Ein Computer-Vision Projekt - zehn Plattformen

Modellbasiertes Design erleichtert die Arbeit (ein Stück weit)

Alexander Wirthmüller, MPSI Technologies GmbH

Nach der initialen Auswahl einer zu einem Projekt passenden MCU- oder FPGA-Plattform wird oft wenig Wert darauf gelegt, die Code-Basis flexibel genug zu halten, um mit überschaubarem Aufwand auf eine Alternativ-Plattform wechseln zu können. Gründe für einen Wechselwunsch können Anforderungsänderungen, Nicht-Verfügbarkeiten oder attraktive neu erscheinende Devices sein. In diesem Erfahrungsbericht werden einige Möglichkeiten aufgezeigt, Plattform-Unabhängigkeit auch in komplexen Embedded Software Projekten zu erhalten.

Für die Mehrzahl von Embedded-Software Projekten verbleibt selbst nach detaillierter Anforderungsanalyse eine breite Auswahl von MCU's oder FPGA(-SoC)'s verschiedener Hersteller, die dazu geeignet sind, die gefragte Funktionalität unter Gesichtspunkten wie Preis oder Leistungsbedarf umzusetzen. Da auch die Programmier- (C, C++) und Beschreibungssprachen (VHDL, Verilog) im Embedded-Bereich weitgehend „gesetzt" sind, also unabhängig vom Chip-Hersteller angewandt werden können, vollzieht sich der „vendor lock-in" an anderer Stelle – beispielsweise durch die zu starke Gewöhnung an komfortable grafische IDE's der Hersteller.

In dieser Abhandlung wird zunächst ein Computer-Vision-Projekt als Multi-Plattform-Beispiel vorgestellt, gefolgt von der gewählten Software-Architektur sowie deren modellbasiertem Design. Weiter wird das Vorgehen zum automatisierten Bauen und Testen der Software-Artefakte beschrieben.

Kompakter 3D Laserscanner

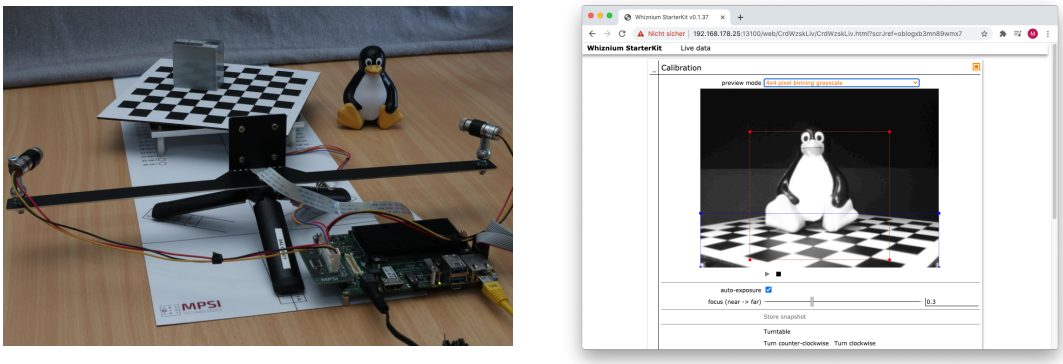

Abbildung 1: Kompakter 3D Laserscanner mit Software (kleiner Auszug)

Der in Abbildung 1 gezeigte kompakte 3D Laserscanner wird einerseits mit dem Anspruch entwickelt, einfach portierbar zu sein, und er soll andererseits im Rahmen einer (teil-)automatisierten CI-/CD-Pipeline stets auf zahlreichen Plattformen lauf- und testfähig gehalten werden.

Projekt-Hintergrund ist der Nachweis der „cross-platform“ Tauglichkeit der Open Source Whiznium Entwicklungswerkzeuge zur modellbasierten Code-Generierung [1]. Die gewonnenen Erkenntnisse sollten gut auf typische Embedded-Entwicklungsszenarien übertragbar sein: genannt seien verschiedene parallel vorzuhaltende Embedded-Produkt-Varianten oder eben der Austausch der MCU-/FPGA-Plattform von einer Produktgeneration zur nächsten.

Hardware-Varianten

Die anzusteuernde Hardware besteht zum Einen aus einem Schrittmotor-getriebenen Drehteller und zwei in der Helligkeit modulierbaren Linienlasern – hier kommen PWM und SPI mit moderater Datenrate (< 2 Mbps) zum Einsatz. Zum Anderen wird die Bildinformation, aus der wiederum eine 3D Punktewolke berechnet wird, von einem > 5 Megapixel Bildsensor geliefert mit Datenraten bis zu 240 MByte/s, wofür entweder ein paralleles 50 MHz Legacy-Interface oder ein 2/4-lane MIPI CSI-2 Interface von Nöten ist.

Plattform (Kürzel)	**Eigenschaften**
Toradex Ixora with Apalis iMX6Q (apalis)	quad armv7, host + device CPU-based
Digilent Arty Z7-20 (arty)	Xilinx FPGA-SoC (dual armv7 host)
Lattice CrossLink-NX eval. board (clnxevb)	ubuntu host + FPGA device via PCIe
terasIC Cyclone V GX starter kit (cvgstk)	ubuntu host + Altera FPGA dev. via PCIe
Microchip PolarFire SoC Icicle kit (icicle)	FPGA-SoC (quad RISC-V 64 host)
Aries MCV eval. platform (mcvevp)	Altera FPGA-SoC (dual armv7 host)
Digilent Nexys Video (nexysv)	Ubuntu host + Xilinx FPGA dev. via PCIe
Efinix Ti180 development board (titdvk)	FPGA (quad soft RISC-V 32 host)
SiLabs UniversalBee development kit (uvbdvk)	ubuntu host w. USB cam + 8-bit MCU dev.
Avnet Zynq UltraScale+ dev. kit (zudvk)	Xilinx FPGA-SoC (dual aarch64 host)

Tabelle 1: Hardware-Varianten

Alle Projekt-Ausprägungen, siehe Tabelle 1, haben gemeinsam, dass ein Linux-Host die high-level Steuerung übernimmt und die Schnittstelle zum Anwender zur Verfügung stellt. Die Aufgabenverteilung bei Ansteuerung und Algorithmik hingegen unterscheidet sich von Plattform zu Plattform, was nicht zuletzt einen Einfluss auf die Host-Device Schnittstelle hat, wie in Abbildung 2 dargestellt.

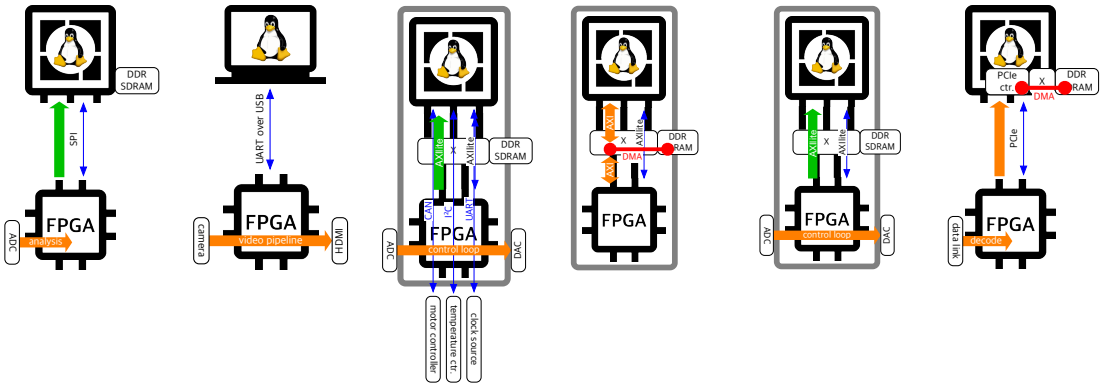

Abbildung 2: Host-Device Anbindungen
(Bandbreite orange: hoch, grün: mittel, blau: gering; FPGA-SoC Varianten 3-5)

Software-Komponenten

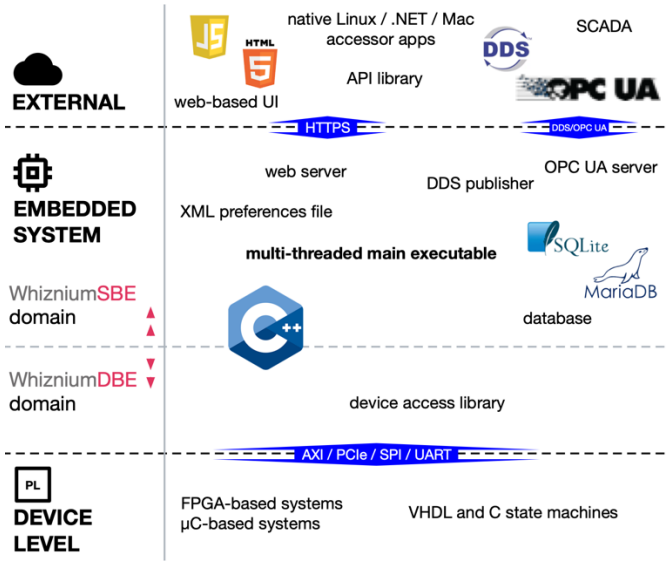

Abbildung 3: Übersicht Softwareprojekte mit WhizniumDBE & SBE

Die Software besteht – plattformübergreifend – Host-seitig aus einem einzelnen WhizniumSBE [2] Projekt namens „Whiznium StarterKit“ [3] sowie Device-seitig aus einem einzelnen WhizniumDBE [4] Projekt namens „Whiznium StarterKit Device“ [5]. Die Aufgabenverteilung ist aus Abbildung 3 ersichtlich.

Universelle modellbasierte Entwicklung (Host-Seite)

Die Linux Host-Software ist derart gestaltet, dass sie auf beliebigen Plattformen kompiliert werden kann, was lediglich erfordert, dass sich einige wenige Plattform-spezifische Funktionsblöcke (z.B. NEON-Instruktionen für armv7/aarch64) innerhalb von #ifdef-Blöcken befinden. Die Zielplattform ist ein einfacher Parameter in der XML-Einstellungsdatei, welche beim Programmstart ausgewertet wird.

```
+ RootWzsk
  + SessWzsk
    + CrdWzskLlv
      + PnlWzskLlvCamera
        + JobWzskAcqPreview/S
          + JobWzskAcqFpgapvw/S
            - JobWzskSrcClnxevb/S (clnxevb)

+ RootWzsk
  + SessWzsk
    + CrdWzskLlv
      + PnlWzskLlvCamera
        + JobWzskAcqPreview/S
          + JobWzskAcqFpgapvw/S
            - JobWzskSrcArty/S (arty)

+ RootWzsk
  + SessWzsk
    + CrdWzskLlv
      + PnlWzskLlvCamera
        + JobWzskAcqPreview/S
          - JobWzskSrcV4l2/S (apalis)
```

Web UI *jobs* in blau, kommunizieren über HTTP(S) JSON/XML Blöcke

Preview acquisition *job* in grün, reagiert auf neue Frames und leitet siw an Web UI *job* weiter

FPGA preview *job* in orange, fragt FPGA Vorschau-Buffer Status ab und transferiert Daten

Source *jobs* in rot, interagieren mit FPGA (UART-over-USB vs. AXIlite) bzw. mit Kamera über die V4L2 API

Abbildung 4: Hierarchische Laufzeitstruktur von *jobs* für drei Plattformen

Hardware-Abstraktion wird erleichtert durch die hierarchische Laufzeit-Struktur von C++ Objekten („*jobs*“), bei welcher stets der für die Plattform passende Datenpfad gewählt wird. Die entsprechende Struktur und das Messaging zwischen den Objekten ist Teil der modellbasiert generierten Code-Struktur eines jeden WhizniumSBE Projektes. In Abbildung 4 ist dies für die Aufnahme, Aufbereitung und Anzeige von Vorschau-Bildern illustriert.

Im Zuge der Entwicklungsarbeit wurden neben der modellbasierten Code-Generierung folgende „cross-platform“-erleichternde Aspekte identifiziert:

- die Verfügbarkeit von Linux sowie der Compiler-Toolchains auf x86_64, armv7, aarch64 und RISC-V 32/64
- die Yocto und Buildroot Projekte zum Bauen eines Plattform-spezifischen Linux
- die Verfügbarkeit von Linux „Board Support Packages“ zu den verwendeten System-on-Module‘s (SoM’s)

Universelle modellbasierte Entwicklung (Device-Seite)

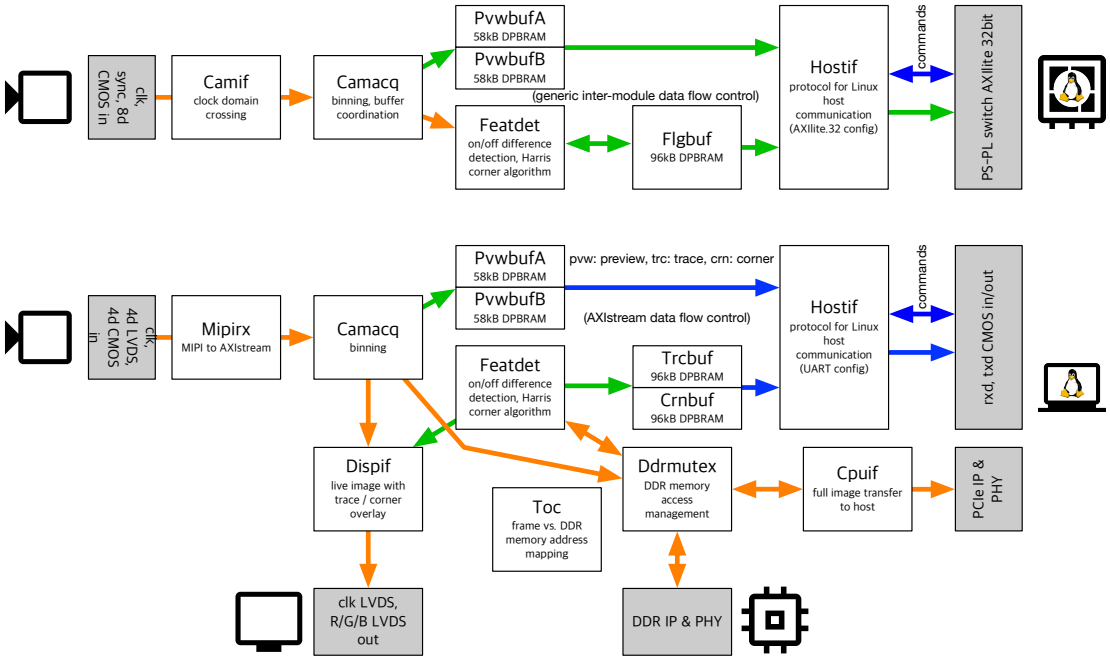

Abbildung 5: „minmale“ / arty (oben) und „maximale“ / nexysv (unten) Ausprägung der FPGA-Device-Funktionalität

FPGA-Designs sind intrinsisch modular und mit Zustandsautomaten aufgebaut, so dass es nahe liegt, Funktionalität feingliedrig zu verteilen. In Abbildung 5 finden sich zwei Ausbaustufen der FPGA-Funktionalität mitsamt der wichtigsten (VHDL-)Module.

Im FPGA-Kontext ist die Verwendung von Hersteller-spezifischen Funktions-(„IP“-) Blöcken meist nur dann notwendig, wenn es um die Ansteuerung Silizium-spezifischer Funktionalität geht, beispielsweise DDR-Speicher-Schnittstellen oder PCIe Transceivern.

Auf [6] wird ein Ansatz verfolgt, sich mittels Kapselung selbst hier herstellerunabhängig zu machen.

MCU-Designs sind im Gegensatz zu FPGA-Designs nicht zwangsweise modular und mit Zustandsautomaten aufgebaut, sie können es aber durchaus sein. WhizniumDBE generiert entsprechende MCU-Software effizient und platzsparend, wie in der 8-bit SiLabs MCU Ausprägung des Projektes gezeigt: dessen C-Code hat hohen Wiedererkennungswert mit dem ansonsten im Projekt vorherrschenden VHDL-Code.

Folgende Aspekte erleichtern „cross-platform“ Entwicklung Device-seitig:

- die Hardware-Abstraktion (identische Bytecode-Kommunikation über UART, SPI und AXIlite) der Host-Device-Schnittstelle in ein Linux *character device* bzw. die Verwendung von Userspace I/O für PCIe/DMA
- die VHDL Code-Entwicklung in Visual Studio Code und Validierung des Synthese-Schritts im plattform-agnostischen GHDL
- der weitgehende Verzicht auf grafische Features der FPGA-Vendor-IDE’s

Continuous Integration und Continuous Deployment

Ein Jenkins-basierter Workflow ist im Aufbau begriffen und wird, sobald verfügbar, auf der Projekt-Homepage [6] zur Verfügung gestellt. Eckpunkte sind:

- Auslöser eines neuen Builds: Aktualisierung von Whiznium oder Templates hierin, Überholung von Features, Hinzufügen von Plattformen
- *Build*-Systeme und *Toolchains*: Yocto, Buildroot, GNU Compiler Collection, FPGA-Vendor Syntese- und Place&Route-Werkzeuge
- Artefakte: Executables, Bitstreams, Memory Images
- Aktualisierung der Firmware: Installation über Ethernet auf ubuntu Host-Rechner, A-/B-Update für eMMC-basierte Embedded Linux Systeme, JTAG-over-USB für alle anderen
- Hardware-Testaufbau und Multiplexing: vier Laserscanner mit verschiedenen Kamera-Ausprägungen (USB, parallel CMOS, 2x MIPI CSI-2); entry-level FPGA Board für GPIO-Multiplexing, Efinix Titanium 180 FPGA Board für MIPI Multiplexing; Power-Multiplexing und Strom-Messung per high-side ADC in SiLabs UniversalBee MCU Board
- automatische Messgrößen und *Pass*-/*Fail*-Kriterien: Boot-Dauer und -Stromverlauf, Reaktion der Host-Software auf API Calls, Überwachung des OPC UA Namespace
- manuelle Messgrößen: Vorschaubilder und 3D-Punktewolke

Zusammenfassung

Es wurde anhand eines umfassenden Beispiels gezeigt, welche Aspekte helfen, Embedded-Software-Projekte möglichst plattformunabhängig zu entwickeln. Während es auf den ersten Blick mit zusätzlichem Aufwand verbunden sein mag, auf die ein oder andere grafische Komfort-Funktion der Hersteller zu verzichten, so sind die verfügbaren

Open Source Entwicklungswerkzeuge hinreichend leistungsfähig, um dennoch herstellerunabhängig zum Ziel zu kommen. Dies zahlt sich spätestens bei einem anstehenden Plattform-Wechsel aus.

Referenzen

[1] Whiznium Dokumentation auf GitHub: https://github.com/mpsitech/The-Whiznium-Documentation
[2] Whiznium Service Builder's Edition auf GitHub: https://github.com/mpsitech/wznm-WhizniumSBE
[3] Whiznium StarterKit auf GitHub: https://github.com/mpsitech/wzsk-Whiznium-StarterKit
[4] Whiznium Device Builder's Edition auf GitHub: https://github.com/mpsitech/wdbe-WhizniumDBE
[5] Whiznium StarterKit Device auf GitHub: https://github.com/mpsitech/wskd-Whiznium-StarterKit-Device
[6] Projekt für herstellerunabhängige High-Speed FPGA Interfaces auf GitHub: https://mpsitech.github.io/Laser-Scanner-By-Platform

(Alle Links aufgerufen am 5. November 2023)

Autor

Alexander Wirthmüller ist Inhaber des Start-ups MPSI Technologies, welches er nach umfangreichen Erfahrungen in forschungsnahen Anwendungen 2016 in München gegründet hat, um Systementwicklern die Zeit zurückzugeben, die bei klassischer Programmierung durch repetitive Coding-Aufgaben verloren geht. Nach einem ersten kommerziellen Mikrocontroller-basierten Projekt als Schüler, konnte Alexander während seines Studiums sowie danach in diversen Forschungseinrichtungen zahlreiche Projekte, von Leistungselektronik, über verteiltes Rechnen, bis hin zu Photonik mit Anwendungen in der Robotik, bearbeiten bzw. begleiten. Hierbei spielte stets der Gedanke, mittels effizient programmierter Software zu besseren Ergebnissen zu kommen, eine wichtige Rolle. Alexander besitzt ein Diplom in Elektrotechnik von der ETH Zürich.

Kontakt

Internet: www.mpsitechnologies.com
Email: aw@mpsitechnologies.com