-Prototypen zählen eine integrierte Entwicklungsumgebung (Projektpartner itemis AG), ein Compiler (Projektpartner OFFIS) und eine Debugger-Komponente (PLS GmbH). Es wurden Schnittstellen definiert, die eine Kommunikation zwischen diesen Werkzeugen ermöglichen. Dabei wurden universelle Konzepte mit guter Erweiterbarkeit gewählt damit in Zukunft weitere Werkzeuge ebenfalls diese Schnittstellen nutzen können.

Ein Debugger bietet zahlreiche Möglichkeiten, Daten über den aktuellen Zustand und/oder das Verhalten von Hard- und Software zu erfassen. Für die DFA bedeutet das, dass der Debugger die Ausführung steuern und Informationen über die aktuelle Konfiguration bereitstellen kann. Die Schnittstellen sollen es ermöglichen, auch aus anderen Werkzeugen auf diese Informationen zuzugreifen. Der Debugger stellt Drittsystemen zwei Möglichkeiten zur Verfügung, um auf den Hardware-Beschleuniger zuzugreifen: Zugriff über ein objektbasiertes COM-Interface und Zugriff über ein Ethernet-basiertes Kommunikationsprotokoll für andere Werkzeuge. Das Debug-Protokoll wird durch die "Artifact-Trace" - Definition ergänzt, die als gemeinsame Basis für die Kommunikation zwischen verschiedenen Werkzeugen dient.

Artifact-Trace

Im Rahmen des Projekts wurde ein Datenformat namens Artifact-Trace entwickelt, das Referenzen zu den Ursprüngen der aktuellen Konfiguration der DFA enthält. Artifact-Trace wird verwendet, um einen einheitlichen Datensatz aufzubauen, der die Kommunikation zwischen den verschiedenen Werkzeugen auf der Grundlage eines einheitlichen Schemas ermöglicht. Die Artifact-Reference ist eine abstrakte Definition, die beschreibt, wie ein Artefakt identifiziert werden kann, das Teil des Kompilierungsprozesses war (z. B. Position im GALEC-Code, DFG-Knoten, DFA-Register usw.).

Artifact Trace ist ein Dateiformat (siehe Anhang II), das Debug-Informationen (Backtrace) enthält, die während des Kompilierungsprozesses erzeugt wurden. Es beinhaltet Verweise auf alle Artefakte, die zur Erstellung der DFA-Konfiguration verwendet und/oder generiert wurden. Der Backtrace besteht aus Elementen und Referenzen zwischen diesen Elementen, sodass jede Komponente in der Toolchain in der Lage ist, ein Low-Level-Element einem High-Level-Element zuzuordnen – und umgekehrt.

Beispiel: Ein Algorithmus, geschrieben in GALEC, wird in einen Data Flow Graph (DFG) kompiliert. Der DFG wird anschließend in eine DFA-Konfiguration übersetzt. Diese Konfiguration dient als Grundlage zur Erstellung von C-Code, der aus einem Konfigurationsvektor mit Adressen und Datenwerten sowie Code besteht, der diese Vektoren in die DFA-Hardware schreibt. Die C-Datei wird schließlich in Binärcode kompiliert, der auf dem Zielprozessor ausgeführt wird.

Jeder Transformationsschritt zeichnet die Beziehung zwischen Eingabe- und Ausgabedaten in einem XML-Format auf – mit Identifikatoren und Rückverweisen von den Ausgabeelementen zu den Eingabeelementen. Das Ergebnis wird als "Artifact Trace" bezeichnet. Dieser enthält zum Beispiel ein "DfaRegisterSettingReference" als "originArtifactReference" sowie Verweise auf alle Zwischenmodelle ("GalecCodeFragmentReference", "DfgElementReference", "DfaConfigurationElementReference").

Die resultierenden Dateien können über den Kontextmenüeintrag "Artifact Trace/Load Artifact Trace Symbols" in die GUI geladen werden. Die Informationen aus diesem Artifact-Trace werden in der Kommunikation des "DFA Debug Protocol" verwendet (wie zuvor beschrieben). Die Verwendung des Artifact-Trace als gemeinsame Referenz ermöglicht es allen Werkzeugen (z. B. IDE, interaktiver Compiler usw.), miteinander über Artefakt-Verweise aus ihrem jeweiligen Bereich zu kommunizieren.

Werkzeug-Kommunikationsprotokoll

Der Debugger stellt einen Web-Socket-Server bereit, der auf einem konfigurierbaren lokalen Host-Port geöffnet wird. Dieser Server ermöglicht die Kommunikation mit Drittsystemen über ein speziell entwickeltes Protokoll, das als Debug-Protokoll für die DFA-Komponente konzipiert wurde. Dieser Ethernet-basierte Ansatz dient als Alternative zur COM-Technologie und ist besonders nützlich für Drittsysteme, die keinen Zugriff auf COM haben – etwa aufgrund der Verwendung anderer Programmiersprachen oder Betriebssysteme.

Das Debug-Protokoll definiert, wie Nachrichten, die zwischen dem Debugger und anderen Werkzeugen ausgetauscht werden, aufgebaut sein müssen. Das Protokoll ist als objektorientiertes XSD-Schema definiert. Die zentrale Struktur des DFA-Debug-Tool-Protokolls ist die Message, die als abstrakte Basisklasse für alle weiteren abstrakten oder konkreten Nachrichtentypen dient. Sie legt fest, dass jede Nachricht einen Header enthalten muss, der mindestens folgende Informationen umfasst: Name des Werkzeugs, das die Nachricht sendet, Versionsnummer des sendenden Werkzeugs, Zeitpunkt, zu dem die Nachricht erzeugt wurde Jeder Nachrichtentyp kann sowohl vom Debugger als auch vom verbundenen Client verwendet werden, sofern in der Definition nichts anderes angegeben ist. Ein Sender kann das Notification-Schema verwenden, um dem Empfänger etwas mitzuteilen, ohne eine Antwort zu erwarten.

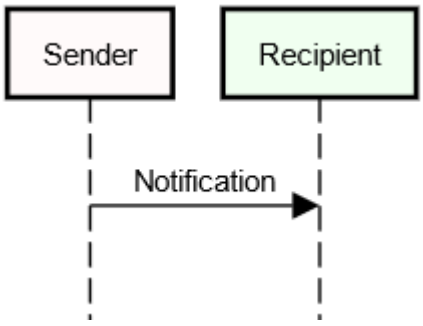


Abbildung 7 Notification-Schema

Ein Requester kann eine Request-Nachricht an einen Responder senden, um eine bestimmte Antwort anzufordern (z. B. angeforderte Daten, Ergebnis einer ausgeführten Operation usw.). Request-Schnittstellen sind immer als Paar aus einer Request- und einer Response-Nachricht definiert. Die Response-Nachricht enthält die Antwort auf die gestellte Anfrage. Eine Request-Nachricht wird verwendet, wenn der Absender eine Antwort vom Empfänger erwartet. Jede Request muss eine eindeutige Kennung enthalten, anhand derer der Absender die zugehörige Response zuordnen kann. Eine Response-Nachricht ist eine besondere Form der Notification, die vom Responder an den Requester gesendet wird, nachdem eine Request empfangen wurde.

Sie enthält die vom Requester angeforderten Daten. Jede Response muss die Kennung enthalten, die Teil der ursprünglichen Request war.

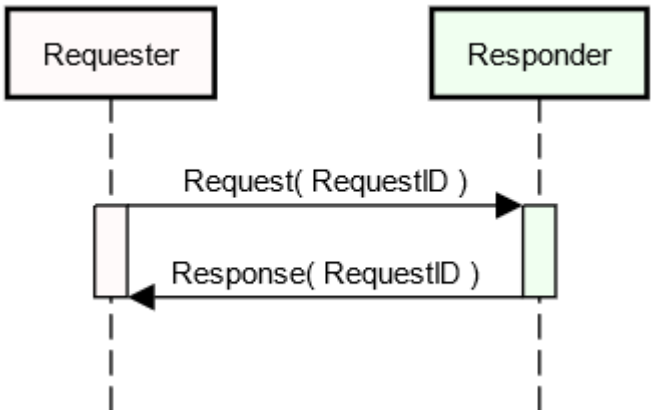


Abbildung 8 0 Request-Response-Schema

Ein Abonnent kann sich bei einem Abonnementanbieter für Benachrichtigungen anmelden. Abonnement-Schnittstellen sind immer durch eine Abonnementnachricht und mindestens eine Benachrichtigungsnachricht definiert. Um das Abonnement zu kündigen, muss der Abonnent eine Abmeldungsnachricht senden. Eine Abonnementnachricht ist eine spezielle Form einer Anfrage, die stets eine Abonnementantwort als Antwort erwartet. Eine Abonnementantwort ist eine spezielle Form einer Antwort, die zusätzlich die vom Abonnementanbieter vergebene Abonnement-ID enthält. Diese Abonnement-ID kann verwendet werden, um das Abonnement zu kündigen. Eine Abmeldung ist eine spezielle Benachrichtigung, mit der ein Abonnent dem Abonnementanbieter signalisiert, dass er das Abonnement beenden möchte.

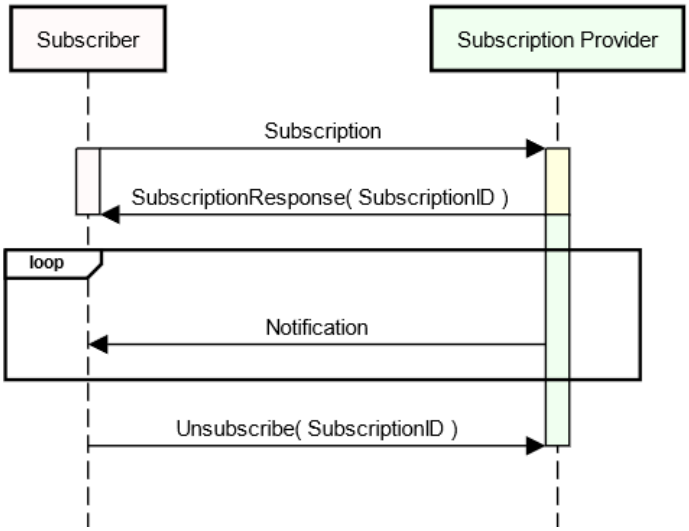


Abbildung 9 0 Subscription- (Abbonnement-) Schema

Es sind einige spezifische Nachrichtentypen definiert. Dabei handelt es sich um Proof-of-Concept-Definitionen, die prototypisch implementiert wurden. Diese Schnittstellen können bei Bedarf erweitert werden, wenn neue Anforderungen dies erforderlich machen.

Die Modellauswahl von Elementen dient zum Signalisieren, dass ein Element, das auf ein oder mehrere Artefakte aus "Artifact-Trace" verweist, z. B. in einer Benutzeroberfläche ausgewählt wurde. Diese

Schnittstelle ist als Abonnement-Schnittstelle definiert. Eine Modellauswahl-Benachrichtigung ist eine spezielle Benachrichtigung, die Artefaktverweise des Elements enthält, dessen Auswahlstatus sich geändert hat, sowie ein Kennzeichen, das den neuen Auswahlstatus signalisiert.

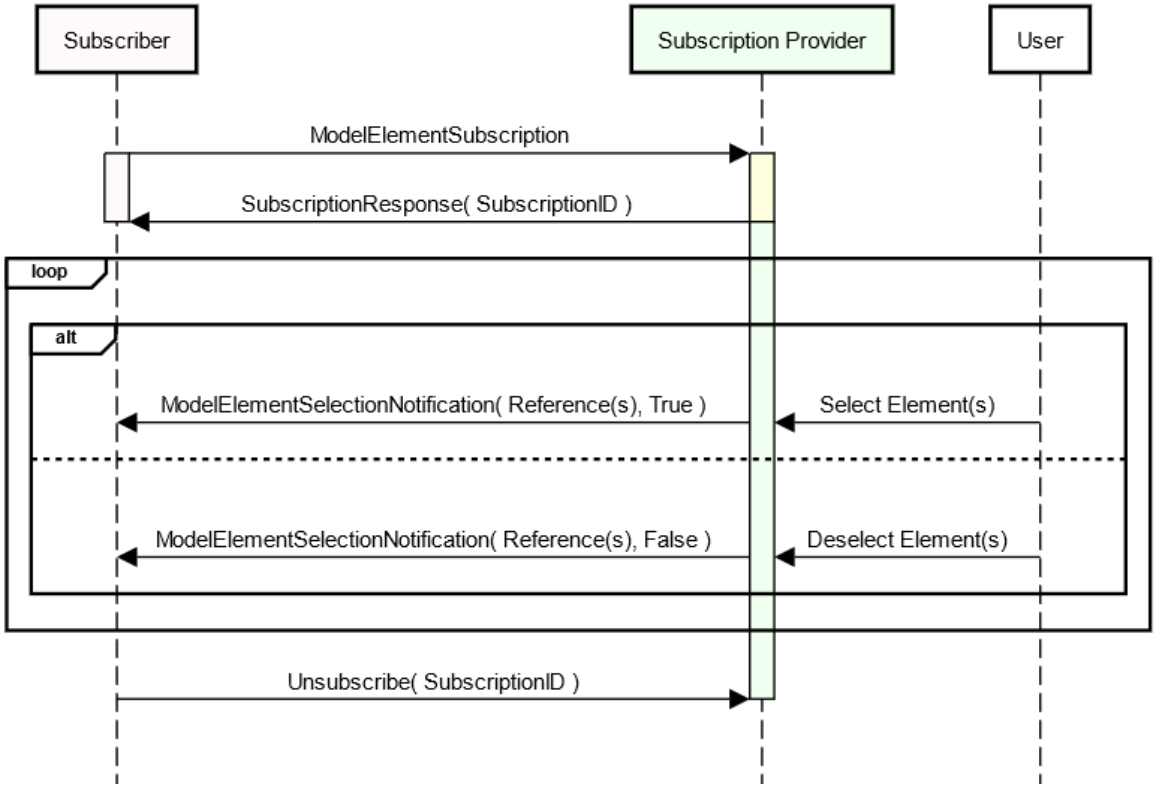


Abbildung 10 Modellelementauswahl-Schema

Eine weitere Schnittstelle dient zur Abfrage des aktuellen Zustandes der DFA. Die möglichen Zustände der DFA sind:
Getrennt - Der Debugger hat keine Verbindung zum Zielsystem und/oder kann nicht auf die erforderlichen Register zugreifen, um die DFA zu überwachen.
Läuft - Die DFA ist derzeit aktiv (unabhängig davon, ob aktuell ein Algorithmus ausgeführt wird).
Angehalten - Die DFA befindet sich im Debug-Modus.
Diese Schnittstelle sollte ausschließlich verwendet werden, um den Debugger nach dem aktuellen DFA-Zustand zu fragen. Sie ist als Anfrageschnittstelle definiert. Die DFA-Zustandsantwort ist eine spezielle Antwort, die einen Enumerationswert enthält, der angibt, ob der Debugger auf den DFA-Zustand des Zielsystems zugreifen kann und – falls ja – den aktuellen Zustand der DFA.

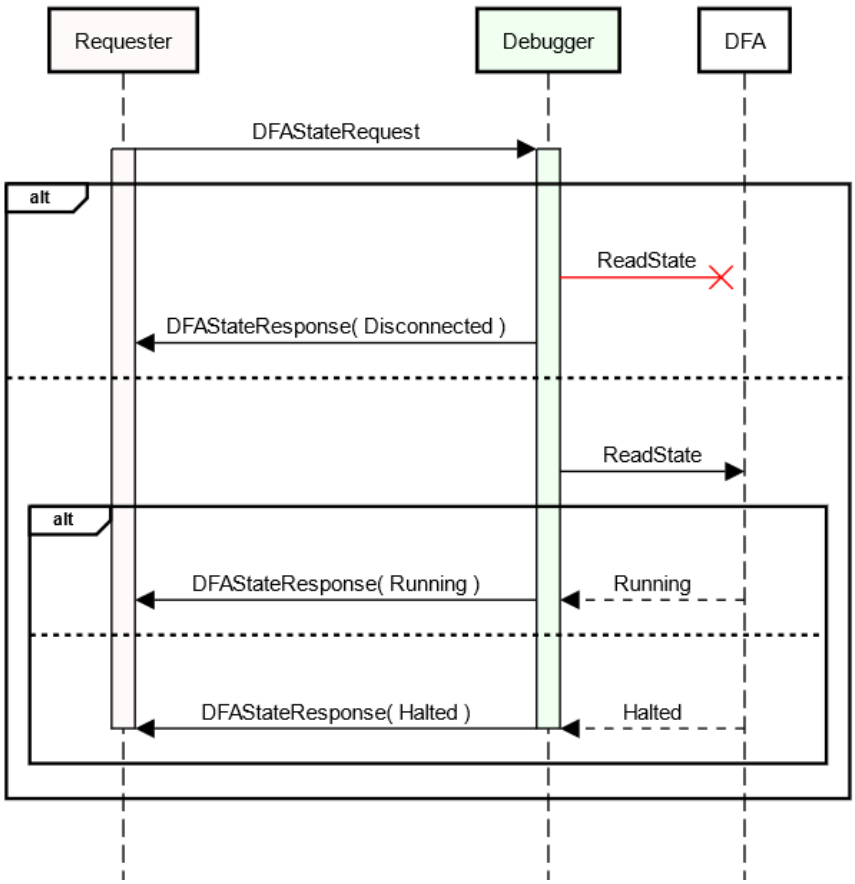


Abbildung 11 DFA-Zustandsantwort-Schema

Die Schnittstelle Algorithmus-Ergebnis dient dem Austausch von Informationen über die Ergebnisse von (Teil-)Algorithmen, die auf der DFA ausgeführt wurden. Sie ist ausschließlich für Debugging-Zwecke vorgesehen und als Subskriptionsschnittstelle implementiert. Die Benachrichtigung enthält den Namen des ausgeführten Algorithmus sowie die dazugehörigen Ein- und Ausgabedaten. Aktuell werden diese Daten in Form von Rohdatenabbildern bereitgestellt. Zukünftig kann dies durch strukturierte Datenrepräsentationen ersetzt werden, sobald entsprechende Laufzeitstrukturen der DFA-Instanz verfügbar sind – beispielsweise über Artifact-Tracing oder andere Debugging-Metadatenformate (z. B. ELF/DWARF).

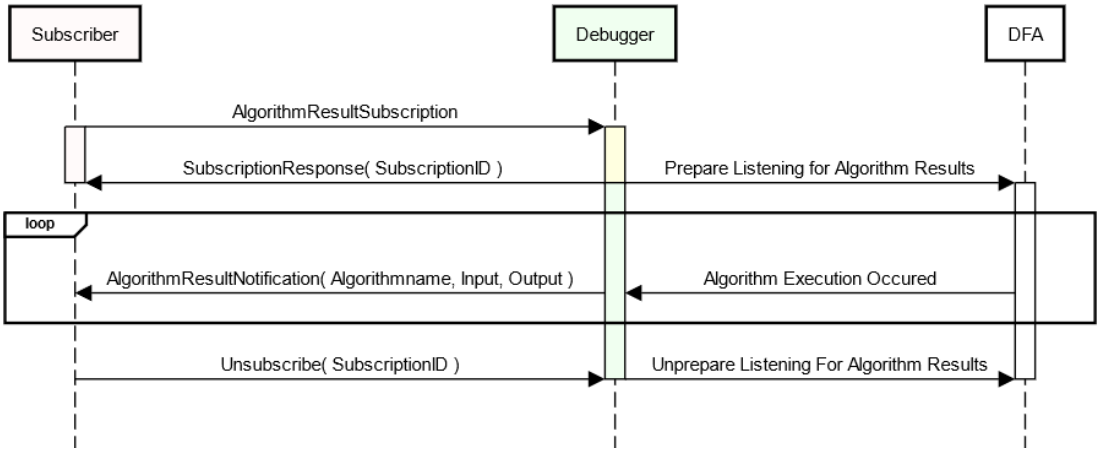
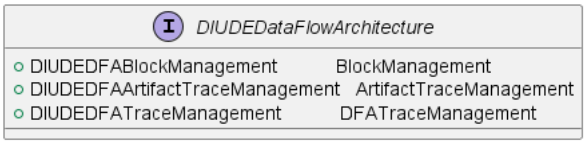
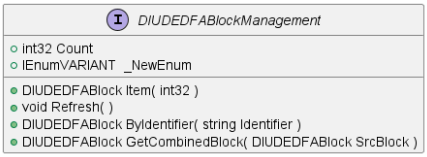
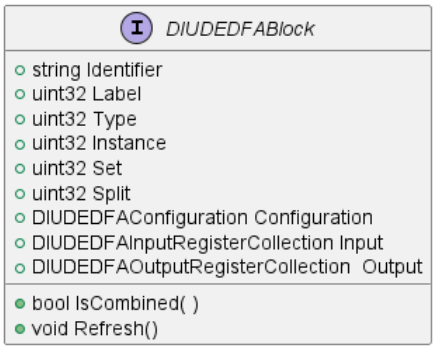
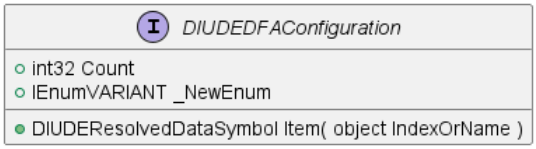


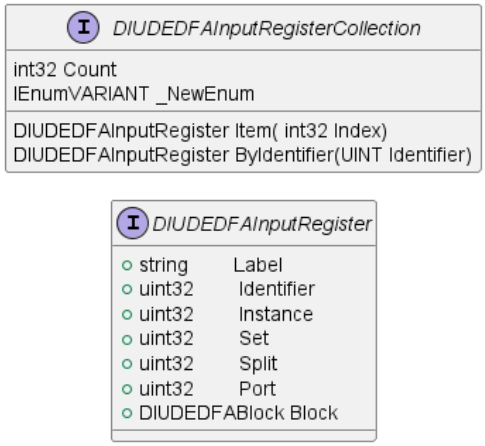
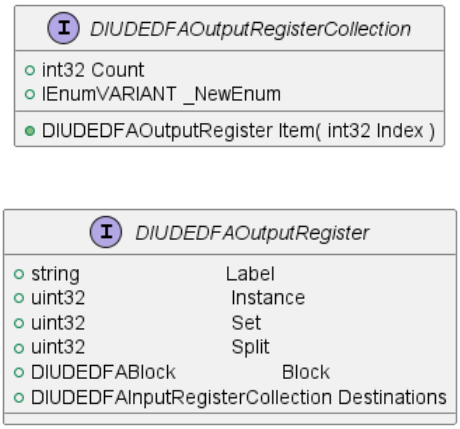
Abbildung 12 Algorithmus-Result-Schema

Derzeit sind nicht alle Anwendungsfälle bekannt. Daher müssen diese Schnittstellen in hohem Maße erweiterbar sein. Das Schema, in dem die Schnittstellen definiert sind, ist objektorientiert aufgebaut, und die Basisschnittstellen (Message, Notifikation, Request usw.) lassen sich einfach erweitern. Dies kann entweder durch zusätzliche Spezifikationen innerhalb des ursprünglichen Schemas oder durch Definitionen in zusätzlichen Schemata erfolgen, die auf das ursprüngliche Schema verweisen.

Component-Object-Model

Interface	UML-Notation
Das Wurzelelement des DFA-Modells ist das Interface **DIUDEDataFlowArchitecture**. Es stellt Drittsystemen Zugriff auf den DFA-Hardwarebeschleuniger bereit, einschließlich einer Sammlung der Basisblöcke, einer Komponente zur Verwaltung von Artefakt-Verfolgungen und einer Komponente zur Verwaltung von Trace-Aufzeichnungen. Die Modellstruktur wurde so gewählt, dass sie den Zugriff auf die verschiedenen Aspekte der Arbeit mit dem DFA-Hardwarebeschleuniger ermöglicht.	
Das Interface zur Basisblock-Sammlung (**DIUDEDFABlockManagement**) stellt die Eigenschaft Count und die Methode Item für den direkten Zugriff auf einzelne Blöcke bereit; die Eigenschaft _NewEnum erlaubt die Aufzählung der DFA-Basisblöcke. Zusätzlich bietet das Interface Suchfunktionen sowie die Möglichkeit, den Datenflussgraphen manuell neu aufzubauen.	
Das Basisblock-Interface **DIUDEDFABlock** enthält die identifizierenden Eigenschaften eines Basisblocks (Label, Split, Set, Instanz), den Blocktype (Type) sowie die Registerkonfiguration (Configuration). Die identifizierenden Elemente werden als Zeichenkette im Identifier zusammengefasst. Die Datenflussgraph-Darstellung basiert auf den Sammlungen von Eingangs- und Ausgangsregistern (Input, Output), die nach einer Aktualisierung (Refresh) aus der aktuellen Registerkonfiguration berechnet werden. Sie bilden den aktuell konfigurierten Datenflussgraphen ab.	
Die Konfiguration eines Basisblocks (**DIUDEDFAConfiguration**) wird als Sammlung von Registern dargestellt.Sie unterstützt gezielten und sequentiellen Zugriff auf Register über Count, Item und _NewEnum.	

Ein Register wird durch das allgemeine nicht DFA-spezifische Interface DIUDEResolvedDataSymbol dargestellt, das Zugriff auf Namen, Typ und aktuellen Wert bietet. Zusätzlich erlaubt dieses Interface den Zugriff auf Registerfelder, einschließlich Enumerationswerten und -namen.

Interface	UML-Notation
Zur Abbildung der speziellen Eigenschaften des Datenflussgraphen wurden eigene Interfaces für die Eingangsregister-Sammlung (**DIUDEDFAInputRegisterCollection**) und für das Eingangsregister selbst (**DIUDEDFAInputRegister**) geschaffen. Als Eigenschaft eines Basisblocks stellen sie alle möglichen Eingangsregister dar. Als Eigenschaft eines Ausgangsregisters zeigen sie die Eingangsregister, die mit diesem Ausgang verbunden sind. Die Anzahl der Eingangsregister hängt vom Block-Typ ab. Die Sammlung unterstützt gezielten und sequentiellen Zugriff sowie eine Suchmethode, um ein Eingangsregister über denselben Bezeichner wie im Ausgangsregister zu finden. Die Bezeichner ergeben sich aus einer internen Berechnung der Speicheradresse relativ zum Beginn der DFA-Hardware. Sie können über eine Formel berechnet werden, die Set, Split und Instanz einbezieht. Das Eingangsregister-Interface bietet einen Direktzugriff auf die Eigenschaften des zugehörigen Basisblocks und Registers sowie auf den Block selbst.	
Die Sammlung der Ausgangsregister (**DIUDEDFAOutputRegisterCollection**) erlaubt gezielten und sequentiellen Zugriff auf alle Ausgangsregister eines Blocks. Die Anzahl hängt vom Typ des Basisblocks ab. Das Ausgangsregister-Interface (**DIUDEDFAOutputRegister**) bietet einen Kurz-Zugriff auf die Eigenschaften des zugehörigen Basisblocks und auf den Block selbst. Die zusätzliche Eigenschaft 'Destinations' ist eine Sammlung der verbundenen Eingangsregister. Der Inhalt dieser Sammlung wird aktualisiert, sobald sich der Wert des zugehörigen Registers ändert. Die Kombination aus Ausgangs- und Eingangsregister bildet die Kanten zwischen Blöcken und ihren Registern im Datenflussgraphen.	

Hardware-Beschleuniger sind technische Komponenten, die darauf spezialisiert sind, bestimmte Operationen besonders effizient auszuführen. KI-basierte Software kann diese Hardware-Beschleuniger nutzen, um Berechnungen effizienter durchzuführen. Um den Entwicklungs- und Debugging-Prozess für Hardware-Beschleuniger zu erleichtern, kann ein Debugger Informationen über den aktuellen Zustand des Hardware-Beschleunigers bereitstellen. Ein weiterer Vorteil besteht darin, dass ein spezialisierter Debugger die vom Hardware-Beschleuniger gesammelten Daten analysiert und visualisiert. Im Rahmen dieses Projekts wurden verschiedene Analyse- und Visualisierungsmethoden untersucht.

Die grundlegenden Analysefunktionen bieten einen strukturierten Überblick über den Hardware-Beschleuniger und seine Komponenten. Sie basieren auf einer Hardwarebeschreibung des Herstellers beispielsweise mit IP-XACT-Dateien. Diese Beschreibung wird analysiert und intern in ein Modell des Ziel-Hardware-Beschleunigers überführt. Durch diese Re-Konfigurierbarkeit können auch zukünftige Generationen des Hardware-Beschleunigers einfach unterstützt werden.

Die prototypische Software nutzt das Modell um die Konfiguration des Hardware-Beschleunigers in tabellarischer Form in der Konfigurationsübersicht darzustellen.

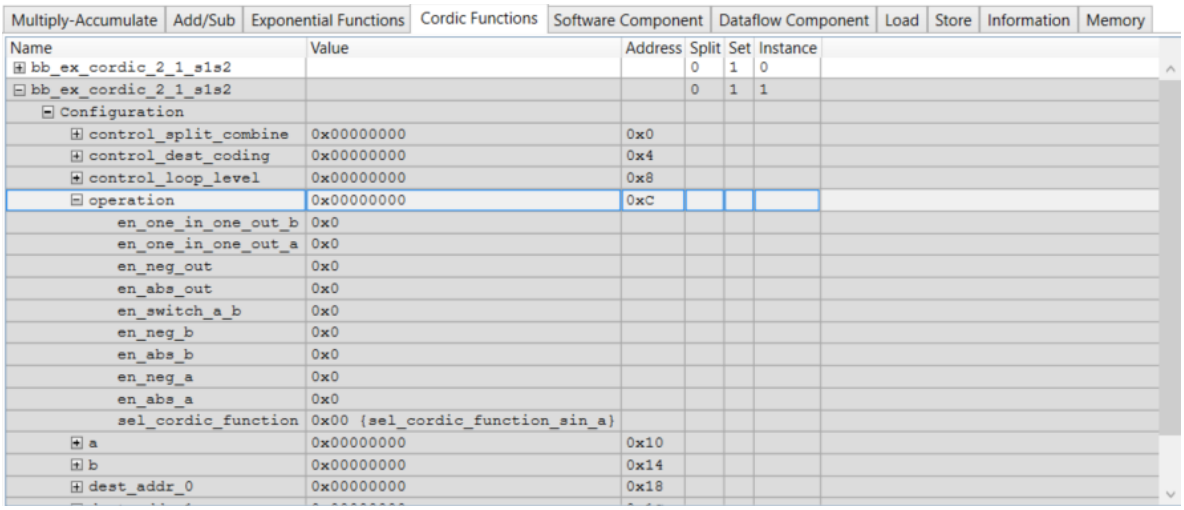


Abbildung 13 Konfigurationsübersicht zu Basisblöcken mit Splits sowie Register und Registerfelder

Die Namen und Typen von Blöcken, Registern und Feldern werden aus dem internen Modell übernommen. GUI-Elemente werden automatisch entsprechend den Spezifikationen des jeweiligen Feldes oder Registers erzeugt. Mit einem Feld verknüpfte Enumerationswerte werden angezeigt, um eine benutzerfreundliche Darstellung numerischer Werte zu ermöglichen.

Die Blöcke werden mit ihren identifizierenden Eigenschaften angezeigt: Name, Split, Set und Instanz-Kennung. Jeder Block enthält ein aufklappbares Unterelement "Konfiguration", das eine Liste der zugehörigen Register des Basisblocks enthält. Eine Spalte "Wert" zeigt den aktuellen Wert des Registers in hexadezimaler Notation an. Die Darstellung wird entsprechend der Bitbreite des Registers und/oder Feldes formatiert und akzeptiert nur Werte, die innerhalb der Spezifikation liegen.

Es werden verschiedene Farben eingesetzt, um unterschiedliche Zustände anzuzeigen. Die Texteingabefelder bieten zusätzliche Komfortfunktionen. Felder mit Aufzählungswerten zeigen eine Dropdown-Liste mit allen möglichen Werten an. Die numerischen Felder speichern eine Historie vom Benutzer eingegebener Werte und stellen diese über eine Dropdown-Liste zur schnellen Auswahl häufig genutzter Werte bereit.

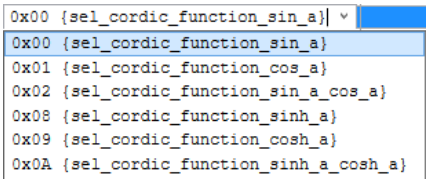


Abbildung 14 Auswahl der gewünschten Funktion bei einem Cordic-Basisblocks

Der DFA-Hardware-Beschleuniger verwendet ein internes Netzwerk, die Interconnect-Matrix, um den Ausgang eines Blocks mit dem Eingang eines anderen Blocks zu verbinden und so einen Datenflussgraphen zu erzeugen. Zur Konfiguration verfügt der Hardwarebeschleuniger dazu über Register mit einer besonderen Funktion, die Zieladress-Register "dest_addr_<n>", mit "<n> als Ganzzahl. Diese Register definieren die verwendeten Verbindungen innerhalb der Interconnect-Matrix. Der Beschleuniger erlaubt nur eine Teilmenge aller theoretisch möglichen Verbindungen. Das bedeutet auch, dass die „dest_addr“-Register nur eine begrenzte Menge von Zieladressen enthalten dürfen. Das Modell berechnet auf Basis der Interconnect-Matrix die gültigen Verbindungen und ermittelt für jedes „dest_addr“-Register die zulässigen Zielverbindungen als Enumerationswerte. So kann der Benutzer direkt über eine textuelle Darstellung – anstelle eines numerischen Werts – ein Ziel-Basismodul und Register als Ausgangsverbindung auswählen. Der resultierende Datenflussgraph kann in einer hierarchischen Baumansicht dargestellt werden, wobei die eingehende Verbindung eines Basisblocks den verbundenen Ausgangsblock als Unterelement anzeigt.

Die Wurzelknoten des Baums sind die Blöcke, die keine weiteren ausgehenden Verbindungen besitzen. Der größte Vorteil der Hierarchischen Konfigurations-Ansicht besteht in der Kombination der Blockverbindungen mit der Konfiguration der einzelnen Blöcke innerhalb einer Darstellung. Das bedeutet, dass sowohl strukturelle Informationen als auch Blockdetails gemeinsam angezeigt werden.

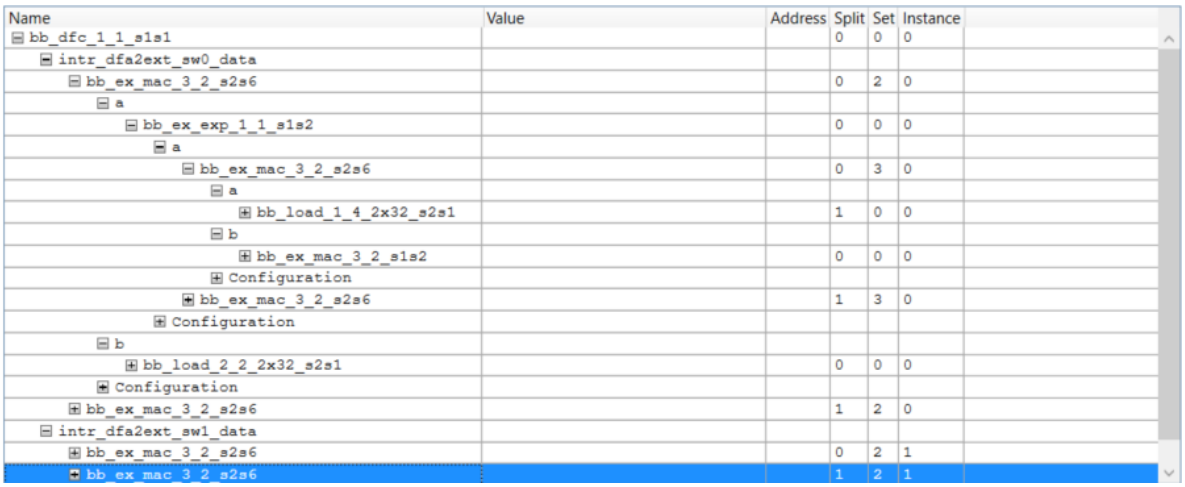


Abbildung 15 Hierarchische Konfigurations-Ansicht mit Verbindung zwischen den Basisblöcken

Die aktuelle Konfiguration des Beschleunigers kann in eine für den Menschen lesbare Konfigurationsdatei im XML-Format exportiert und zu einem späteren Zeitpunkt wieder importiert werden. Dabei wird eine XML-Datei mit folgendem Inhalt erzeugt, der auch in einer Schemadatei "DataFlowArchitectureConfiguration.xsd" beschrieben ist: ein Name für den Algorithmus, eine Identifikationszeichenfolge, die die aktuell verwendete DFA-Architektur beschreibt, eine Sammlung aller Basisblöcke, die über die Interconnect-Matrix verbunden sind, die Kennungen und Registerinhalte der ausgewählten Blöcke sowie eine Cluster-Konfiguration, die beschreibt, welche Register gleichzeitig geschrieben werden können

Nicht jedes Register wird in die Konfigurationsdatei geschrieben. Stattdessen nutzt die Komponente die Interconnect-Informationen, um Basisblöcke zu ermitteln, die eine gültige Konfiguration aufweisen und Teil eines Netzwerks sind. Auf diese Weise wird nur die Konfiguration derjenigen Blöcke exportiert, die tatsächlich am aktuellen Algorithmus beteiligt sind. Die resultierende XML-Datei kann vom Benutzer eingesehen und bearbeitet und anschließend auch wieder eingelesen werden.

Die entwickelte Grafische Darstellung ist die Ansicht mit dem höchsten Abstraktionsgrad. Sie entspricht dem Datenflussgraphen wie er durch die vorgelagerten Teile der Werkzeugkette, die von den anderen Projektpartnern erstellt wurden, zur Beschreibung der Algorithmus-Konfiguration verwendet wird. Der Graph besteht aus Knoten, die die Basisblöcke des DFA-Hardware-Beschleunigers darstellen, sowie aus Kanten zwischen den Blöcken, die die Verbindungen vom Ausgang eines Blocks zum Eingang des nächsten Blocks repräsentieren.

Die Blöcke sind mit einer kompakten Darstellung ihrer Konfiguration beschriftet; die Verbindungspunkte der Kanten an den Blöcken sind mit den Namen der Ein- und Ausgänge versehen. Die Funktion eines Blocks wird entweder durch einen gebräuchlichen Namen wie "DFC" oder durch eine mathematisch/algorithmische Darstellung der Konfiguration beschrieben. Für die Visualisierung wird ein Layout-Algorithmus verwendet, der die Knoten und Kanten mit minimalen Überlappungen anordnet. Die Blöcke können vom Benutzer manuell verschoben werden, um das Layout individuell an die eigenen Anforderungen anzupassen.

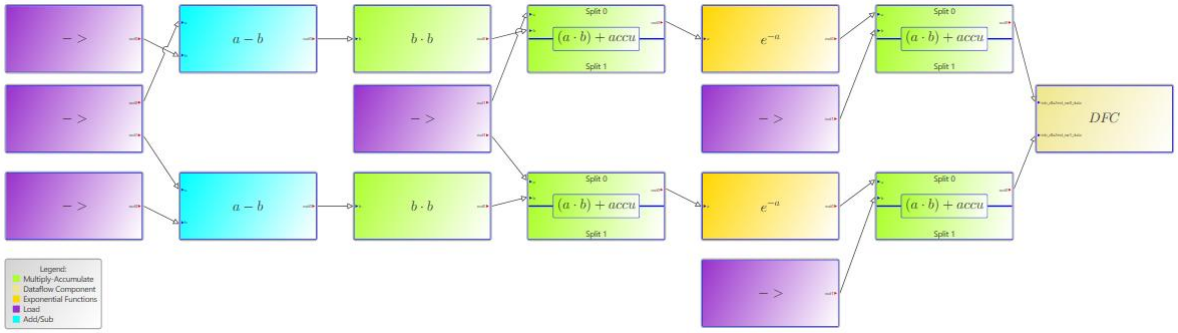


Abbildung 16 Grafische Darstellung eines konfigurierten DFA-Algorithmus

Jedem Basisblocktyp ist eine eigene Farbe zugewiesen. Die Zuordnung der Farben zu den Blocktypen wird in einer Legende erläutert. Dies erleichtert es dem Benutzer, relevante Blöcke schneller zu erkennen. Die Konfiguration von Basisblöcken mit arithmetischen Operationen (z. B. MAC, AddSub, Cordic, Exp) ist in der Regel über mehrere Register verteilt. Diese Konfiguration umfasst die Auswahl der Eingangsquellen (Ports und/oder Akkumulator), die mathematische Operation sowie die Loop-Ebene. Der Debugger wertet den Registerinhalt aus und berechnet daraus einen mathematischen Ausdruck, der als Beschriftung des Blocks angezeigt wird.

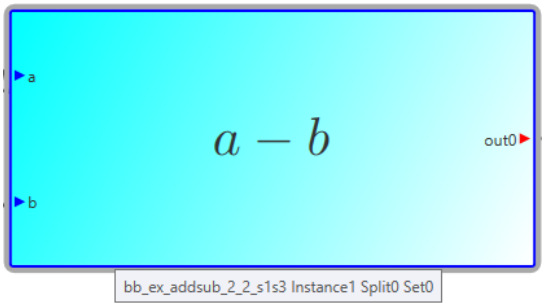


Abbildung 17 Darstellung eines Add/Sub-Blocks mit der aktuellen Konfiguration als Subtraktion und mit dem Block-Identifier im Tooltip.

Der Beschleuniger unterstützt die Kombination von Blöcken mit unterschiedlichen "Splits" zu einem kombinierten Block, um Berechnungen mit 64-Bit-Datentypen durchzuführen – im Gegensatz zu den standardmäßigen 32-Bit-Datentypen im Normalbetrieb. Der Debugger unterstützt diese Funktion, indem er die Blöcke zu einem Block zusammenfasst und diesen in zwei Bereiche unterteilt darstellt, um die "Combined"-Eigenschaft zu visualisieren. Die einzelnen Teile sind mit dem jeweiligen Split-Bezeichner beschriftet, um eine weitere visuelle Unterscheidung der beteiligten Blöcke zu ermöglichen.

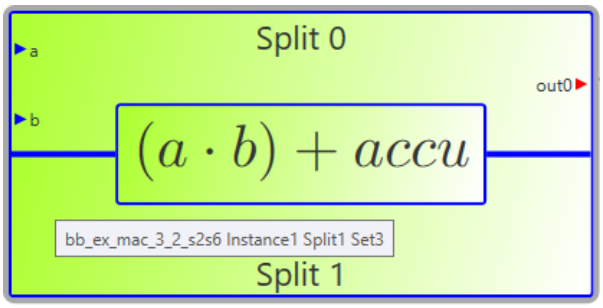


Abbildung 18 Darstellung eines Multiply/Add-Blocks im "Combined"-Mode mit Akkumulation des Ergebnisses

Alle Blöcke verfügen über mehrere Eingangs- und Ausgangsports. Da die meisten davon ungenutzt sind, werden nur die tatsächlich verwendeten Ports vom Algorithmus zur Anzeige ausgewählt. Alle erkannten Verbindungen werden überprüft und als verwendet registriert. Die Blöcke werden mit einer Liste an der linken Seite (Eingänge) und der rechten Seite (Ausgänge) dargestellt. Die Verbindungen zwischen den Ports werden so geroutet, dass möglichst wenige Kreuzungen entstehen.

Basierend auf dem analysierten Datenflussgraphen wurde untersucht ob die Generation von GALEC-Code möglich und sinnvoll ist. Zuerst wurde nach einem Algorithmus zum Ermitteln der Abarbeitungsreihenfolge der einzelnen DFA-Blöcke gesucht und dieser prototypisch implementiert. Anschließend wurde GALEC-Code in Ausführungsreihenfolge für die Eingangsverbindungen und für das konfigurierte Verhalten jedes Basisblocks generiert. Zur Datenübergabe wurden in GALEC Vektoren angewendet. Die 'Loop-Level'-Fähigkeit der DFA-Hardware wurde durch 'for'-Schleifen modelliert.

```
algorithm
  [...]
  // bb_ex_mac_3_2_s1s2_0_0_0 behavioural model
  for loop0 in 0:2 loop
    for loop1 in 0:1 loop
      for loop2 in 0:1 loop
        'bb_ex_mac_3_2_s1s2_0_0_0.mult' := 'bb_load_1_4_2x32_s2s1_0_0_0.out'[loop0,loop1,loop2] *
'bb_load_2_2_2x32_s2s1_0_0_0.out'[loop0,loop1,loop2];
        'bb_ex_mac_3_2_s1s2_0_0_0.addsub_result' := 'bb_ex_mac_3_2_s1s2_0_0_0.mult' + 'bb_ex_mac_3_2_s1s2_0_0_0.accu';
        'bb_ex_mac_3_2_s1s2_0_0_0.accu' := bb_ex_mac_3_2_s1s2_0_0_0.addsub_result
      end for;
      'bb_ex_mac_3_2_s1s2_0_0_0.out'[loop0,loop1] := 'bb_ex_mac_3_2_s1s2_0_0_0.addsub_result';
      'bb_ex_mac_3_2_s1s2_0_0_0.accu' := 0;
    end for;
  end for;

  // bb_store_2_1_2x32_s2s1_0_0_0 behavioural model
  for loop0 in 0:1 loop
    'bb_store_2_1_2x32_s2s1_0_0_0.offset0' := 0 + (1*loop0);
    for loop1 in 0:1 loop
      'bb_store_2_1_2x32_s2s1_0_0_0.offset1' := 0 + (2*loop1);
      'bb_store_2_1_2x32_s2s1_0_0_0.addr' := 6144 + ('bb_store_2_1_2x32_s2s1_0_0_0.offset0' + ('bb_store_2_1_2x32_s2s1_0_0_0.offset1'));
      dfa_ram['bb_store_2_1_2x32_s2s1_0_0_0.addr'] := 'bb_ex_mac_3_2_s1s2_0_0_0.out'[loop0,loop1];
    end for;
  end for;
```

Abbildung 19 Auszug aus generiertem GALEC-Code für eine Matrix-Multiplikation

Der erzeugte Code wurde durch einen vom Projektpartner OFFIS bereitgestellten GALEC-Parser auf syntaktische und manuell auf algorithmische Korrektheit geprüft. Nach Nachweis der Machbarkeit wurden die Arbeiten an der Codegenerierung nicht weiter vertieft, da die anderen gefundenen Darstellungsarten sich als für die Anwender weitaus sinnvoller und nützlicher erwiesen haben.

Die Visualisierung von Daten, die während der Ausführung auf dem Hardware-Beschleuniger aufgezeichnet wurden, ermöglicht tiefere Einblicke in den tatsächlichen Ablauf. Die Möglichkeiten zur Visualisierung hängen davon ab, welche Trace-Daten aufgezeichnet werden konnten – je nach Systemintegration, Hardware-Beschleuniger und den verwendeten Werkzeugen. Im Rahmen dieses Projekts wurden verschiedene Analyse- und Visualisierungsmethoden für Trace-Daten untersucht.

Die Tabellen-Übersicht bietet einen Überblick über eine vollständige Aufzeichnung der Ausführungsdaten. Sie enthält alle in der Aufzeichnung enthaltenen Informationen. Die konkrete Darstellung richtet sich danach, welche Daten verfügbar sind. Diese Visualisierung dient der manuellen Analyse aufgezeichneter Daten. Sie stellt Filter-, Sortier- und Gruppierungsfunktionen zur Verfügung, um die manuelle Analyse zu erleichtern. Die Datenfelder eines Eintrags werden in vorkonfigurierten Spalten angezeigt. Da nicht jedes Feld in allen Eintragstypen vorkommt, bleiben manche Zellen leer. Je nach Feldtyp erfolgt die Darstellung als Zeichenkette, Dezimal- oder Hexadezimalzahl. Der Eintragstyp sowie Block- oder Registerbezeichner werden als benutzerfreundliche Namen dargestellt, Speicheradressen in hexadezimaler Notation, alle übrigen Daten als Zahl.

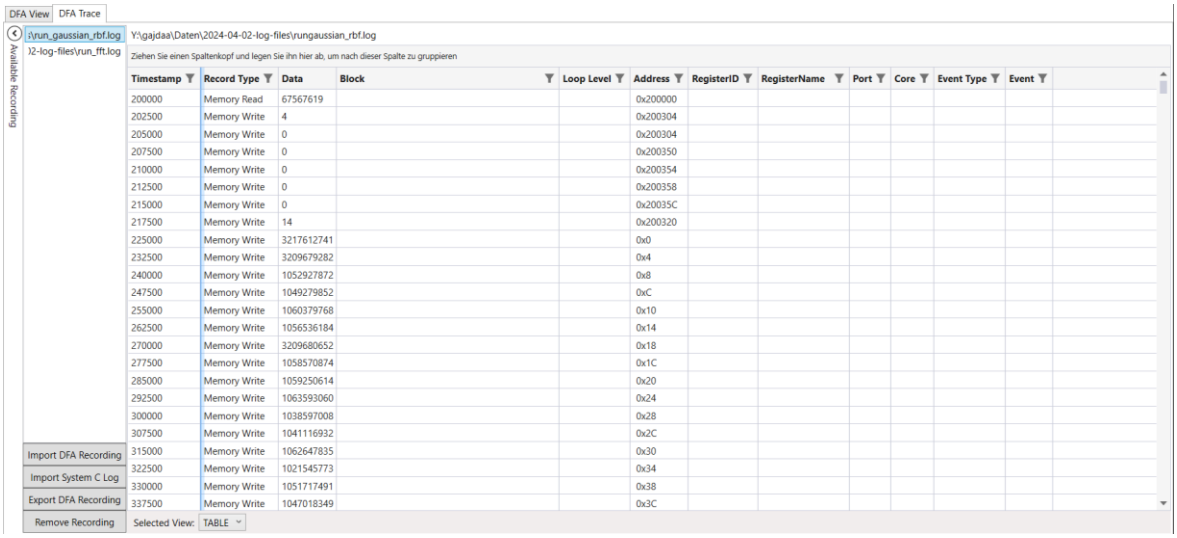


Abbildung 20 Tabellen-Übersicht eines DFA-Trace

Die Ansicht bietet Filterfunktionen pro Spalte, erlaubt das Neuanordnen der Spalten und ermöglicht auf- oder absteigendes Sortieren nach Spalteninhalt.

Die Konfigurations-/Ausführungs-Timeline stellt alle Registeränderungen innerhalb der DFA in einem zeitlichen Zusammenhang dar. Es erfolgt keine getrennte Darstellung, da die Konfiguration möglicherweise noch gar nicht abgeschlossen ist, wenn dies Ausführung der DFA beginnt.

In der prototypischen Software zeigt die Ansicht "Configuration/Execution Timeline" im oberen Teil die grafische Darstellung des Zustands zum aktuell gewählten Zeitpunkt auf der Zeitleiste. Im unteren Teil wird eine Tabelle mit den Records angezeigt, die zu diesem Zeitpunkt aufgezeichnet wurden. Zur Navigation auf der Zeitleiste stehen folgende Möglichkeiten zur Verfügung: Mit den Schaltflächen links und rechts der Zeitleiste kann zum vorherigen oder nächsten Zeitpunkt gesprungen werden. Der Zeitleistencursor kann per Drag-and-Drop an eine gewünschte Stelle im Zeitverlauf verschoben werden. Zusätzlich kann der Cursor durch einen Klick links oder rechts neben ihn auf den nächsten Zeitpunkt gesetzt werden, an dem sich die Konfiguration der DFA geändert hat.

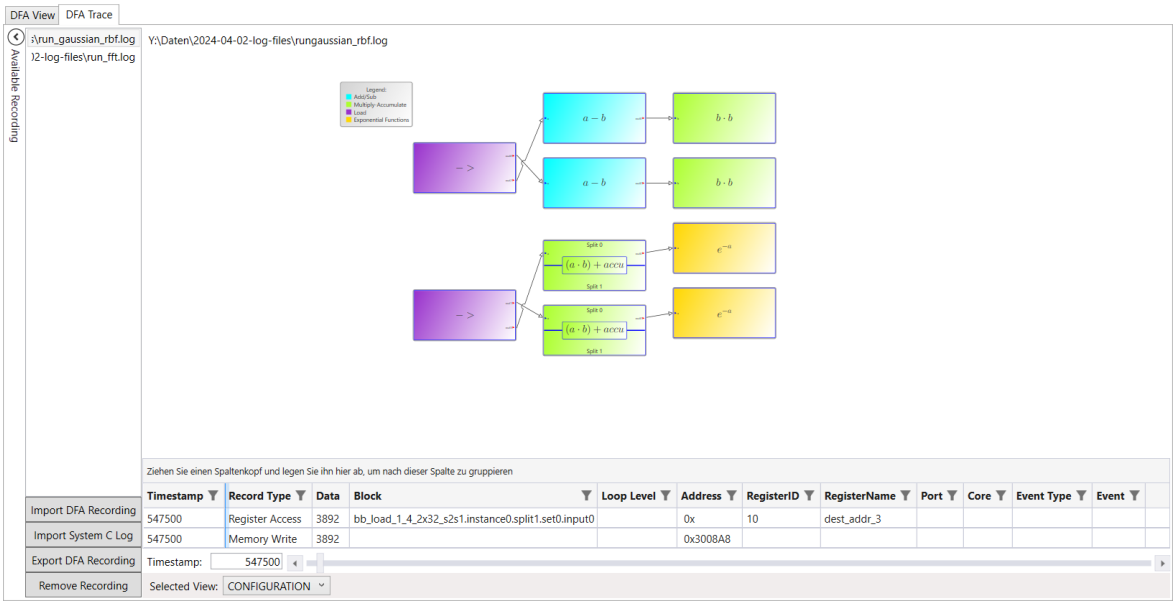


Abbildung 21 Ansicht "Configuration/Execution Timeline" in der prototypischen Software mit unvollständiger DFA-Konfiguration

Der Datenflussgraph wird auf Basis aller Schreibzugriffe auf Konfigurationsregister aktualisiert. Daraus ergibt sich die aktuelle Konfiguration, aus der der Datenflussgraph erzeugt wird, der dann angezeigt wird. Abbildung xx zeigt ein Beispiel für dieses Verhalten: Derselbe Eintrag wie in Abbildung yy wird angezeigt, jedoch zu einem Zeitpunkt, an dem die DFA noch nicht vollständig konfiguriert ist.

Abbildung 22 Ansicht "Configuration/Execution Timeline" in der prototypischen Software nach vollständig erfolgter DFA-Konfiguration

Der Datenflussgraph basiert auf denselben Mechanismen wie die Registerkonfigurationsdarstellung des DFA. Da die DFA-Einträge auch übertragene Daten zwischen Basisblöcken enthalten, sind zukünftige Erweiterungen der Darstellung denkbar, z. B.: Annotationen der übertragenen Daten, Darstellung eines separaten Datenflussgraphs je Datenpaket, Hervorhebung einzelner Datenpfade u. v. m. Untersuchungen in diesem Projekt haben gezeigt, dass verschiedene Arten der Darstellung der Eintragsdaten sinnvoll und umsetzbar sind.

1.3 AP 4: Use-Case ICE – Smart Monitoring mit KI-basierten, virtuellen Sensoren AP 5 Use-Case Fuel Cell – Modellbasierte Regelung des Brennstoffzellenstapels

Optimierung der Methoden und Werkzeuge

Das Anwenderfeedback zur Optimierung der Methoden und Werkzeuge kam im Wesentlichen von den Projektpartnern Bosch, itemis AG und OFFIS. Mit Bosch gab es intensiven Austausch direkt mit den DFA-Entwicklern zur optimalen Darstellung von DFA-Konfigurations- und Trace-Daten sowie zur Integration des System-C – Modells. Das XML-Format zur Beschreibung von DFA-Konfigurationen wurde insbesondere mit OFFIS in mehreren Iterationen weiterentwickelt. Ebenso wurde ein GALEC-Syntax-Parser bereitgestellt. Itemis lieferte im Wesentlichen die Beiträge für das Artifact-Trace und lieferte Feedback zu den Konzepten der Toolkommunikation innerhalb der Werkzeugkette.

1.4 AP 6: Projektmanagement

Die Koordination und Durchführung eines Projektes dieses Umfangs benötigt entsprechendes Management. Die wichtigsten Aufgaben im Zusammenhang mit diesem Arbeitspaket beinhalteten für die PLS GmbH:

- Die Planung, Organisation und Koordination von regelmäßigem Austausch durch Telefonkonferenzen und die inhaltliche Vorbereitung von Präsentationen zu den Projekttreffen im Rahmen des AP3.
- Die Organisation und Bereitstellung von technischer Infrastruktur. Dazu gehören Verwaltung von Austauschlaufwerken und Versionskontrollsystemen.
- Koordination und Ausführung der Reviews, Sicherstellung und Verteilung von Dokumentation und notwendigen Informationen. Planung und Durchführung der kostenneutralen Verlängerung.
- Controlling und Risikomanagement.

Die erfolgreiche Durchführung und Abschluss des Projekts wurde durch eine Präsentation an den Projektträger am 12.12.2024 in Renningen demonstriert.

2 Wichtigste Positionen des zahlenmäßigen Nachweises

Die Ausgaben- und Kostenplanung wurde im Wesentlichen eingehalten. Bei den FE-Fremdleistungen, GWT-TUD GmbH entsprachen die Ausgaben der Planung. Bei den eigenen Personalkosten sind geringfügig weniger angefallen. Für Projekt-Präsenztreffen wurde weniger benötigt als veranschlagt, da die PLS GmbH als kleinster Projektpartner kein Präsenztreffen an seinem Standort organisiert hat.

3 Notwendigkeit und Angemessenheit der geleisteten Arbeit

Die Arbeiten folgten dem Projektantrag. Bei der Zeitplanung waren ein großer Teil der anderen Projektpartner von Kapazitätsengpässen und der Nichtverfügbarkeit eines Brennstoffzellen-Stapels betroffen. Um die Ziele des Vorhabens, insbesondere bezüglich Verwertung und wissenschaftlichen Anschluss, dennoch zu erreichen hatten die Projekt-partner eine kostenneutrale Verlängerung als Lösung identifiziert. Diesem Vorgehen hat sich die PLS GmbH angeschlossen, da die Bearbeitung der von den anderen Projektpartnern abhängigen Arbeitspakete sonst nur bedingt oder ggf. mit erhöhten Kosten möglich gewesen wäre. Die nach Beantragung gewährte Verlängerung des Vorhabens um 6 Monate ermöglichte die Ziele der Förderung vollumfänglich umzusetzen.

Der spezielle Anteil der PLS GmbH lag in der Erforschung und prototypischen Entwicklung von Werkzeugen für die Testung und Validierung von Algorithmen für die neuartige KI-Hardware (Datenflussprozessor, DFA). Diese erlauben eine Prä- und Post-Analyse des Systems und berücksichtigen die massive Parallelität der dynamisch konfigurierbaren Hardware. Die entwickelten Methoden für Debugging und Trace bieten neue Möglichkeiten für die Visualisierung, Zustandsbeobachtung und -veränderung des eingesetzten Datenflussprozessors. Sind sie damit allein als auch besonders im Verbund der entstandenen Werkzeugkette ein wichtiger Baustein für den Einsatz von KI-Methoden in tief eingebetteten Systemen im Allgemeinen und der entwickelten DFA-Hardware des Projektpartners Bosch im Besonderen, da nur die Entwicklung von Algorithmen deren Leistungsfähigkeit voll entfaltet.

4 Voraussichtlicher Nutzen, insbesondere der Verwertbarkeit des Ergebnisses im Sinne des fortgeschriebenen Verwertungsplans

Mit der Anwendung von SoC, welche den untersuchten Datenflussprozessor integriert haben, ist durch Tier-1 und OEM's ab 2026 zu rechnen. Ab diesem Zeitpunkt ist auch mit verstärkter Entwicklung von Algorithmen und entsprechendem Bedarf an Werkzeugen zu rechnen. Durch den durch das Vorhaben erreichten Forschungsvorlauf kann die PLS GmbH ab diesem Zeitpunkt erste einsatzfähige Produkte für Debugging und Trace bereitstellen. Darüber hinaus sind gute Voraussetzungen geschaffen um auch leistungsfähigere Nachfolgeversionen des untersuchten Datenflussprozessors bzw. andere Hardware-Beschleuniger mit ähnlichen Mechanismen zu unterstützen. Durch die Universalität des Ansatzes des unterstützten Datenflussprozessors zur schnellen, energieeffizienten und kostengünstigen Berechnung von nahezu beliebigen mathematischen Algorithmen ist der Einsatz nicht auf den Automobil-Bereich beschränkt. Überall, wo in tief eingebetteter Hardware insbesondere KI-Methoden zum Einsatz kommen, ist eine Anwendung denkbar und damit auch an einem entsprechenden Bedarf an Werkzeugen zu rechnen.

Die PLS GmbH wird den im Vorhaben erreichten Technology Readiness Level TRL 6 (Prototyp in vereinfachter Einsatzumgebung) bis 2026 zu TRL 8 (Nachweis der Funktionsfähigkeit im Einsatzbereich) weiterentwickeln um einsatzfähige Werkzeuge bereitstellen zu können. Dafür ist geplant, die Zusammenarbeit mit den Projektpartnern Bosch, itemis AG und OFFIS sowie der TU Dresden fortzusetzen. Ziel ist es die entwickelte Technologie im Markt zu etablieren (TRL9) und darauf aufbauend Debug- und Testmethoden für KI-Methoden auch auf anderer tief eingebetteter Hardware zu entwickeln.

Die Gesamtheit der Entwicklungswerkzeuge des DFA-Hardwarebeschleunigers werden es ermöglichen, Anwendungen für die Hardwareplattform kostengünstiger zu entwickeln, wodurch sich deren mögliche Einsatzzwecke vervielfältigen werden. Es wird erwartet, dass sich daraus dann auch weitere Entwicklungen für unterstützende Tools ergeben und insgesamt eine positive Dynamik rund um den DFA-Hardwarebeschleuniger geben wird.

5 Während der Durchführung des Vorhabens dem ZE bekannt gewordener Fortschritt auf dem Gebiet des Vorhabens bei anderen Stellen

Das Entwicklungstempo und die Menge der Forschungsarbeiten im Bereich KI-Anwendung waren im Vorhabens-Zeitraum weiter zunehmend. Sehr wenige Publikationen beschäftigten sich aber mit der Verwendung von KI-Methoden auf echtzeitfähiger Hardware in tief eingebetteten Systemen. Wenn überhaupt in diesem Bereich eine Anwendung erfolgt, dann basierend auf einer Client-Server-Kommunikation mit Internetanfragen auf ein extern laufendes KI-Modell.

6 Erfolgte und geplante Veröffentlichungen des Ergebnisses

Die PLS GmbH sieht gute Möglichkeiten die Ergebnisse dieses Vorhabens auf Anwenderkonferenzen zu präsentieren:

- embedded world Conference 2026
- Embedded Software Engineering Kongress 2026

7 Literaturverzeichnis

Bannow, N. (13. November 2024). DFA-IP - Debug & Trace Guide.

EMPHYSIS. (2021, February 22). *4.2. GALEC: The Programming Language for Algorithm Code Containers' Source Code*, 1.0.0-Alpha.4 (Draft). Retrieved from 4.2. GALEC: The Programming Language for Algorithm Code Containers' Source Code: https://www.efmi-standard.org/media/home/eFMI-Standard-1.0.0-Alpha-4.html#_galec_the_programming_language_for_algorithm_code_containers_source_code

EMPHYSIS. (2021, February 22). *Functional Mock-Up Interface for embedded systems (eFMI)*, 1.0.0-Alpha.4 (Draft). Retrieved from Functional Mock-Up Interface for embedded systems (eFMI): https://www.efmi-standard.org/media/home/eFMI-Standard-1.0.0-Alpha-4.html

Robert Bosch GmbH. (2022, 01 17). DFA - User Guide. (2.7).

8 Abbildungsverzeichnis

Abbildung 1 Prototypische Toolbar für den Core00 eines die DFA integrierenden SoC und die DFA. Die Elemente zur Ausführungssteuerung auf der linken Seite sind jeweils dem aktiven Core zugeordnet, das im Moment gerade die DFA ist. 8

Abbildung 2 Auszug aus einer XML-basierten DFA-Konfiguration 9
Abbildung 3 Auszug aus einer Log-Datei des System-C - Modells 12
Abbildung 4 Rekonstruierter Datenabhängigkeitsgraph aus dem Software-Log 13
Abbildung 5 Bitposition und -breite der Identifikator-Typen 15

Abbildung 6 Grafische Darstellung der Konfiguration und der Zeitachse der Datenereignisse, extrahiert aus den Konfigurationsdaten 18

Abbildung 7 Notification-Schema 20
Abbildung 8 0 Request-Response-Schema 21
Abbildung 9 0 Subscription- (Abbonnement-) Schema 21
Abbildung 10 Modellelementauswahl-Schema 22
Abbildung 11 DFA-Zustandsantwort-Schema 23
Abbildung 12 Algorithmus-Result-Schema 23
Abbildung 13 Konfigurationsübersicht zu Basisblöcken mit Splits sowie Register und Registerfelder 26
Abbildung 14 Auswahl der gewünschten Funktion bei einem Cordic-Basisblocks 27
Abbildung 15 Hierarchische Konfigurations-Ansicht mit Verbindung zwischen den Basisblöcken 28
Abbildung 16 Grafische Darstellung eines konfigurierten DFA-Algorithmus 29

Abbildung 17 Darstellung eines Add/Sub-Blocks mit der aktuellen Konfiguration als Subtraktion und mit dem Block-Identifier im Tooltip. 29

Abbildung 18 Darstellung eines Multiply/Add-Blocks im "Combined"-Mode mit Akkumulation des Ergebnisses 29

Abbildung 19 Auszug aus generiertem GALEC-Code für eine Matrix-Multiplikation 30
Abbildung 20 Tabellen-Übersicht eines DFA-Trace 31

Abbildung 21 Ansicht "Configuration/Execution Timeline" in der prototypischen Software mit unvollständiger DFA-Konfiguration 32

Abbildung 22 Ansicht "Configuration/Execution Timeline" in der prototypischen Software nach vollständig erfolgter DFA-Konfiguration 32

Berichtsblatt

1. ISBN oder ISSN	2. Berichtsart (Schlussbericht oder Veröffentlichung)
Nicht zutreffend	Schlussbericht

3. Titel
KI Embedded KI-Grundlagenentwicklung mit Leitanwwendungen Virtuelle Sensorik und Brennstoffzellenregelung Schlussbericht der PLS Programmierbare Logik & Systeme GmbH

4. Autor(en) [Name(n), Vorname(n)]
Rießland, Heiko
Gajda, Andreas
Fräulin Fridolin

5. Abschlussdatum des Vorhabens
31.12.2024

6. Veröffentlichungsdatum
01.12.2026

7. Form der Publikation
Kongress

8. Durchführende Institution(en) (Name, Adresse)
PLS Programmierbare Logik & Systeme GmbH
Straße der Freundschaft 92
02991 Lauta

9. Ber.-Nr. Durchführende Institution

10. Förderkennzeichen
19I21043F

11. Seitenzahl
36

12. Fördernde Institution (Name, Adresse)
BMWK
10100 Berlin

13. Literaturangaben
4

14. Tabellen
1

15. Abbildungen
22

16. DOI (Digital Object Identifier)
Nicht zutreffend

17. Vorgelegt bei (Titel, Ort, Datum)
Nicht zutreffend

18. Kurzfassung
Aktuelle Werkzeuge zur Fehlersuche (Debugger) in herkömmlichen Rechnerarchitekturen gehen von einer überwiegend sequentiellen Abarbeitung von definierten Maschinenbefehlen aus, die auch mehrstufig aus einer Modell- oder Programmiersprache generiert werden. Zusätzlich werden symbolische Informationen erzeugt, welche die Rücktransformation auf die Programmiersprache zulassen. Darüber hinaus gewinnen Debugger mittels Trace-Hardware Informationen über den tatsächlich erfolgten Kontroll- und Datenfluss.
Die Steuerung, Visualisierung, Fehlersuche und Testung neuartiger KI-Hardware (Datenfluss-Architektur, DFA) und den darauf ausgeführten massiv parallelen Algorithmen ist mit den auf sequentieller Abarbeitung basierenden Ansätzen aber nicht umsetzbar. Deshalb wurden neue Ansätze für das Debugging, die Laufzeitbeobachtung und den Test solcher neuartigen Beschleuniger-Hardware erforscht und entwickelt.
Die verwendeten Methoden umfassen Konzeption und prototypische Implementierungen von Schnittstellen, Untersuchungen an realen Hardware-Systemen und an einem System-C - Simulator, sowie Entwurf und prototypische Implementierung von Algorithmen zur Filterung und analytischen und visuellen Aufbereitung von Laufzeitinformationen.
Ergebnis ist ein erster Prototyp, welcher die Kontrolle und Beobachtung der DFA auf verschiedenen Abstraktions-Leveln vom Registerniveau bis zur grafischen Konfiguration sowie die Auswertung von Laufzeitdaten (Trace) ermöglicht. Offene Datenformate und Schnittstellen ermöglichen den einfachen Austausch von Informationen mit anderen Werkzeugen.
Mit dem Ergebnis wurden die Voraussetzungen geschaffen um einsatzfähige Produkte für das Debugging, Tracing, die Visualisierung und den Test der DFA und ähnlicher Hardware bereitstellen zu können. Durch die Universalität des DFA-Ansatzes zur Berechnung von nahezu beliebigen mathematischen Algorithmen ist der Einsatz überall denkbar, wo in tief eingebetteter Hardware insbesondere KI-Methoden zum Einsatz kommen.

19. Schlagwörter
Künstliche Intelligenz, Tief eingebettete Systeme, Debugging, Trace, Test, Datenfluss-Architektur, DFA

20. Verlag	21. Preis
Nicht zutreffend	Nicht zutreffend

Nicht änderbare Endfassung mit der Kennung 2719427-15

Document control sheet

1. ISBN or ISSN Not applicable	**2. type of document (e.g. report, publication)** Veröffentlichung (Publikation)
3. title KI Embedded KI-Grundlagenentwicklung mit Leitanwwendungen Virtuelle Sensorik und Brennstoffzellenregelung Schlussbericht der PLS Programmierbare Logik & Systeme GmbH	
4. author(s) (family name, first name(s)) Riessland, Heiko Gajda, Andreas Fräulin Fridolin	**5. end of project** 31.12.2024
	6. publication date 01.12.2026
	7. form of publication Kongress
8. performing organization(s) name, address PLS Programmierbare Logik & Systeme GmbH Straße der Freundschaft 92 DE-02991 Lauta Germany	**9. orginators report no.**
	10. reference no. 19I21043F
	11. no. of pages 36
12. sponsering agency (name, address) BMWK DE-10100 Berlin Germany	**13. no. of references** 4
	14. no. of tables 1
	15. no. of figures 22
16. DOI (Digital Object Identifier) Not applicable	
17. presented at (title, place, date) Not applicable	

18. abstract

Current tools for debugging (debuggers) in conventional computer architectures are based on the predominantly sequential execution of defined machine instructions, which may be generated through multiple stages from a model or programming language. Additionally, symbolic information is produced to allow reverse mapping back to the original programming language. Debuggers also use trace hardware to gather information on the actual control and data flow that occurred during execution. However, for novel AI hardware — specifically dataflow architectures (DFAs) — and the massively parallel algorithms executed on them, these sequentially-based approaches are no longer viable for control, visualization, debugging, or testing.

For this reason, new approaches have been researched and developed for debugging, runtime observation, and testing of such advanced accelerator hardware.

The methods used include the design and prototypical implementation of interfaces, investigations using real hardware systems and a SystemC simulator, as well as the development and prototypical implementation of algorithms for filtering and the analytical and visual processing of runtime information.

The result is an initial prototype that enables control and observation of the DFA across various levels of abstraction — from the register level to graphical configuration — as well as the evaluation of runtime data (traces). Open data formats and interfaces allow for easy exchange of information with other tools.

With this result, the foundation has been laid for the development of deployable products for debugging, tracing, visualization, and testing of DFAs and similar hardware. Thanks to the universality of the DFA approach, which enables the execution of virtually any mathematical algorithm, it is applicable wherever deeply embedded hardware — especially involving AI methods — is used.

19. keywords

Artifical Intelligence, Deeply embedded systems, Debugging, Trace, Test, Data Flow Architecture, DFA

20. publisher Not applicable	**21. price** Not applicable