GENODE-FX: EINE FPGA-BASIERTE GRAFISCHE BENUTZERSCHNITTSTELLE

Norman Feske <norman.feske@genode-labs.com>
Genode Labs, Feske & Helmuth GbR, Dammweg 2, 01097 Dresden

Genode-FX ist eine Kombination aus Hardware-IP und Software, die es erlaubt, komplexe grafische Benutzeroberflächen mittels low-cost FPGAs als System-on-Chip-Lösungen zu realisieren. Für die designierten Anwendungsbereiche Steuerungs- und Messtechnik besteht besonderes Augenmerk auf der zeitlichen Vorhersagbarkeit grafischer Ausgaben und einer garantierten Reaktionszeit auf Benutzereingaben. Der Artikel bietet einen Überblick über das zugrunde gelegte Hardware-Design und unsere Software-Architektur, die solche zeitlichen Garantien ermöglicht.

1. Einleitung

FPGAs (Field-Programmable Gate Arrays) kommen zunehmend in eingebetteten Steuergeräten und high-end Messgeräten zum Einsatz. Im Gegensatz zu konventionell verwendeten digitalen Signalprozessoren und Mikrocontrollern ermöglicht ein FPGA die Umsetzung einer kompletten System-on-Chip Lösung, die auf eine spezielle Anwendung optimiert ist. Somit können Signalverarbeitungsaufgaben, Peripherie-Controller, Glue-Logic sowie komplette CPUs in Form von Soft-Cores auf einem Chip konsolidiert werden. Da ein FPGA beliebig rekonfiguriert werden kann, ohne dass physische Hardware verändert werden muss, wird es immer einfacher und kostengünstiger möglich, die Hardware-Funktionen eines FPGAs an den speziellen Bedürfnissen eingebetteter Software auszurichten. Durch diesen Hardware-Software-Codesign-Ansatz wird es möglich, immer anspruchsvollere Funktionen wie komplexe Datenanalyse- und Darstellungsfunktionen im online-Betrieb eines eingebetteten System anzubieten.

Einhergehend mit der Zunahme eingebetteter Funktionalität, werden herkömmlich eingesetzte textbasierter LCD-Displays und Schalter zunehmend von Touchscreens abgelöst. Zum Ansteuern solcher Displays existieren diskrete Grafiklösungen in Form von 32-Bit-Mikrocontrollern mit integriertem Display-Controller. Als Alternative zur Integration eines solchen grafikfähigen Chips auf einem embedded Board verfolgen wir mit Genode-FX den Ansatz, die komplette GUI als Teilfunktion eines FPGAs zu realisieren.

Mit diesem Ansatz stellen wir uns zwei Herausforderungen. Anwendungen in der Steuerungs- und Messtechnik stellen häufig Echtzeitanforderungen an eine grafische Benutzerschnittstelle. Insbesondere muss gewährleistet sein, dass die dargestellten Informationen mit geringer zeitlicher Latenz auf dem Bildschirm erscheinen. Da Touchscreens keine taktile Reaktion auf Benutzereingaben bieten, ist schnelles visuelles Feedback der GUI auf Benutzereingaben umso wichtiger. Die zweite Herausforderung besteht in der inherenten Beschränkung von Hardware-Ressourcen, wie FPGA-Slices, Speicher und die Rechenzeit einer Soft-Core-CPU.

Der verbleibende Artikel ist wie folgt aufgebaut. Die Abschnitte 2 und 3 beschreiben die Hardware- bzw. Software-Bestandteile von Genode-FX. Die Echtzeiteigenschaften der Lösung werden in Abschnitt 4 durch zwei experimentelle Messungen näher beleuchtet.

2. Ein System-on-Chip für interaktive Grafik

Die Hardware-Seite von Genode-FX basiert auf dem Xilinx Embedded Development Kit (EDK), welches neben Entwicklungswerkzeugen und Synthesecompiler eine Vielzahl von fertigen IP-Cores zur Verfügung stellt, aus denen der Entwickler ein komplettes eingebettetes System mittels Xilinx FPGAs realisieren kann. Diese IP-Cores umfassen u.a. Controller für eine Vielzahl von Schnittstellen, Timer, Speichercontroller und eine 32-Bit Soft-Core CPU. Zusätzlich ist es leicht möglich, das System durch selbst entwickelte Hardware-Komponenten zu ergänzen. Abbildung 1 gibt einen groben Überblick über ein typisches Genode-FX-System, welches mit dem Xilinx EDK realisiert ist. Es besteht aus folgenden Komponenten:

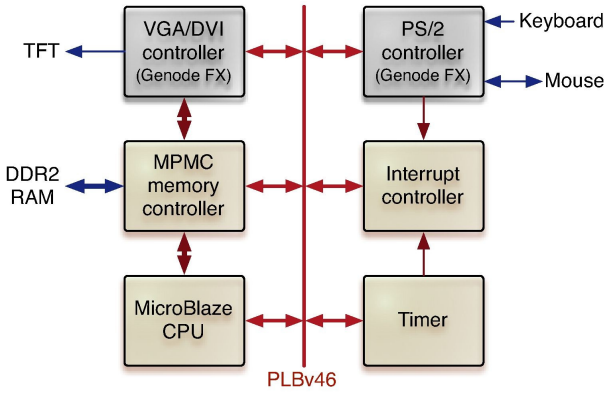

Abbildung 1: Überblick über die Hardware-Komponenten von Genode-FX

MicroBlaze CPU MicroBlaze ist eine 32-Bit RISC Soft-Core CPU, die zum Designzeitpunkt parametrisiert werden kann. So ist es möglich, Cache-Größen, die Pipeline-Tiefe, Fließkommaunterstützung und viele andere Parameter auf die Anwendung zuzuschneiden. Für die Softwareentwicklung steht die GNU-Compiler-Collection zur Verfügung.

Timer Die Timer-Komponente ist über den prozessorlokalen Bus (PLB) mit der MicroBlaze CPU verbunden, bietet einen Takt-Zähler als Zeitbasis und generiert periodisch Interrupts.

PS/2-Controller Dieser speziell für Genode-FX entwickelte Controller macht PS/2 Eingabeereignisse von Maus und Tastatur mittels Hardware-Register der MicroBlaze CPU sichtbar. Für jedes Eingabeereignis wird ein Interrupt generiert, sodass die CPU das empfangene PS/2 Paket verarbeiten kann. Für den Einsatz von Touchscreens kann alternativ zum PS/2-Controller eine RS232-Schnittstelle ge-

nutzt werden.

Interrupt-Controller Der Interrupt-Controller aggregiert alle Interrupt-Quellen im System und gibt auftretende Interrupts an die CPU weiter. In einem minimalen Genode-FX-System fungieren der Timer und der PS/2-Controller als Interrupt-Quellen.

Display-Controller Dieser speziell für Genode-FX entwickelte VGA/DVI-Display-Controller nutzt einen Teil des Hauptspeichers als Frame-Buffer und bietet frei programmierbare Video-Timings. Der Pixel-Takt kann zum Designzeitpunkt gewählt werden, wobei Pixeltakte von mehr als 100MHz möglich sind. Hiermit können hohe Auflösungen mit bis zu 1280x1024 Pixeln in 16-Bit Farbtiefe realisiert werden.

Multi-Port Memory-Controller (MPMC) Der MPMC ist die Schnittstelle zum DDR2-Hauptspeicher des Systems. Durch mehrere Ports ermöglicht er Zugriffe unterschiedlicher Systemkomponenten auf den Hauptspeicher.

Der MPMC ist in der Lage, den Zugriff mehrerer Komponenten auf den Hauptspeicher zu koordinieren. In einem Genode-FX-System nutzen sowohl die MicroBlaze CPU als auch der Display-Controller schnelle Burst-Transfers um auf den Speicher zuzugreifen.

Das Design ist nicht auf eine bestimmte FPGA-Familie beschränkt. Es werden sowohl die low-cost Spartan3-Serie als auch high-end Virtex4 oder Virtex5 FPGAs unterstützt. Auf Virtex-Plattformen mit fest eingebautem PowerPC-Prozessor kann dieser als Alternative zur MicroBlaze CPU genutzt werden. Weiterhin ist die Nutzung von DDR2-Speicher nicht zwingend notwendig. In früheren Versionen kam ein speziell für Genode-FX entwickelter Dual-Port SRAM-Controller zum Einsatz.

3. Software-Stack

Das Herz der Genode-FX-Software hat seinen Ursprung im DOpE Real-Time GUI-Server [1,2]. Dieser Abschnitt beschreibt die Besonderheiten dieses GUI-Software-Stacks, insbesondere im Bezug auf die zeitliche Vorhersagbarkeit der Latenz grafischer Ausgaben und der Eingabebehandlung. Die Software besteht aus einer zentralen GUI-Server-Komponente und einem oder mehreren Anwendungen, die über eine klar definierte Programmierschnittstelle mit dem Server interagieren. Der GUI-Server koordiniert dabei die grafischen Ausgaben, stellt Benutzereingaben an die betreffenden Anwendungen zu und steuert die Hardware an.

Entkopplung der API von der Darstellung
Herkömmliche GUI-Architekturen basieren auf der synchronen Interaktion zwischen Anwendungen und GUI-Server. Nach dem Erhalten einer Anfrage für eine Zeichenoperation, führt der GUI-Server diese (potentiell zeitaufwändige) Operation sofort aus und gibt anschließend die Kontrolle an die Anwendung zurück. Die erzeugte Last hängt somit vom Verhalten aller beteiligten Anwendungen ab. Der Schlüssel, eine zeitliche Vorhersagbarkeit grafischer Ausgaben für alle Anwendungen garantieren zu können, liegt in der Entkopplung der Ausführung zeitaufwändiger Zeichenoperationen von den API-Aufrufen der einzelnen Anwendungen. Zu diesem Zweck hält der DOpE-GUI-Server die komplette GUI-Repräsentation aller Anwendungen in Form einer Hierarchie von GUI-Elementen (Widgets) vor. Die unterstützten Widgets umfassen unter anderem Fenster, Buttons, Grid-Layout, Text, Pixelgrafiken und Eingabefelder. Anwendungen und der Benutzer interagieren stets mit dem GUI-Modell in Form von Transaktionen. Diese Transaktionen sind leichtgewichtige Operationen, die lediglich aus einer atomaren Anpassung von Datenstrukturen bestehen. Wird z. B. ein Fenster bewegt, wird die neue Fensterposition und der Umstand, dass der betroffene Bildbereich nicht mehr aktuell ist, vermerkt. Es findet jedoch keine Zeichenoperation statt. Auf diese Weise können Anwendungen und der Benutzer potentiell große Mengen an Transaktionen vornehmen, ohne unmittelbar Kosten zu verursachen.

Losgelöst von der Interaktion zwischen GUI-Server und seinen Anwendungen, ist ein separater Prozess im GUI-Server stetig bemüht, das GUI-Modell in Pixel zu transformieren. Im Gegensatz zu den leichtgewichtigen Transaktionen der Anwendungen, ist diese Transformation zeitaufwändig, kann jedoch sehr genau konditioniert werden. Durch das lokal vorhandene Wissen über die Widget-Hierarchien aller Anwendungen sind dem GUI-Server sämtliche Grafikoperationen bekannt, die für die Transformation eines beliebigen Bildbereichs notwendig sind. Somit kann bereits vor der Ausführung einer Transformation deren Aufwand abgeschätzt und in die Planung von Zeichenoperationen einbezogen werden. Im GUI-Modell ist stets vermerkt, welche Bildbereiche nicht mehr dem Modell entsprechen und folglich neu transformiert werden müssen. Im ungünstigsten Fall ist dies der gesamte Bildschirm. Dieser Fall ist somit eine Obergrenze des noch zu erledigenden Transformationsaufwands. Für jede Transaktion gilt demnach, dass deren Resultat garantiert binnen der Zeit auf dem Bildschirm sichtbar wird, die der GUI-Server benötigt, um den gesamten Bildschirm zu transformieren. Die Implementierung ist darauf optimiert, jedes Pixel des zu transformierenden Bildbereichs möglichst nur ein einziges Mal zu beschreiben. Hierfür kommen Clipping-Verfahren zum Einsatz, die das lokal verfügbare Wissen über das GUI-Modell nutzen. Hiermit wird erreicht, dass sich der Transformationsaufwand eines beliebigen Bildbereichs in etwa proportional zu dessen Größe verhält. Somit existiert eine maximale zeitliche Latenz, mit der jede Transaktion einer Anwendung oder des Benutzers auf dem Bildschirm sichtbar wird.

Application-Programming-Interface
Die Programmierschnittstelle des GUI-Servers besteht aus einer einfachen Kommando-Sprache. Die folgende Sequenz von Transaktionen erzeugt ein Fenster, bestehend aus zwei Buttons, die horizontal in einem Raster organisiert sind:

```
grid = new Grid()
button_1 = new Button(-text "OK")
button_2 = new Button(-text "Cancel")
g.place(button_1, -row 1 -column 1)
g.place(button_2, -row 1 -column 2)
win = new Window(-content grid)
win.open()
```

Die Schnittstelle abstrahiert dabei von physischen

Parametern, wie Pixel-Positionen, Font-Größen und Farben. Es bleibt dem GUI-Server und seiner Konfiguration überlassen, die gegebene semantische Beschreibung bestmöglich in Pixel der Zielplattform zu transformieren. Jedes Kommando entspricht einer GUI-Transaktion. Typische Anwendungen führen zum Start eine größere Menge von Transaktionen durch. Im laufenden Betrieb bleibt die Anwendung jedoch weitestgehend vom GUI-Server entkoppelt. Nach dem Erzeugen eines Fensters, wie im obigen Beispiel, verbleibt die weitere Behandlung von Fensteroperationen (Verschieben, Vergrößern, Hervorholen), das Mouse-Over-Highlighting sowie die Behandlung des Tastatur-Fokus komplett in der Hand des GUI-Servers. Die Anwendung wird nur über Ereignisse benachrichtigt, für welche sie explizit Interesse bekundet, z. B. die Aktivierung eines Buttons durch den Benutzer.

Kontrollfluss zwischen Anwendung und GUI-Server
Die originale Version des DOpE GUI-Servers wurde als Teil eines Multi-Tasking-Betriebssystems eingesetzt. Mehrere Anwendungen kommunizierten über Remote-Procedure-Calls (RPC) mit dem GUI-Server. Dieser bestand aus zwei Threads, einem Interface-Thread zur Behandlung eingehender RPC-Aufrufe und einem Thread zur Ausführung der Transformation der Widget-Hierarchie in Pixel. Für Genode-FX haben wir die Interaktion zwischen Anwendungen und dem GUI-Server in einen Kontrollfluss vereint, sodass der gesamte GUI-Server in Form einer Bibliothek verwendet werden kann. Somit ist es möglich, die GUI-Funktionalität ohne unterliegendes Betriebssystem direkt auf der MicroBlaze CPU auszuführen. Multi-Threading kann zwar für eine Anwendung genutzt werden, ist jedoch keine zwingende Voraussetzung.

Die Genode-FX-Software operiert zyklisch in drei Schritten. Zunächst werden Benutzereingaben behandelt. Aus Sicht des GUI-Servers sind dies Transaktionen des Benutzers auf dem GUI-Modell. In einem zweiten Schritt wird die Anwendung mittels Event-Callback-Funktionen aktiviert. Dies kann wiederum eine Menge von Transaktionen zur Folge haben. Da diese Transaktionen jedoch leichtgewichtige Operationen darstellen, haben sie nur geringen Einfluss auf die Ausführungszeit der Anwendung, sodass eine Worst-Case-Execution-Time-Analyse des Codes der Anwendung unabhängig vom Aufwand der Zeichenoperationen im GUI-Server möglich ist. In einem dritten Schritt transformiert der GUI-Server das GUI-Modell in Pixel, wobei die maximale Anzahl an transformierten Pixeln pro Iteration limitiert werden kann. Durch diese Begrenzung lässt sich sicherstellen, dass die Behandlung von Benutzereingaben und die Aktivierung der Anwendung mit beschränkter Latenz stattfindet. z. B. kann bei einem Pixeldurchsatz von einer Million Pixeln pro Sekunde durch die Begrenzung der Transformation auf 10.000 Pixel pro Iteration eine Reaktionszeit von 10 ms auf Benutzereingaben erreicht werden.

4. Evaluation

Die Planung von Zeichenoperationen basiert auf der Annahme, dass sich der Aufwand zum Erzeugen eines Bildbereichs aus dem GUI-Modell proportional zu dessen Größe in Pixeln verhält. Im Folgenden sind zwei Experimente zur Validierung dieser Annahme beschrieben. Für beide Experimente kam das in Abschnitt 2 beschriebene Hardware-Design zum Einsatz und wurde auf einem Xilinx Spartan3A-700 Starter-Kit implementiert. Das Design benötigt 7.512 von 11.776 (64%) der Look-Up-Tabellen und 6.184 von 11.776 (52%) der im FPGA verfügbaren Flip-Flops. Der DDR2-SDRAM ist über einen 100 MHz getakteten 16-Bit-Datenbus mit dem MPMC verbunden. Die MicroBlaze CPU und der PLB sind mit 50 MHz getaktet, wobei dieser Takt ebenfalls als Pixeltakt dient. Die MicroBlaze CPU wurde mit 4 KB Daten- und Instruction-Caches konfiguriert. Der Speicherdurchsatz beträgt 16,21 MB/s (Lesen), 23,68 MB/s (Schreiben) bzw. 13,24 MB/s (Kopieren), gemessen mit einer For-Scheife, die auf 32-Bit-Worten operiert. Für beide Experimente wurde die dargestellte GUI-Konfiguration verwendet.

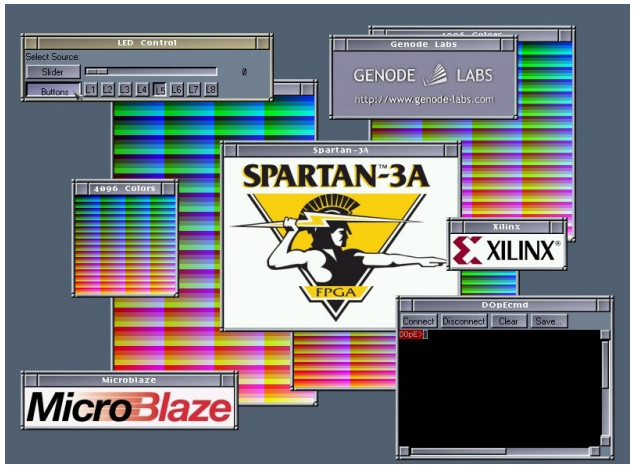

Abbildung 2: Beispiel-GUI-Konfiguration

Das erste Experiment untersucht, inwieweit der Pixeldurchsatz von lokalen Merkmalen der GUI-Konfiguration abhängt, insbesondere von den im Bildbereich dargestellten GUI-Elementen. Für jede Bildposition wurde deren umliegende 16x16-Pixel umfassende Umgebung separat transformiert und vermessen. Der ermittelte Aufwand ist für jede Pixel-Position der Beispiel-GUI-Konfiguration in folgendem Diagramm erfasst.

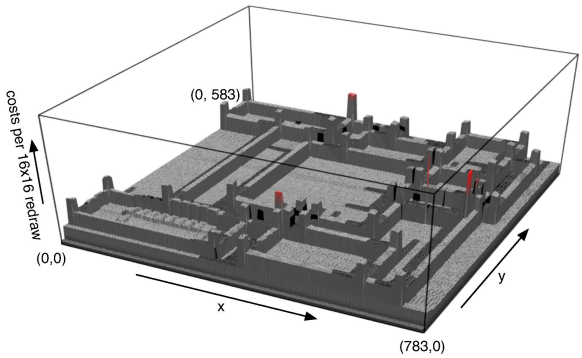

Abbildung 3: Einfluß lokaler Bildmerkmale auf den Transformationsaufwand

Im Idealfall würden wir gleiche Kosten, unabhängig vom transformierten Bildbereich erwarten. Wir können jedoch Anomalien beobachten, die durch den Berechnungsaufwand für die Sichtbarkeitsanalyse (Clipping) an den Kanten von Fenstern entstehen. Dieser Aufwand entsteht an allen Fenstergrenzen, insbesondere an den Ecken. Abgesehen von diesem Effekt ist jedoch zu erkennen, dass der Aufwand zum Transformieren von Fensterinhalten nahezu

konstant ist. z. B. enthält die Region an der linken Ecke eine Menge von Buttons, ein Scale-Widget und Text-Widgets, wobei die meisten anderen Fensterinhalte aus skalierten Bildern bestehen. Der Aufwand ist jedoch nahezu unabhängig von den dargestellten Widget-Typen.

Das zweite Experiment untersucht die Annahme, dass der Pixeldurchsatz unabhängig von der Größe des zu transformierenden Bildbereichs ist. Für jede Position (x,y) des linken oberen Bildviertels der Beispiel-GUI-Konfiguration wurde eine separate Transformation der Pixel zwischen der linken oberen Bildecke (0,0) und der Position (x,y) vermessen und der gemessene Aufwand durch die Anzahl der transformierten Pixel geteilt. Das Diagramm in Abbildung 4 zeigt die Kosten pro Pixel für jede (x,y)-Kombination, wobei Transformationen mit weniger als 8 Pixeln Höhe bzw. Breite nicht dargestellt sind.

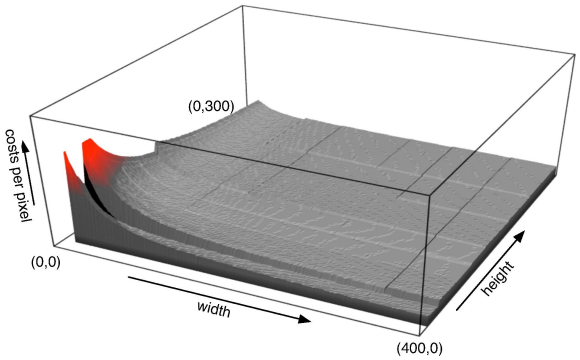

Abbildung 4: Pixeldurchsatz für Transformationen verschiedener Höhe und Breite

Es lässt sich beobachten, dass sehr dünne Transformationen einen Mehraufwand mit sich bringen. In diesen Fällen fällt der Aufwand, die Widget-Hierarchien zu traversieren, gegenüber den eigentlichen Zeichenoperationen stärker ins Gewicht. Eine zweite Beobachtung sind die deutlich sichtbaren Cache-Effekte für Transformationen mit weniger als 14 Pixeln Höhe. In der Praxis treten extrem schmale Transformationen selten auf. Der von uns verfolgte Ansatz, Pixel-Transformation von den Transaktionen der Anwendungen zu trennen, führt dazu, dass mehrere Transaktionen in einer gemeinsamen größeren Transformation resultieren. Somit bleibt der Einfluss der beobachteten Randfälle im praktischen Betrieb marginal.

Auf unserer Hardware-Plattform können wir einen durchschnittlichen Pixeldurchsatz von 1,4 Millionen Pixel/s beobachten. Daraus lässt sich die maximale Latenz, mit der eine Transaktion einer Anwendung auf dem Bildschirm sichtbar wird, ableiten. Für eine Auflösung von 800x600 Pixeln erreichen wir eine worst-case Latenz von 344 ms (480,000 Pixel / 1,4 Millionen Pixel/s). Bei einer Auflösung von 640x480 Pixeln beträgt die Latenz nur 220 ms. Mit der Kenntnis des Pixeldurchsatzes ist es möglich, eine sinnvolle obere Schranke für die Größe einer einzelnen Transformation zu wählen. Um eine Reaktionszeit auf Benutzereingaben von maximal 20 ms zu gewährleisten, sind die Transformationsschritte auf maximal 28,000 Pixel pro Iteration zu begrenzen.

5. Fazit und Ausblick

Die im Artikel vorgestellte Hardware-Software-Kombination ermöglicht die Realisierung komplexer grafischer Benutzeroberflächen mittels low-cost FPGAs. Besonderes Augenmerk wurde auf die Einfachheit des Hardware-Designs und die optimierte Nutzung der begrenzten Hardware-Ressourcen gelegt. Die wichtigste Eigenschaft der Lösung besteht in der Möglichkeit, sowohl eine garantierte obere Schranke für die Latenz grafischer Ausgaben als auch garantierte Reaktionszeiten auf Benutzereingaben zu gewährleisten.

Sowohl die speziell für Genode-FX entwickelten Hardware-Komponenten als auch der Software-Stack werden als Open-Source-Projekt [3] entwickelt und können auf einer Vielzahl von FPGA-Plattformen eingesetzt werden. Beispielsweise wird Genode-FX auf der in [4] beschriebenen Virtex5-basierten Rapid-Prototyping-Plattform eingesetzt. Neben dem Einsatz als GUI-Lösung in Geräten, die bereits einen FPGA besitzen, sehen wir eine Vielzahl von Anwendungsfällen, wie Fahrkartenautomaten, in denen eine FPGA-basierte Grafiklösung wie Genode-FX einen eingebetteten PC ablösen könnte. Dieser Ansatz bietet neben einer deutlichen Verringerung der Systemkomplexität und damit reduzierter Fehleranfälligkeit auch das Potential zum Einsatz energieeffizienterer und kostengünstigerer Geräte.

Danksagung

Vielen Dank an Matthias Alles, der das Projekt ursprünglich initiierte und den Autor einlud, unsere Hardware und Software-Entwicklungen in einem gemeinsamen Projekt zu vereinen.

Literaturverzeichnis

[1] N. Feske and H. Härtig. Demonstration of DOpE — a Window Server for Real-Time and Embedded Systems. In 24th IEEE Real-Time Systems Symposium (RTSS), pages 74–77, Cancun, Mexico, Dec. 2003.

[2] N. Feske, Securing Graphical User Interfaces, Dissertation, TU Dresden, 2009

[3] Genode-FX Sourceforge: http://fpga-graphics.org Genode Labs: http://www.genode-labs.com

[4] M. Alles, T. Lehnigk-Emden, C. Brehm, and N. Wehn. A Rapid Prototyping Environment for ASIP Validation in Wireless Systems. eda-Workshop'09, Dresden, Mai 2009.