

Masterarbeit

Andre Jestel

**Software-Partitionierung einer Laserscanner-
basierten Objekterkennung auf einer
MPSoC-Plattform mit Android**

Andre Jestel

**Software-Partitionierung einer Laserscanner-
basierten Objekterkennung auf einer MPSoC-Plattform
mit Android**

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Bernd Schwarz
Zweitgutachter: Prof. Dr. rer. nat. Michael Schäfers

Eingereicht am: 14. Februar 2013

Andre Jestel

Thema der Arbeit

Software-Partitionierung einer Laserscanner-basierten Objekterkennung auf einer MPSoC-Plattform mit Android

Stichworte

MPSoC, SoC, SMP, Android₄, FAUST, OMAP₄, Cortex-A₉, Laserscanner, HPEC, RANSAC

Kurzzusammenfassung

Diese Masterarbeit beschreibt die Software-Partitionierung und -Implementierung einer Laserscanner-basierten Objekterkennung für einen ARM CortexA₉-DualCore Prozessor auf der "Open Multimedia Application Platform 4430" (OMAP₄) unter Verwendung des Android 4 Betriebssystems. Ziel der Arbeit im Rahmen des "Fahrerassistenz- und Autonome Systeme" (FAUST) Projektes der HAW Hamburg ist die Erprobung von Technologien für Laserscanner-gestützte Ausweich-, Brems- und Einparkassistentenfunktionen, die auf einem SoC-Modellfahrzeug im Maßstab 1:10 validiert werden. Die Abstandswerte vom Laserscanner aus einem 180° Scanbereich werden dabei mit einer Winkelauflösung von 0,35° durch ein Schwellwertverfahren in zusammenhängende Objekte segmentiert.

Title of the paper

Software partitioning of laserscanner-based object recognition on MPSoC running Android

Keywords

MPSoC, SoC, SMP Android₄, FAUST, OMAP₄, Cortex-A₉, Laserscanner, HPEC, RANSAC

Abstract

This master thesis describes the software partitioning and implementation of a laserscanner-based object recognition for an ARM CortexA₉-DualCore processor on "Open Multimedia Application Platform 4430" (OMAP₄) running the Android 4 operation system. Goal of this thesis within the FAUST project is the technology field trial of advanced driver assistance systems validated for a vehicle on a scale of 1:10. The received distance values from the laser scanner out of a 180 degree scan range with an angular resolution of 0.35 degrees are segmented to related objects by a thresholding procedure.

Inhaltsverzeichnis

| | | |
|----------|---|-----------|
| 1 | Einleitung | 1 |
| 1.1 | Kapitelübersicht | 3 |
| 2 | Übersicht und Konzepte | 4 |
| 2.1 | Portierung eines Schwellwertverfahren zur Segmentklassifizierung | 5 |
| 3 | Laserscanner-basierte Umgebungserfassung | 9 |
| 3.1 | Einsatz von Umfelderkennungssensorik im Automobil | 11 |
| 3.2 | Prinzipien der Lasertechnologie | 14 |
| 3.3 | Analyse und Bewertung der Genauigkeit des Laserscanners | 16 |
| 4 | Der ARMv7 Cortex-A9 Zweikern-Prozessor | 19 |
| 4.1 | Prozessorarchitektur mit Interprozessor-Interfaces | 20 |
| 4.2 | Memory Management Unit für virtuelle Speicherverwaltung | 23 |
| 4.3 | Speicherbarrieren und Reihenfolge von Assembler-Instruktionen | 27 |
| 5 | Inbetriebnahme: "Open Multimedia Application Platform 4430" | 31 |
| 5.1 | DualCore-Initialisierung mit Bootloader | 32 |
| 5.2 | PowerVR SGX540 GPU-beschleunigte Visualisierung | 34 |
| 5.3 | PWM-Signalgenerierung durch externen AVR- μ C | 36 |
| 6 | Konfiguration und Entwicklungsumgebung für Linaro Android 4.0.3 | 39 |
| 6.1 | Formatierung der SD-Karte mit Betriebssystemabbild | 40 |
| 6.2 | Eclipse-Plugin "Android Development Tools" auf Windows Host-PC | 44 |
| 6.3 | "Native Development Kit" zur I ² C-Bus Steuerung | 46 |
| 7 | System-Integration und Software-Implementierung | 49 |
| 7.1 | USB-Schnittstellenzugriff auf den <i>Hokuyo URG-04LX</i> Laserscanner | 52 |
| 7.2 | Verwendung des seriellen SCIP 2.0 Kommunikationsprotokolls | 54 |
| 7.3 | Objekterkennung mit <i>RANSAC</i> | 57 |
| 7.4 | Mutex-basiertes Thread-Synchronisationskonzept | 62 |
| 8 | Zusammenfassung und Ausblick | 64 |
| | Anhang A bis D | 72 |

1 Einleitung

Eingebettete Systeme wie in Smartphones und autonomen Fahrerassistenzsystemen bestehen aus "System-on-a-Chip" (SoC) Plattformen und werden aktuell von "Multiprocessor System on a Chip" (MPSoC) Plattformen abgelöst [52, 47]. Die Bedeutung der eingebetteten Systeme im Hinblick auf Arbeitsplätze und Wertschöpfung für den Industriestandort Deutschland steigt weiter an [13]. Für die Automatisierung von Prozessabläufen ist reaktives Verhalten von Sensorik und Aktorik durch Algorithmen auf eingebettete, hybride Software- und Hardwarearchitekturen abzubilden. Dabei gilt es eine zunehmende Informationsmenge von der Signalverarbeitungskette prozessgerecht unter Einhaltung von Echtzeitanforderungen aufzubereiten.

Die Einhaltung dieser Anforderungen durch Steigerung der CPU-Taktfrequenzen ist nicht mehr realistisch, da die Frequenzen von aktuellen Prozessoren auf Grund der Tatsache stagnieren, dass die Ausbreitungsgeschwindigkeit der elektrischen Impulse in einem Prozessor nicht höher als die Vakuumlichtgeschwindigkeit ($c \approx 300 \cdot 10^6 \frac{m}{s}$) sein kann. Zum Zurücklegen der durchschnittlichen Flächendiagonale eines aktuellen Prozessors ist eine Taktperiode von rund 0,2 Nanosekunden erforderlich, so dass die Taktfrequenz physikalisch bei 5 Gigahertz ihr Maximum erreicht. Thermodynamische Effekte aufgrund der hohen Integrationsdichte der Halbleiter in den Chips führen dazu, dass diese Grenze nicht erreicht wird und verursachen einen steigenden Energieverlust durch Abwärme bei zunehmender Taktfrequenz.

Eine parallele Verarbeitung auf mehreren Prozessorkernen dient bei gleichbleibender Taktfrequenz zur Steigerung des Instruktionsdurchsatzes bei unveränderter Verlustleistung des Gesamtsystems. Die zusätzliche Rechenkapazität mehrerer Prozessorkerne trägt bei Video-, Audio-, und Kommunikationsanwendungen zur Einhaltung der Echtzeitanforderungen bei. Rechenintensive Funktionen des Systems, die sich im Rahmen eines Hardware/Software-Codesigns [22] nicht in Form von "Register Transfer Level" (RTL) Modellen auf "Field Programmable Gate Array" (FPGA) auslagern und synthetisieren lassen, können als Software-Implementierung bestehen bleiben und mit geringeren Aufwand parallelisiert werden. Die Aufgabenverteilung der Softwareprozesse auf die verfügbaren Prozessorkerne erfolgt entweder prioritätsgesteuert vom Betriebssystem dynamisch zur Laufzeit (*symmetric multiprocessing*, SMP) oder wird zur

Entwicklungszeit statisch festgelegt (*asymmetric multiprocessing*, AMP), wie zum Beispiel bei einer dedizierten *Master-Slave* Aufgabenverteilung [33].

Die Ziele und Bestandteile der Masterarbeit umfassen die:

1. Portierung einer Laserscanner-gestützten Objekterkennungssoftware [49] mit erhöhter Messgenauigkeit und Ergebnis-Visualisierung.
2. SMP-Partitionierung unter dem *Linaro* "Android 4.0.3" Betriebssystem [24] auf der "Open Multimedia Application Platform 4430" (OMAP4) [44].
3. Implementierung des "Random Sample Consensus" (RANSAC) [8] Ausgleichverfahrens zur Stabilisierung der erkannten Objekte durch Messdaten-Bereinigung.

Im bestehenden System wurde die Fahrspurführung in MATLAB entwickelt und daraus mit dem *Xilinx System Generator* [54] RTL-Modelle erzeugt. Diese sind auf einem *Spartan-3A DSP FPGA* synthetisiert [53] auf dem sich für die Softwareausführung auch zwei *MicroBlaze-SoftCore* Prozessoren befinden. Geplant ist die Portierung und Integration der Systemkomponenten (Abbildung 1.1) auf die *Zynq-7000 Extensible Processing Platform* [55], auf der sich neben FPGA-Ressourcen der baugleiche *Cortex-A9* Zweikernprozessor [5] der OMAP4430-Plattform befindet, so dass die Lauffähigkeit der Objekterkennung auf der Zynq-7000 Plattform garantiert werden kann. Weiter nutzt das Android-Betriebssystem die OMAP4-Subsysteme zur Hardware-Beschleunigung der Grafikoperationen.

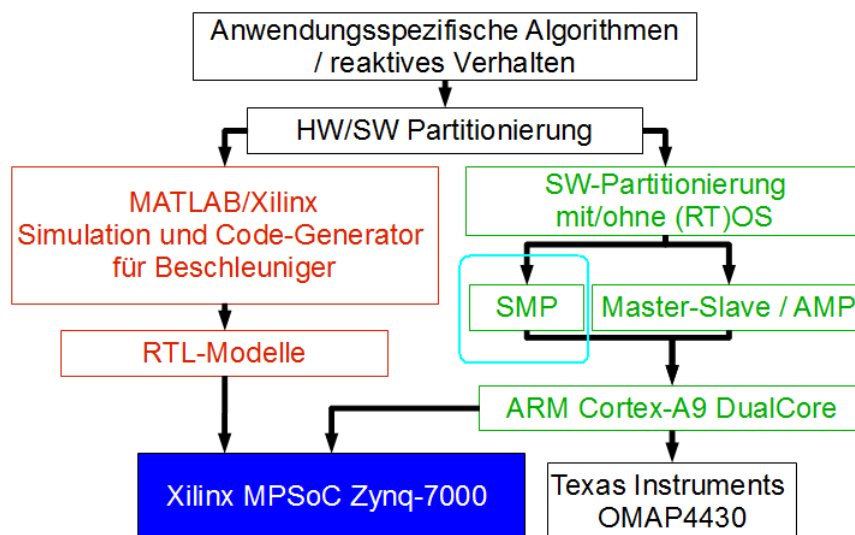


Abbildung 1.1: Roadmap: "Eingebettete Systeme auf rekonfigurierbaren MPSoCs" mit dem Hardware (rot) und Software (grün) Co-Design Partitionierungssträngen. Schwerpunkt der Masterarbeit ist die SMP-Partitionierung einer 2D-Umgebungserfassung (hellblau).

1.1 Kapitelübersicht

1. **Kapitel:** Vorstellung eines Schwellwertverfahrens, das die Abstandswerte des Laserscanners in zusammenhängende Objekte segmentiert. Bei der Portierung auf die OMAP4430-Plattform mit höherem Datendurchsatz, wurde das Verfahren mit höherer Messgenauigkeit implementiert und SMP-Partitioniert partitioniert.
2. **Kapitel:** Übersicht der Lasertechnologie-basierten Sensorik und deren Einsatz im Automobil zur Umgebungserfassung. Vorstellung der parametrisierten Kommunikationsbefehle und einer Genauigkeitsanalyse des Hokuyo Laserscanners URG-04LX.
3. **Kapitel:** Prozessorspezifische Assembler-Instruktionen der Cortex-A9 Architektur zur virtuellen Speicherverwaltung, Interprozessor-Synchronisation und Erhaltung der L1 Cache-Kohärenz werden herausgestellt.
4. **Kapitel:** Die Co-Prozessor Subsysteme der OMAP4430-Plattform zur Hardware Beschleunigung von Datenerfassung und Visualisierung wurden identifiziert und zur Erprobung in Betrieb genommen.
5. **Kapitel:** Die Installationsabläufe des Android 4.0.3 Betriebssystems von Linaro und der Entwicklungswerkzeuge werden beschrieben. Aus den Kopplungsvarianten der OMAP4-Anbindung mit dem Entwicklungs-PC und den externen Peripheriegeräten resultieren mehrere Debug-Konfigurationen, auf die zum Vergleich hingewiesen wird.
6. **Kapitel:** Darstellung Software-Implementierung zur Abbildung der Datenverarbeitungskette, in der Abstandsdatenerfassung, Objekterkennung, Ergebnisvisualisierung und PWM-Ansteuerung realisiert wurden. Schwerpunkt ist das SMP-Entwurfsmuster für die dynamische Aufgabenpartitionierung auf die CPU-Kerne zur Laufzeit bei der sich die Threads mutex-basiert synchronisieren.
7. **Kapitel:** Zusammenfassung und Ausblick auf die FPGA-basierte MPSoC-Plattform "Zynq-7000 All Programmable SoC" von Xilinx mit identischem Zweikern-Prozessor, auf der die Integration der Fahrerassistenzfunktionen im Forschungsprojekt unter einem HW/SW Co-Design geplant ist.

2 Übersicht und Konzepte

Die SMP-partitionierte Android-App zur Laserscanner-basierten Umgebungserfassung mit Echtzeit-Visualisierung auf der OMAP4430-Plattform ist in der Lage den Lenkwinkel des SoC-Fahrzeugs (Abbildung 2.1) durch PWM-Signale zu steuern, die von einem über I²C-Bus angeschlossenen, externen AVR- μ C generiert werden.

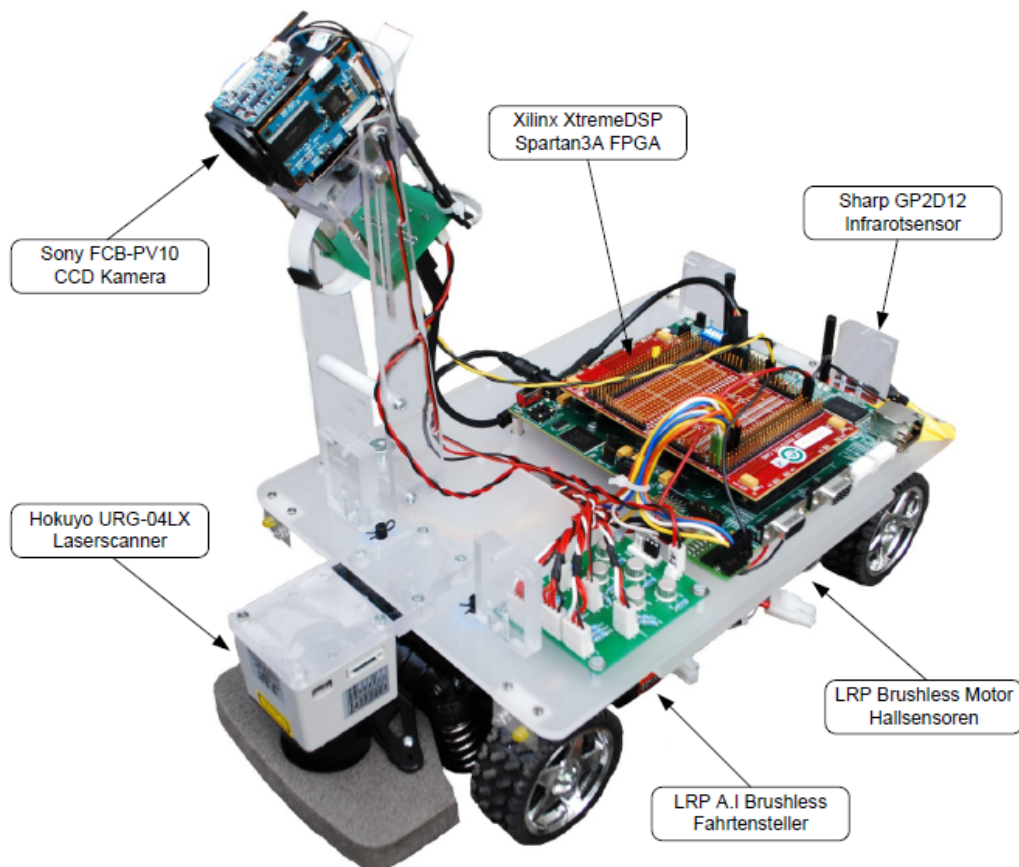


Abbildung 2.1: SoC-Fahrzeug des HPEC-Projekts. [32]

Über einen Entwicklungs-PC, der UART-Konsolenausgaben über einen RS232-auf-USB Konverter empfängt und über Ethernet mit der OMAP4-Plattform verbunden ist, ist die Objekterkennung mit dem "Android Development Tools" (ADT) Eclipse-Plugin für den Cortex-A9 DualCore

implementiert. Das dynamische SMP-Partitionierungskonzept nutzt dabei die "Snoop Control Unit" (SCU) zur Interprozessor-Synchronisation über den gemeinsamen Speicher. Das über eine SD-Karte ausgeführte Android 4.0.3 Betriebssystem von Linaro [34] enthält die OMAP4-Treiber für die Subsysteme zur Hardware-Beschleunigung, die den Zweikern-Prozessor bei der Visualisierung über einen HDMI/DVI Port entlasten (Abbildung 2.2).

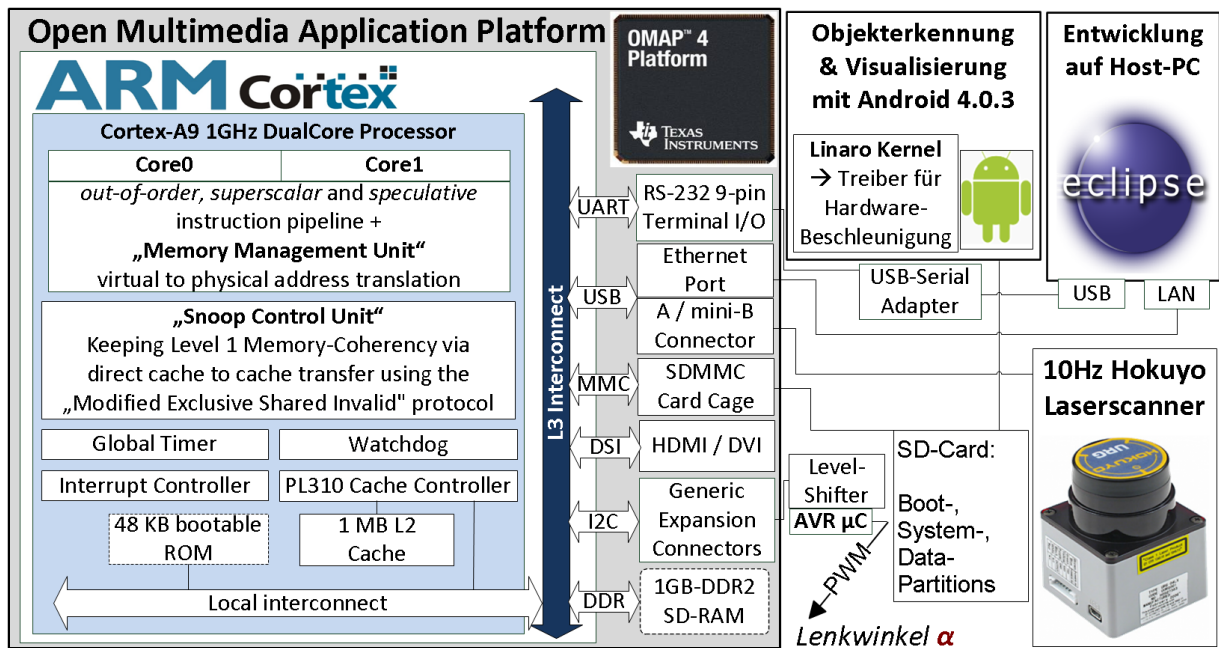


Abbildung 2.2: Technologie-Übersicht der Laserscanner-basierten Objekterkennung.

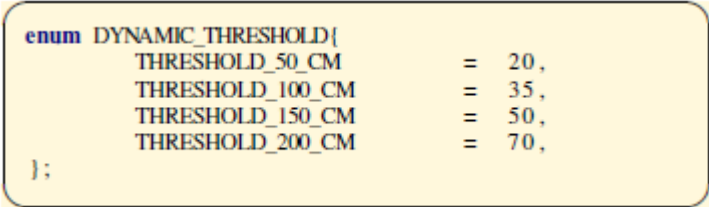
2.1 Portierung eines Schwellwertverfahren zur Segmentklassifizierung

Eine vollständige Segmentierung der Umgebung innerhalb eines eines Scanvorgangs ist gegeben, wenn jeder Abstandswert eindeutig einer zusammenhängenden Regionen zugeordnet ist. Im Bereich der digitalen Bildverarbeitung wird die Segmentierung auch zur Klassifizierung von Objekten innerhalb eines Bildes genutzt. Diese Ansätze werden übertragen auf die Segmentierung von Laserscandaten angewandt, wobei anstatt einen Bildes mit Pixelinformationen eine Menge von Abstandswerten betrachtet wird. Für die Objekterkennung des SoC-Fahrzeugs wurden die Konzepte der Vorarbeit als Grundlage für die Implementierung verwendet, die benachbarten Abstandswerten Segmente generiert. Die Portierung des Schwellwertverfahrens

auf die OMAP4430-Plattform mit höherem Datendurchsatz optimiert die Umgebungserfassung gegenüber den Testergebnissen und Bewertungen der Vorarbeit [49] durch eine erhöhte Messgenauigkeit.

Kontinuierlicher Schwellwert mit *Vector Floating Point* Einheit

Die Wahl eines statischen Schwellwertes führte dazu, dass mehrere zusammenstehende Objekte in Entfernungen von unterhalb 1,5 Meter als ein zusammenhängendes Segment identifiziert wurden und ein einzelnes Objekt, das sich mehr als 2,5 Meter entfernt befindet, auf mehrere Segmente aufteilt wurde. Zur Anpassung an die Abstandswerte wurde der Schwellwert zur Laufzeit aus einer vordefinierten Wertetabelle ausgewählt (Abbildung 2.3).



```
0  enum DYNAMIC_THRESHOLD{
    THRESHOLD_50_CM      = 20,
    THRESHOLD_100_CM     = 35,
    THRESHOLD_150_CM     = 50,
    THRESHOLD_200_CM     = 70,
5  };
```

Abbildung 2.3: LookUp-Tabelle für einen dynamischen Schwellwert in Abhängigkeit der gemessenen Distanz [9].

Vorteil des Tabellenabgleichs ist die proportional zur Tabellengröße garantierte Obergrenze an Speicherzugriffen und Vergleichsoperationen in Festkommaarithmetik. Nachteilig sind die sprunghaften Wechsel bei Abstandswerten nahe den Diskretisierungsgrenzen. Für die OMAP4-Implementierung erfolgt die Berechnung des Schwellwertes durch eine lineare Interpolation der Tabellenwerte ohne diskrete Fallunterscheidungen. Die "Vector Floating Point" (VFP) Einheit eines Cortex-A9 Kerns (Kapitel 4) arbeitet dabei mit bis zu doppelter Genauigkeit nach der IEEE-754 Spezifikation und ersetzt die Näherungsverfahren für trigonometrische Funktionen.

180° Scanwinkel und 0,35° Auflösung

Hindernisse am Ende von Kurvenfahrten erreichen den vom Laserscanner abgedeckten Frontbereich aufgrund des Scanwinkels von 90° erneut, erst nachdem sich das Fahrzeug zum Hindernis ausgerichtet hat (Abbildung 2.4). Durch die Erweiterung des Winkels auf 180° wird das Hindernis bereits vor dem Einlenken erkannt und bleibt während der gesamten Kurvenfahrt als ein identifiziertes Objekt im Scanbereich. Darüber hinausgehend können die seitlichen Umgebungsinformationen für komplexere Ausweichmanöver genutzt werden. Zusätzlich zur Datenreduktion ist der Laserscanner so konfiguriert, dass von 3 aufeinanderfolgenden Abstandswerten den jeweils kleinsten Wert an die OMAP4430-Plattform überträgt. Durch die Auswertung jedes Abstandswertes wird die Winkelauflösung von 1,05° auf 0,35° verringert und

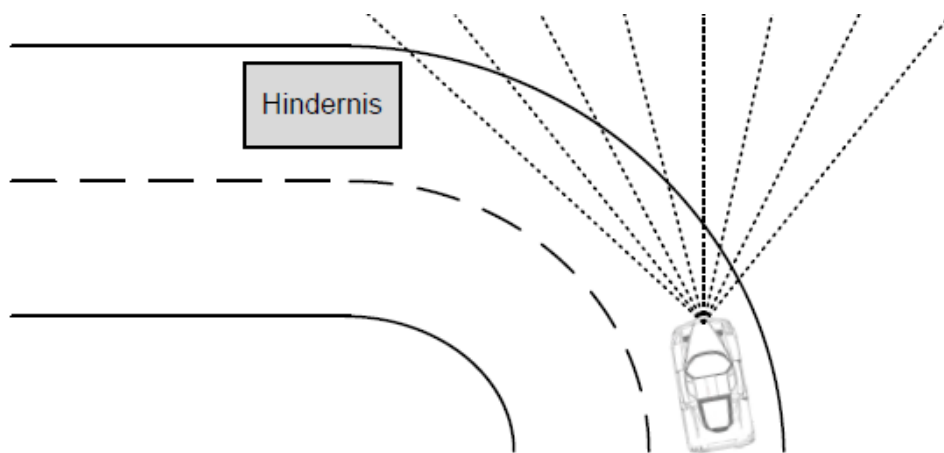


Abbildung 2.4: Nicht garantierte Detektion von Hindernissen während einer Kurvenfahrt [49].

somit die minimale Größe von Objekte um erkannt zu werden. Zusammen mit der Verdopplung des Scanwinkels erhöht sich die Anzahl der Abstandswerte gegenüber der Vorarbeit auf 512.

Entfernungsmesswerte bis zu 5,6 Meter über USB

Bei zuvor verwendeten Kodierung mit 12 Bits für einen Abstandswert war die messbare Entfernung auf 4.095 Millimeter begrenzt. Um die maximal-garantierte Distanz des Laserscanners von 5.600 Millimeter zu nutzen sind, wurde die 3 Byte-Kodierung des Übertragungsprotokolls gewählt. Die Übertragung der Datenpakete vom Laserscanner erfolgte über die RS232-Schnittstelle und war mit einer Baudrate von 115,2 Kbits pro Sekunde konfiguriert. Für die OMAP4430-Plattform werden die Messwerte über das USB 2.0 Protokoll im *Full Speed* Modus mit einer Datentransferrate von 9 Mbits pro Sekunde übertragen. Die Erweiterung des Scanbereiches und der Winkelauflösung hat eine um den Faktor 6 gestiegene Datenübertragung zur Folge, der durch den Schnittstellenwechsel-bedingten Faktor 75 an höherem Datendurchsatz kompensiert wird. Hinzu kommt, dass die Datenpakete nicht mehr im *Request-Response* Verfahren einzeln angefordert und quittiert werden, sondern kontinuierlich vom Laserscanner gesendet, und nach dem Erhalt nur aus dem Empfangspuffer gelesen werden.

Hardware-beschleunigte Visualisierung durch GPU

Für die Auswertung der Objekterkennung wurde in der Vorarbeit ein JAVA-Applet entwickelt (Abbildung 2.5), das die erkannten Objekte aus einer Textdatei liest und visualisiert. Dazu wurden die Abstandswerte einer Segmentierung manuell aus dem Speicher der MPSoC-Plattform auf den Entwicklungs-PC übertragen, in eine Textdatei geschrieben und nach dem Öffnen über das *User Interface* dargestellt. Auf der OMAP4430-Plattform wurde die Visualisierung in die implementierte Android-App integriert. Die Aktualisierung erfolgt von den SW-Threads in

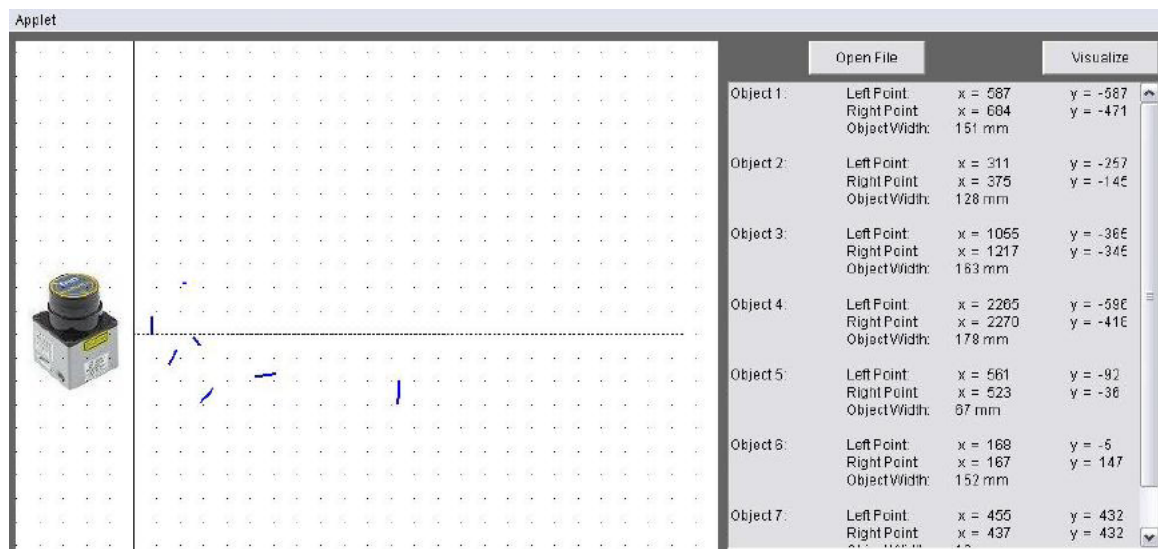


Abbildung 2.5: Java-Applet zur Visualisierung von erkannten Objekten [49].

Echtzeit nach jedem Segmentierungsvorgang. Über die "Android Development Tools" (Kapitel 6.2) werden Momentaufnahmen von der OMAP4430-Plattform über die Ethernet-Schnittstelle angefordert und zum Entwicklungs-PC übertragen. Die Ausgabe der Zeichenoperationen am Monitor wird zur Entlastung der Cortex-A9 Prozessorkerne von der *PowerVR-SGX540* "Graphics Processing Unit" (GPU) Hardware-Subsystem (Kapitel 5.2) der OMAP4430-Plattform zur HDMI/DVI-Schnittstelle geleitet.

Parallelisierung der Messdatenerfassung und Objektsegmentierung

Bei der statischen Partitionierung der SW-Threads auf den Zweikern-Prozessor der Vorarbeit wurden Messdatenerfassung und Objekterkennung sequentiell und auf einem Kern ausgeführt. Der andere Kern sendete Steuersignale zum Start eines Auswertungszyklus und empfing Statussignale zur Konsolenausgabe. In dem 100 Millisekunden Intervall zwischen dem Empfang von zwei aufeinanderfolgenden Messdatenpaketen, war ein Kern nicht an der Objekterkennung beteiligt und der andere vollständig mit der Objektsegmentierung ausgelastet.

Das SMP-Partitionierungskonzept (Kapitel 7) für den Cortex-A9 DualCore verteilt und balanciert die Rechenlast der Threads prioritätsgesteuert zur Laufzeit auf die beiden Kerne. Während ein Kern die empfangenen Messdaten in Objekte segmentiert, steht der andere den Betriebssystem-Prozessen oder anderen Anwendungen zur Verfügung und beginnt mit dem nächsten Segmentierungszyklus sobald die Messdaten vom Laserscanner im USB-Empfangspuffer vorliegen. Die Synchronisation über SW-Mutexe garantiert den gegenseitigen Ausschluss, so dass auf die beiden kritischen Ressourcen USB-Schnittstelle und gemeinsamer Speicher, zu einem Zeitpunkt nur von einem Thread zugegriffen wird.

3 Laserscanner-basierte Umgebungserfassung

Für die Umgebungserfassung im Frontbereich des SoC-Fahrzeugs überträgt ein 2D-Laserscanner (Abbildung 3.1), die gemessenen Abstandswerte über eine Mini-USB Schnittstelle an die OMAP4430-Plattform. Der Laserscanner hat bei Inbetriebnahme eine Stromaufnahme von 800mA und wäh-

| | |
|------------------------------|---|
| Power Source | 5V DC \pm 5% |
| Detection Range | 20mm ~ 5.6m: White Square Sheet 70mm 240° |
| Scan Window | 240° |
| Accuracy | Up to 1m: \pm 10mm. Above 1m 1% of distance |
| Angular Resolution | 0.3515625° (360° / 1,024 steps) |
| Light Source | Laser diode (λ =785nm) Laser safety class 1 |
| Scan Time | 100 msec/scan |
| Sound Level | Less than 25 dB |
| Interface | RS232 + USB 2.0 |
| Synchronous output | NPN open collector |
| Command system | SCIP 1.1 / 2.0 |
| Temperature /Humidity | -10°C -50°C, < 85% RH (no condensation / icing) |
| Vibration Resistance | Double amplitude 1.5mm 10 ~ 55 Hz, 2 hrs in X, Y, Z direction |
| Impact Resistance | 196 m/s ² , 10 times each in X, Y and Z direction |
| Weight | Approx 160g |

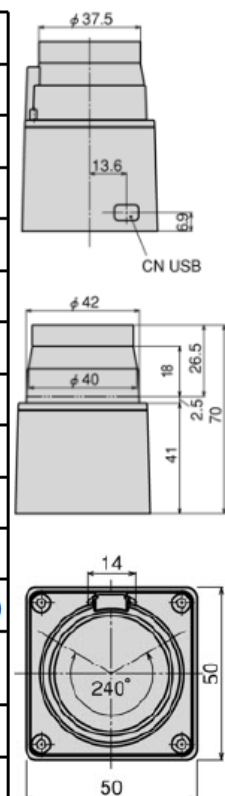


Abbildung 3.1: Spezifizierung und Abmessungen des URG-04LX Laserscanners[28].

rend dem Betrieb einen durchschnittlichen Stromverbrauch von 500mA. Die 5V-Spannungsversorgung erfolgt über die Pins der RS232-Schnittstelle, da der USB-Standard den Stromverbrauch mit maximal 500mA limitiert. Durch eine Photodiode wird der reflektierte Laserstrahl empfangen

und anhand der Phasenverschiebung des modulierten Strahls die Entfernung zum Hindernis berechnet. Der Laserscanner arbeitet nach dem "Amplitude Modulated Continuous Wave" Prinzip, wobei bei konstanter Frequenz die Amplitude und somit die Intensität des Lichts moduliert wird (vgl. Kapitel 3.2). Die Berechnung des korrespondierenden Winkels zu einem gemessenen Abstandswert ergibt sich aus der vom Hersteller angegebenen Anzahl an Schritten. Über eine komplette Rotation des Scankopfs ergeben sich für 360° insgesamt 1024 Schritte. Diese werden in drei Regionen untergliedert, wobei die erste Region den eigentlichen Scanbereich mit den Schritten 44 - 725 in einem Winkel von 240° umfasst. In einem Winkel von 15° werden die Schritte 0 - 44 und 725 - 768 als Toleranzbereich für den Scanvorgang verwendet. Die restlichen Schritte liegen im nicht-scannenden Bereich des Laserscanners (Abbildung 3.2). Bei einen

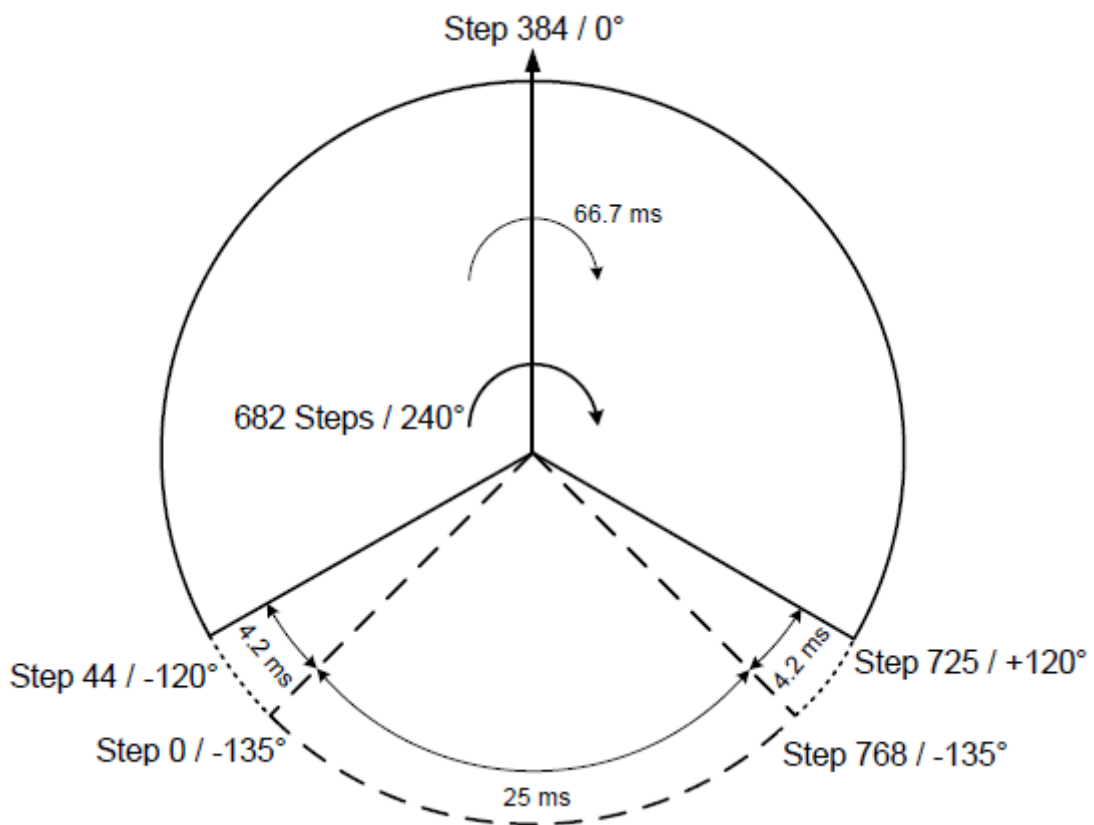


Abbildung 3.2: Laserscannerbereich und die korrespondierende Schritte zur Winkelauflösung.

Öffnungswinkel von 240° und einer Winkelauflösung von $0,35^\circ$ besteht eine Scanzeile aus 682 Messpunkten. Durch ein vom Host initiiertes Kommando werden die gewählten Messpunkte kodiert an den Initiator zurück übertragen. Für die Laserscanner-basierte Objekterkennung impliziert eine Scanfrequenz von 10Hz, dass alle 100 ms eine Aktualisierung der Abstandswerte und somit eine Neu-Generierung der zusammenhängenden Segmente stattfindet. In Testfahrten wurde eine von der Fahrspurführung abhängige maximale Geschwindigkeit von ca. 2 m/s als

realistisch angesehen. Bei dieser Geschwindigkeit wird die Detektion der Fahrspur garantiert und ein Ausbrechen des SoC-Fahrzeugs durch zu hohe Kräfte verhindert. Die Erfassung von Abstandswerten erfolgt mit einer festen Schrittweite, was bedeutet, dass die Verteilung der Messwerte sowohl von der Distanz zwischen Laserscanner und Objekt abhängig ist als auch von der Ausrichtung des Objekts gegenüber dem Scanner. Für eine detaillierte Erkennung von kleinen Objekten, muss der Öffnungswinkel und damit die Schrittweite klein genug sein (Abbildung 3.3).

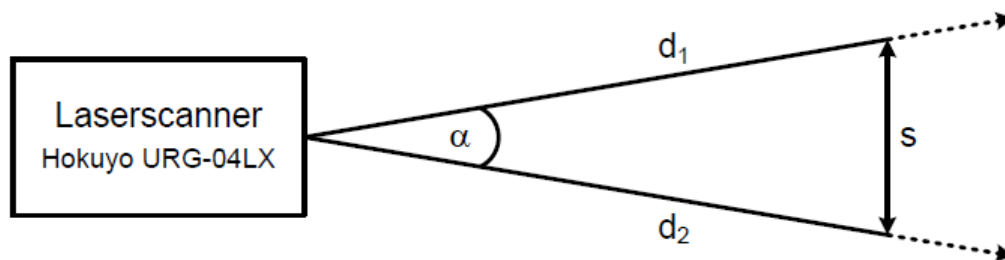


Abbildung 3.3: Der tangentielle Abstand s zwischen zwei benachbarten Messpunkten ist abhängig von der Distanz d und der Winkelauflösung α [49].

3.1 Einsatz von Umfeldersfassungssensorik im Automobil

Zur Erhöhung der aktiven Verkehrssicherheit und des Fahrkomforts ist die Anzahl an Sensorik in der Automobilbranche in den letzten Jahren stetig gestiegen. Eine Vielzahl von Kontroll- und Umgebungsinformationen werden von Sensoren erfasst und durch komplexe Regel- und Steueralgorithmen zur Erhöhung der Sicherheit und des Komforts genutzt (Abbildung 3.4). Die im Automobil eingesetzten Sensoren, die der Vorausschau und der Umgebungserfassung dienen, werden vorwiegend in die Kategorie der aktiven Sensoren eingegliedert [40]. Als erste Umfeldersfassungssensorik wurden Ultraschallsensoren zur Abstandsmessung eingesetzt, welche sich jedoch aufgrund der kurzen Reichweite nur zur Erfassung von Objekten in unmittelbarer Nähe des Automobils eignen [50]. Für weiter entfernte Objekte werden RADAR und LIDAR Sensoren eingesetzt, die eine Reichweite von bis zu 200 Metern abdecken und vorwiegend in Abstandsregeltempomaten (Adaptive Cruise Control) eingesetzt werden. Aus der Sensordatenfusion von aufeinanderfolgenden Abtastungen, der in Abbildung 3.4 dargestellten Erfassungsbereiche, wird die Bewegung und die Entfernung eines Objektes bestimmt.

LIDAR Sensoren sind im Vergleich zum RADAR Sensor in der Automobilbranche weitestgehend noch nicht etabliert, da die Radarwellenausbreitung gegenüber Witterungseinflüssen wie Regen oder Schnee unempfindlicher ist [42]. Die RADAR Technologie nutzt zur Entfernungsmessung

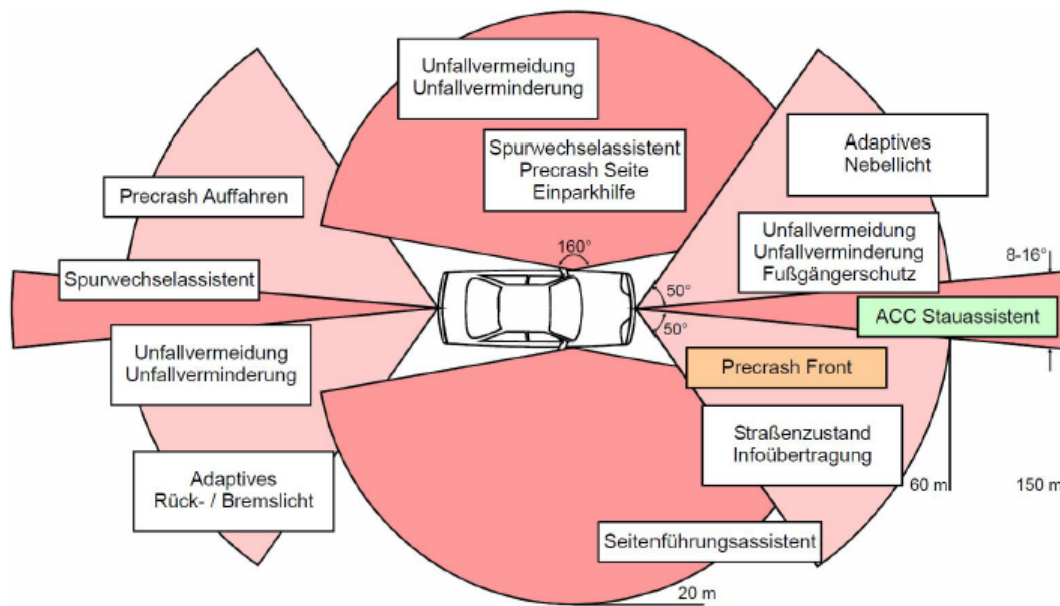


Abbildung 3.4: Anwendungsbereiche von LIDAR Sensorik im Automobil [43]

gebündelte elektromagnetische Wellen im Mikrowellenbereich und arbeiten mit einer Leistung von ca. 10 mW. Die im Automobil eingesetzten Radarsensoren haben somit keine gesundheitlichen Auswirkungen auf Personen, wenn die Richtlinien nach DIN-VDE 0848 Teil 2 eingehalten werden [16].

Die Anwendungsgebiete von Laserscannern beschränken sich nicht nur auf die Automobilbranche. Ebenso werden sie in fahrerlosen Transportsystemen zur Navigation, in der Militärtechnik zur Ortung und in der Sicherheitstechnik eingesetzt. Vor allem das 3D-Laserscanning hat in den letzten Jahren im Bereich der Topographischen Vermessungen an Bedeutung gewonnen. Airborne Laserscanning Systeme werden in Flugzeugen oder Helikoptern eingebaut und erstellen hochauflösenden Geländemodelle mit einer Höhengenaugigkeit von bis zu 10 cm [48]. Ein weiteres Beispiel sind die LKW-Maut Brücken, die mit 3D-Laserscannern Fahrzeuge klassifizieren [12]. Nachfolgend werden LIDAR Sensoren vorgestellt, die in der Automobilbranche zum Einsatz kommen.

Hella IDIS - Adaptive Cruise Control

Die LIDAR-basierte automatische Abstandsregelung von *Hella* (vgl. Abb. 3.5) wird seit 2006 optional im Chrysler 300C eingesetzt [27]. Ein Infrarot-Lichtsensordetektiert vorausfahrende Fahrzeuge und ermittelt mit der Lichtlaufzeitmessung den Abstand sowie die Relativgeschwindigkeit. Der Abstand zum vorausfahrenden Fahrzeug wird in Abhängigkeit zur eigenen Ist-Geschwindigkeit durch die Anpassung der Motorleistung und der Bremskraft konstant gehalten.

Hierfür werden über den CAN-Bus zusätzlich Informationen vom Geschwindigkeitssensor und Lenkwinkelsensor eingelesen.



| Charakterisierung des Hella IDIS [®] | |
|---|---------------|
| Reichweite | 200 m |
| Arbeitsbereich | 150 m |
| Frequenz | 8 Hz (120 ms) |
| Öffnungswinkel | 12° or 16° |
| Auflösung | 0.1 m |
| Genauigkeit | ± 1 % |
| Interface | CAN |

Abbildung 3.5: Hella IDIS "Adaptive Cruise Control" (ACC) [35] mit Betriebskenngrößen [27]

Der Lidarsensor von Hella ist ein Multi-Beam Laserscanner, welcher mit maximal 16 unabhängigen Sende- und Empfangskanälen arbeitet. Über ein Multiplexverfahren werden die einzelnen Laserdioden getrennt angesteuert und das reflektierte Licht über PIN Dioden erfasst. Die empfangenen Rohdaten werden durch eine Vorfilterung auf maximal 48 Objekte beschränkt. Ein nachgelagertes Tracking-Modul beobachtet die Verkehrssituation und bestimmt aus den vorgefilterten Messdaten, abhängig von der Sichtweite und der erkannten Fahrspur, diejenigen Objekte welche eine Gefahrensituation verursachen könnten. Das ACC-System regelt dementsprechend die Motorleistung oder führt ein automatisches Bremsmanöver ein, welches aus Sicherheitsgründen ein Viertel der maximalen Bremskraft aufwenden darf [41].

Laserscanner Ibeo ALASCA XT

Der Multi-Beam Laserscanner erfasst in der Vertikalen über vier parallele Kanäle gleichzeitig bis zu vier reflektierte Lichtstrahlen. Durch das mehrzeilige Scannen hat der ALASCA XT einen vertikalen Öffnungswinkel von 3,2° und kann hiermit den Nickwinkel des Fahrzeuges kompensieren (vgl. Abb. 3.6). Des Weiteren wird durch die Multi-Echo Technologie, die pro Laserpuls vier Echosignale empfängt, eine zuverlässige Objekterkennung gewährleistet, welche nicht durch schlechte Witterung wie Regen oder Schnee beeinflusst wird [20].

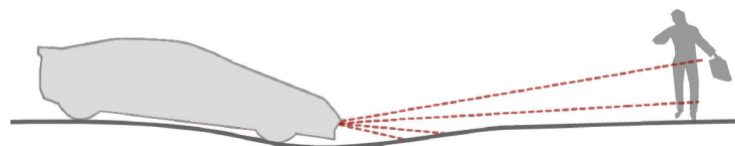
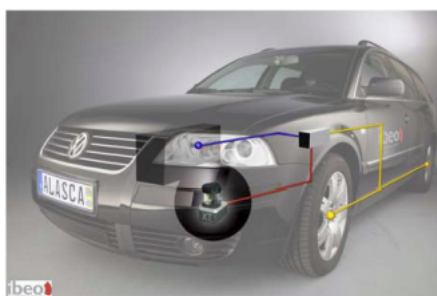


Abbildung 3.6: ALASCA XT Multi-Layer Technologie zur Kompensation des Nickwinkels [29]

Das Umfeld eines Fahrzeuges ist je nach Situation mehr oder weniger relevant. Die resultierenden Messdaten aus dem 240° Scanbereich werden abhängig von der Frequenz, mit einer

konfigurierbaren Winkelauflösung eingeschränkt. Bei einer Frequenz von 25 Hz scannt der ALASCA XT die relevante Bereiche von mit einer Winkelauflösung von 0.25° hingegen werden nicht relevante Bereiche, wie die Fahrzeugseite, mit einer Auflösung von 1° abgescannt. Zur Datenreduktion empfängt die "Electronic Control Unit" (ECU) vor-gefilterte Messdaten und generiert aus den zusammenhängenden Abstandswerten Objekte. Die Positionsverfolgung von sich bewegenden Objekten erfolgt über einen Kalman-Filter. Über den CAN-Bus werden die Objektkoordinaten an den Fahrzeug Computer übertragen, wobei auf Grund der Vielzahl an Sensoren im Automobil, ein allgemeingültiges Koordinatensystem durch die DIN 70000 Norm definiert wurde [15].



| Charakterisierung des ALASCA XT | |
|---------------------------------|---------------------------|
| Reichweite | 0.3 - 200 m |
| Frequenz | 8 - 40 Hz |
| Horizontaler Scanwinkel | 240° |
| Vertikaler Scanwinkel | 3.1° |
| Auflösung | $0.125^\circ - 1.0^\circ$ |
| Genauigkeit | $\pm 1\%$ |
| Interface | ECU CAN |

Abbildung 3.7: Betriebskenngrößen des *Ibeo* ALASCA XT Laserscanners [29]

Liegt eine detaillierte Umfelderkennung vor, kann der ALASCA XT für verschiedene Anwendungen eingesetzt werden. Durch den großen Sichtbereich kann der Laserscanner sowohl für eine automatische Abstandsregelung ACC als auch für Stop-and-Go Systeme genutzt werden. Letztere reduzieren den Kraftstoffverbrauch indem sie den Motor bei bestimmten Bedingungen abschalten. Aufgrund der guten Erkennung im Nahbereich werden zukünftige ACC Systeme über eine Stop-and-Go Funktionalität verfügen [51].

3.2 Prinzipien der Lasertechnologie

Das Wort LASER ist ein Akronym für "Light Amplification by Stimulated Emission of Radiation" und wurde erstmals im Jahre 1959 öffentlich erwähnt [25]. Das Licht eines Lasers wird aufgrund der Einfarbigkeit als monochromatisch bezeichnet und unterscheidet sich von einer normalen Lichtquelle durch die konstante Wellenlänge und dem sehr engen Frequenzspektrum. Des Weiteren erzeugt ein Laser ein paralleles Lichtbündel, bei dem alle Lichtstrahlen in die gleiche Richtung ausgesendet werden. Die einzelnen Wellen des gebündelten Laserstrahls haben nahezu eine identische Phasenlage und sind somit kohärent. Es wird zwischen räumlicher und zeitlicher Kohärenz unterschieden, wobei bei der räumlichen Kohärenz alle Wellen im

Querschnitt eines Laserstrahls phasensynchron sind. Wenn alle Wellen nach einer bestimmten Distanz immer noch in Phase liegen, spricht man von zeitlicher Kohärenz. Für eine weitreichende Entfernungsmessung mit einem Laserscanner wird ein Laser mit einer hohen zeitlichen Kohärenz und einer großen Kohärenzlänge benötigt [31].

Die Entstehung der Laserstrahlen erfolgt mittels stimulierter Emission, bei der ein bestehendes Photon zur Aussendung eines weiteren identischen Photons angeregt wird. Da jedes Photon die gleichen Eigenschaften wie Richtung und Frequenz hat, erfolgt eine Bündelung der Strahlen und somit eine Verstärkung der Strahlung. Ein optischer Resonator speichert die Verstärkung, indem er für eine Rückkopplung des emittierten Lichts durch zwei gegenüberliegende Spiegel sorgt. Die Strahlen werden durch Reflexion mehrfach durch das bei Diodenlasern aus meist Kristall oder Glas bestehende Medium geleitet, wobei die Photonen weitere stimulierte Emissionen erzeugen [17]. Bei Laserscannern werden meist Laserdioden zur Aussendung des Laserstrahls genutzt und das reflektierte Licht wird durch eine Photodiode oder eine Lateraldiode in elektrischen Strom umgewandelt. Laserscanner werden je nach Signalform unterschieden:

- **Pulslaser:** Die Photonen werden periodisch emittiert und zur Erzeugung der Pulse wird im Resonator ein optischer Schalter verwendet, der das Licht nur im geöffneten Zustand durchlässt. So wird mehr Energie freigesetzt und die Intensivität des Laserstrahls erhöht. Gemäß der Fourier-Transformation bestimmt die Dauer eines Pulses die Breite des erzeugten elektromagnetischen Spektrums, sodass je kürzer ein Puls desto breiter sein Spektrum [25]. Femtosekundenlaser sind die leistungsstärksten Laser, die eine Pulsdauer von unter einer Pikosekunde und eine Leistung im Kilowatt Bereich erzielen [18].
- **Dauerstrich-Laser:** Im Gegensatz zum Pulslaser sendet der cw-Laser einen kontinuierlichen Laserstrahl, dessen Wellen monochromatisch und kohärent sind. Die durchschnittliche Leistung von 20 Watt liegt bei cw-Laser aufgrund der kleineren Intensivität unterhalb der Leistung von Pulslasern (ca. 60 Watt) [17].

Die "Light Detection and Ranging" (LIDAR) Technologie ist ein optisches Messverfahren zur räumlichen Messung der Entfernungen von Objekten. Ein periodisch oder kontinuierlich abgestrahlter Laserimpuls wird durch eine rotierende Sendeeinheit oder einen Spiegel in die gewünschte Richtung abgelenkt und ein ebenfalls rotierender Empfänger erfasst den vom Hindernis reflektierten Laserstrahl. Laserscanner verwenden zur Entfernungsmessung je nach Anwendungsgebiet verschiedene Verfahren [42, 10, 51]:

- **Laufzeitmessung:** Bei der "Time of Flight" Messung wird ein oder mehrere Laserstrahlen in kurzen Impulsen ausgesendet und von einem Hindernis reflektiert. Die gemessene

Laufzeit des Laserstrahls repräsentiert die doppelte Distanz zum Hindernis. In der Automobilelektronik wird das "Time of Flight" Verfahren aufgrund der geringen Reaktionszeit und der großen Reichweite zur Hinderniserkennung eingesetzt [27]. Jedoch ist wegen der erforderlichen Messung von geringen Zeiten eine hohe Abtastfrequenz der internen Zähler erforderlich. Die Entfernungsauflösung ist abhängig von der Zählfrequenz des Laserscanners und liegt bei diesem Verfahren meist im unteren Zentimeterbereich. Das "Time of Flight" Messverfahren wird ebenfalls bei TOF Kameras zur dreidimensionalen Objekterfassung eingesetzt, wobei für jeden Bildpunkt die Distanz zum Objekt berechnet wird. Der Vorteil von TOF Kameras ist, dass keine Abtastung stattfindet, sondern der gesamte Sichtbereich aufgenommen wird [26].

- **Phasenverschiebung:** Die Amplitude eines kontinuierlich ausgestrahlten Laserstrahls wird mit mehreren sinusförmigen Schwingungen moduliert und durch den Vergleich der Phasenlage des gesendeten und des empfangenen Signals wird die von der Entfernung abhängige Phasendifferenz bestimmt. Im Vergleich zur Laufzeitmessung kann mit diesem Verfahren aufgrund des geringeren messtechnischen Aufwands, da die internen Zähler nicht so hoch getaktet werden müssen, eine höhere Auflösung im Millimeterbereich erreicht werden. Jedoch ist bei größeren Distanzen eine Eindeutigkeit der Signale nicht gegeben.
- **Triangulation:** Bei der Lasertriangulation wird ein Laserstrahl auf ein Messobjekt gerichtet und das reflektierte Licht mit einem CCD-Sensor aufgenommen. Durch trigonometrische Winkelfunktionen wird abhängig vom Einstrahlwinkel die Entfernung zum Objekt berechnet. Tritt eine Änderung der Position ein, so wird auch der Einstrahlwinkel verändert. Die Lasertriangulation eignet sich aufgrund des rein mathematischen Verfahrens und den konstanten Vorgaben nur für geringe Entfernungen. Vor allem in 3D-Sensoren zur Objekterkennung oder zur Kartenbildung werden Triangulationsverfahren eingesetzt [19].

3.3 Analyse und Bewertung der Genauigkeit des Laserscanners

Bei der Erkennung von Objekten ohne Reflektoren ist die Intensität des zurückgesendeten Laserstrahls entscheidend für die Entfernungsmessung. Der Laserscanner URG-04LX berechnet aus der Phasendifferenz des ausgesendeten und empfangenen Laserstrahls direkt die Abstandswerte, wobei die Genauigkeit abhängig von den Reflexionseigenschaften des Objekts und des

Auftreffwinkels des Laserstrahls ist. Anhand von Testmessungen mit Objekten in variablen Abständen wurde die Genauigkeit des Laserscanners in Bezug auf die Entfernungsmessung analysiert. Da das Datenblatt hierzu keine Aussage trifft, wurden die Testergebnisse mit IEEE Veröffentlichungen verglichen. In der Arbeit von [39] wurde der Hokuyo URG-04LX Laserscanner unter Laborbedingungen geprüft. Bis zum Erreichen eines Gleichgewichts zwischen Stromverbrauch und Wärmeableitung steigt die interne Temperatur des Laserscanners während der Aufwärmphase stetig an (Abbildung 3.8).

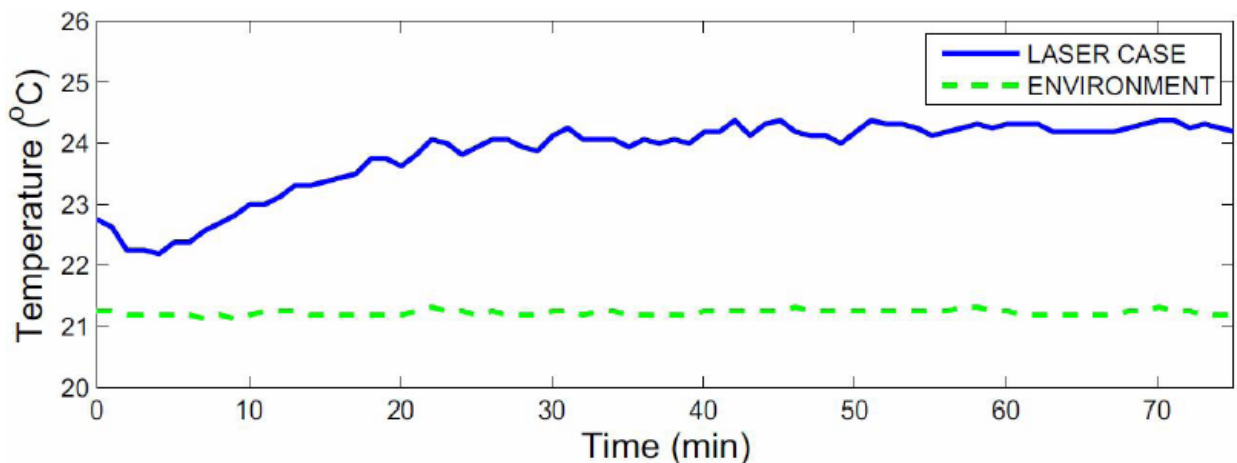


Abbildung 3.8: Anstieg der Laserscanner Temperatur während der Aufwärmphase [39].

Die sogenannte "Warm-Up Time" beträgt zwischen 30 und 40 Minuten. Nach dieser Zeit wird der stationäre Zustand erreicht und die Entfernungswerte sind nahezu konstant. Um die Reflexionseigenschaften von Materialien zu analysieren, wurden Objekte mit variierenden Farben in einer Entfernung von 500 und 1000 mm platziert. In 30 Minuten Intervallen wurden jeweils 50 Testmessungen für den Scanschritt 384 (Winkel 0°) angefordert. In Tabelle 3.1 wird das arithmetische Mittel \bar{x} der Einzelmessungen und die Standardabweichung σ gezeigt, die angibt wie weit die einzelnen Abstandswerte um den Mittelwert streuen. Die angegebene Abweichung Δ wird gebildet aus der Differenz zwischen Mittelwert und tatsächlicher Entfernung. Da sich die Laserdiode in einem Abstand von 1 cm zur Außenwand befindet, wurde dieser Versatz von jedem dekodierten Abstandswert subtrahiert.

Während der Aufwärmphase von 30 Minuten variieren die Abstandswerte stark, da die Abweichungen nicht im angegebenen Toleranzbereich des Laserscanners liegen (vgl. Abb. 3.1). Der Grund hierfür, ist auf den Anstieg der internen Sensor Temperatur zurückzuführen, welche aufgrund der Wärmeabgabe des Scankopfmotors steigt. Bei einer Entfernung von 50 cm beträgt die Abweichung für ein weißes Objekt 2 cm, was sowohl an der nicht garantierten Abstandsmessung bis zu 60 cm als auch am Anstieg der Temperatur liegt. Bei einer Entfernung von 1,5

| Oberflächen- farbe | Entfernungsmessung mit Abständen von 500 mm / 1000 mm | | | | | | | | | | |
|-----------------------|---|-----------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| | | Kaltstart | | 30 min | | 45 min | | 60 min | | 90 min | |
| Weiß | \bar{x} | 520,2 | 1030,2 | 510,9 | 1022,8 | 506,9 | 1019,0 | 508,7 | 1017,7 | 506,4 | 1017,4 |
| | Δ | 20,2 | 30,2 | 10,9 | 22,8 | 6,9 | 19,0 | 8,7 | 17,7 | 6,4 | 17,4 |
| | σ | 2,82 | 3,81 | 3,04 | 2,08 | 2,23 | 2,55 | 3,02 | 2,53 | 2,35 | 2,67 |
| Schwarz | \bar{x} | 453,4 | 983,3 | 455,9 | 983,2 | 454,7 | 984,5 | 453,5 | 984,6 | 452,4 | 983,8 |
| | Δ | -46,6 | -16,7 | -44,1 | -16,8 | -45,3 | -15,5 | -46,5 | -15,4 | -47,6 | -16,2 |
| | σ | 4,44 | 2,48 | 2,99 | 3,26 | 2,95 | 2,21 | 2,73 | 2,60 | 2,76 | 2,37 |
| Grau | \bar{x} | 519,1 | 961,5 | 510,2 | 976,7 | 505,2 | 975,2 | 502,9 | 974,9 | 503,7 | 973,8 |
| | Δ | 19,1 | -38,5 | 10,2 | -23,3 | 5,2 | -24,8 | 2,9 | -25,1 | 3,7 | -26,2 |
| | σ | 2,38 | 1,99 | 2,47 | 2,47 | 2,87 | 2,32 | 2,79 | 2,80 | 2,51 | 2,67 |
| Hellbraun | \bar{x} | 513,4 | 1011,1 | 511,3 | 1006,0 | 502,4 | 1003,3 | 499,6 | 1006,6 | 500,1 | 1005,2 |
| | Δ | 13,4 | 11,1 | 11,3 | 6,0 | 2,4 | 3,3 | -0,4 | 6,6 | 0,1 | 5,2 |
| | σ | 2,24 | 2,67 | 2,95 | 2,94 | 2,26 | 3,26 | 2,47 | 2,59 | 2,05 | 2,65 |

Tabelle 3.1: Testmessung von unterschiedlichen Objekten bei zwei Entfernungen von 500mm und 1000mm. Berechnung des arithmetischen Mittelwerts \bar{x} und der Standardabweichung σ aus 50 Messungen. Die Abweichung Δ berechnet sich aus der absoluten und gemessenen Entfernung [49].

Metern wurde innerhalb von 30.000 Messungen eine konstante Abweichung von mindestens 1 cm während den ersten 30 Minuten festgestellt [39].

Die Testergebnisse zeigen für die Standardabweichung ein Überschreiten des angegebenen Toleranzbereiches von 1% und dass die Abstandswerte bei einem weißen Hindernis geringere Abweichungen haben als bei einem schwarzen Objekt. Des Weiteren ist die gemessene Entfernung bei einem weißen Objekt größer als die tatsächliche Entfernung. Bei schwarzen Hindernissen sind die gemessenen Entfernungen dagegen kleiner, was darauf zurückzuführen ist, dass die Lichtintensität aufgrund der höheren Lichtabsorption von schwarzen Oberflächen sinkt. Helle Objekte zeigen eine stärkere Beeinflussung durch die "Warm-Up Time", da während dieser Zeit eine Annäherung an die tatsächliche Entfernung stattfindet.

4 Der ARMv7 Cortex-A9 Zweikern-Prozessor

Die Software-Partitionierung und Thread-Synchronisierung auf der OMAP4430-Plattform richtet sich nach den Funktionsblöcken des Zweikern-Prozessors, die in diesem Kapitel beschrieben werden. Die Serie der *Cortex*-Prozessoren gehören zur siebten Version der ARM-Prozessorarchitektur (ARMv7 [4]). Als *Architektur* definiert ARM das für den Anwender sichtbare Verhalten des Prozessors, welches unter anderem die Menge der Assemblerbefehlssätze (instruction sets), Register und das Reagieren auf Fehler oder Interrupts (exception handling) beinhaltet. Diese Architekturversion wird vom Hersteller in drei *Profile* unterteilt, welches durch den Buchstaben in der Prozessorbezeichnung identifiziert wird. Das *Application*-Profil zielt auf Rechenleistung (Instruktionsdurchsatz) ab und bietet eine "Memory Management Unit" (MMU), die Multitasking-Betriebssystemen eine virtuelle Speicherverwaltung zur Verfügung stellt. Daneben verfügbar sind das *Real-time*-Profil mit dem Ziel auf kurze Interrupt-Latenzzeiten sowie das *Microcontroller*-Profil, deren Prozessoren sich durch geringen Stromverbrauch auszeichnen. Folgende Eigenschaften sind für die Cortex-A Prozessoren identisch:

- 32-bit "Reduced Instruction Set Computer" (RISC) mit 16 x 32-Bit Register, deren Zugriff und Sichtbarkeit abhängig vom Prozessormodus ist.
- Getrennter Zugriff auf 1st-Level Daten- und Instruktionsspeicher (*Pseudo* Harvard-Architektur).
- 8-bit, 16-bit oder 32-bit (unaligned) Daten-Speicherzugriff, der exklusiv durch Lade- und Speicherbefehle erfolgt (Load/Store Architektur).
- Die Länge der Instruktionen beträgt wahlweise 32-Bit (ARM state) oder 16-Bit (Thumb state).
- "Vector Floating Point" (VFP) Einheit zur Berechnung von arithmetischen Operationen auf Gleitkommazahlen mit einfacher und doppelter Genauigkeit nach IEEE 754.

- "Single Instruction Multiple Data" (SIMD) Instruktionssatz (*NEON*) zur parallelen Ausführung einer Operation auf bis zu 16 Daten mit einer Instruktion, die zur Rechenbeschleunigung bei der Bildverarbeitung beitragen.
- 4GB physikalischer und virtueller Adressraum mit "Memory Management Unit" (MMU) zur Übersetzung der virtuellen in physikalische Adressen.
- Big- und little-endian Unterstützung für die Byte-Reihenfolge im Speicher.
- "Symmetrische Multiprozessorsystem" (SMP) Unterstützung (*Snoop Control Unit*) bei den Mehrkern-Varianten, zur Erhaltung der 1st-Level Cache-Kohärenz

4.1 Prozessorarchitektur mit Interprozessor-Interfaces

Ein Cortex-A9 Prozessorkern besteht aus einer *out-of-order*, *superscalar* und *speculative* Instruktionpipeline (Abbildung 4.1). In der Prefetch-Stufe werden die geladenen Instruktionen in einer

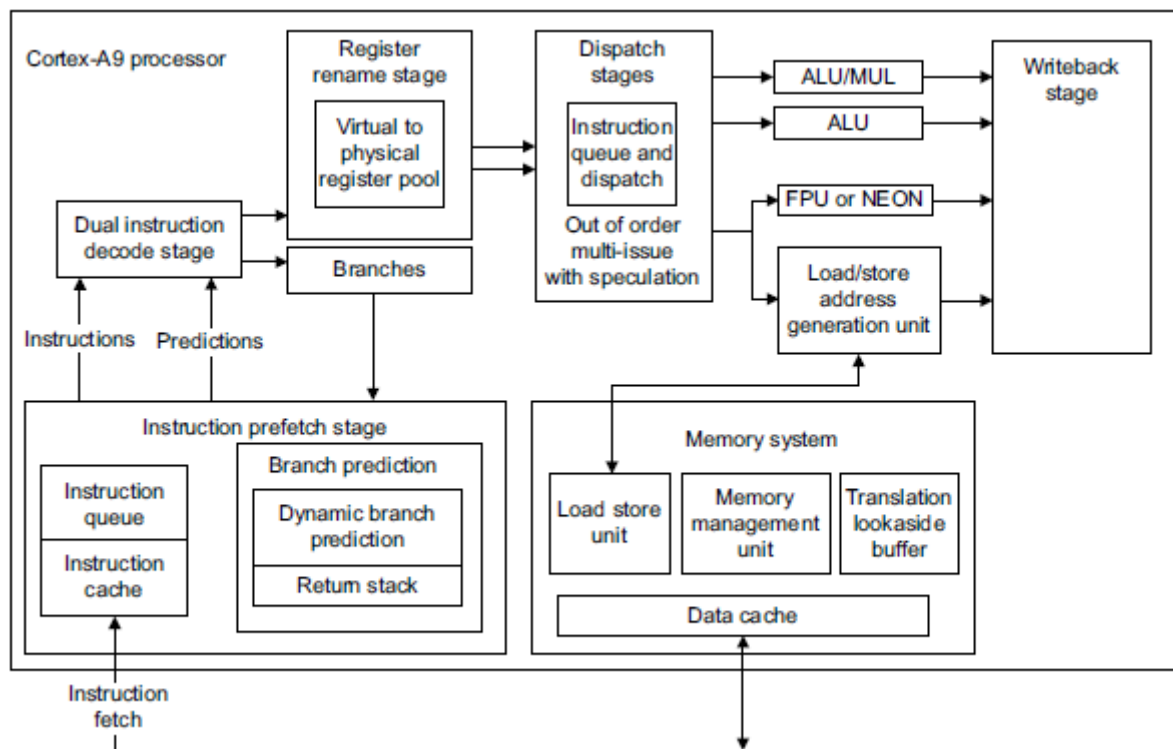


Abbildung 4.1: Top-Level-Diagramm mit den Pipelinestufen (*stages*) des Cortex-A9 Kerns [3].

Warteschlange (*instruction queue*) zwischengespeichert. Liegt ein oder mehrere Operanden für eine Instruktion noch nicht vor, wird eine andere Instruktion, deren Operanden bereits

verfügbar sind, aus der Warteschlange weiterverarbeitet. Die Logik der Prefetch-Stufe prüft vor dieser Modifikation der Instruktionsreihenfolge anhand der Ziel- und Quelloperanden, ob Abhängigkeiten zwischen den vertauschten Instruktionen bestehen und eine *out-of-order* Ausführung zulässig ist. Eine weitere Warteschlange bindet sich in der Dispatch-Stufe, in der unabhängige arithmetische Instruktionen zur parallelen Ausführung an die arithmetischen- und Gleitkommaeinheiten, (*execution units*) verteilt werden. Die Ergebnisse für Operationen auf Register werden in der Writeback-Stufe auf diese zurück geschrieben. Bei Instruktionen, die den Kontrollfluss ändern, wie (un-)bedingte Sprunganweisungen oder arithmetische Operationen mit dem Programmzähler als Zielregister, ist in der Prefetch-Stufe nicht vorhersagbar, ob die Ausführung dieser Anweisung gültig ist und an welcher Zieladresse die nächsten Instruktionen geladen werden. Um ein Anhalten der Prefetch-Stufe bis zur Beendigung der Kontrollfluss-Instruktion zu vermeiden, werden diese Instruktionen nach dem Durchlaufen der Decode-Stufe an die *Branch prediction* Einheit zurückgeliefert. Wird die gleiche Instruktion erneut geladen, verhält sich die Pipeline vorläufig (*spekulative*) gemäß des letzten Auftretens dieser Kontrollfluss-Instruktion. Nur im Falle einer falschen Vorhersage werden die bisher verarbeiteten Instruktionen in der Pipeline als ungültig markiert und an der korrekten Zieladresse fortgefahren.

In der Register-*rename*-Stufe werden die in der Instruktion adressierten Prozessorregister in Abhängigkeit des aktuellen Prozessormodus auf physikalische Register abgebildet. Ein Cortex-A9-Kern verfügt über sieben Prozessormodi:

- Supervisor (**SVC**) - Aktiviert nach einem Reset oder dem Ausführen einer SVC-Instruktion.
- **FIQ** - Aktiviert nach einer *fast interrupt exception*.
- **IRQ** - Aktiviert nach einer *normal interrupt exception*.
- Abort (**ABT**) - Aktiviert nach einem Zugriff auf eine ungültige Speicheradresse.
- Undef (**UND**) - Aktiviert nach dem Decodieren einer Instruktion mit undefinierten Opcode.
- User (**USR**) - Standardmodus mit eingeschränkten Ausführungsrechten.
- System(**SYS**) - Entspricht dem User-Modus mit uneingeschränkten Ausführungsrechten.

Im User-Modus hat der Anwender Zugriff auf 16 x 32-Bit Register, die mit R0 bis R15 bezeichnet werden. Die ARM-Konvention sieht vor, dass abweichend von der *general purpose* Nutzung die Register R12 bis R15 für bestimmte Zwecke verwendet werden: Das Register R12 enthält bei Unterprogrammaufrufen den *frame pointer* auf den ersten Funktionsparameter im Stack

und wird in Assembler-Instruktionen auch als FP bezeichnet. R13 enthält den *stack pointer* (SP), der auf das aktuelle Ende des Stacks zeigt. R14 enthält bei Unterprogrammaufrufen die Rücksprungadresse und wird als *link register* (LR) bezeichnet. Bei R15 handelt es sich um den *program counter* (PC), in dem die Adresse der als nächstes auszuführenden Instruktion steht. In allen Prozessormodi sind R0 bis R12 sowie der PC identisch. Dagegen verfügt jeder Modus jeweils über ein eigenes SP und LR Register, auf die ausschließlich im entsprechenden Modus zugegriffen werden kann. Ausnahme ist der FIQ-Modus, der zusätzlich R8 bis R12 des User-Modus ausblendet und durch eigene Register ersetzt. Zur Steuerung und (De-)Aktivierung der Prozessor-Funktionen dienen 32-Bit Kontrollregister des internen 15. Coprozessors (CP15), auf die im User-Modus nicht zugegriffen werden kann.

Der im Rahmen der Masterarbeit eingesetzte Cortex-A9 ist mit zwei Prozessorkernen konfiguriert (Abbildung 4.2), von denen jeder über einen 4-Wege assoziativen 32KB Instruktions- und 32KB Datencache sowie über einen privaten Timer, Watchdog und eigenes Power Management verfügt. Weiterhin existieren ein globaler Timer, Interrupt- und 2nd-Level Cache-Controller, die von beiden Kernen gemeinsam genutzt werden.

Interprozessor-Kommunikation und -Synchronisation erfolgen aus Sicht des Anwenders über Lese- und Schreibzugriffe auf den gemeinsamen Speicher. Die "Snoop Control Unit" (SCU) erhält dabei die L1-Cache-Kohärenz und trägt somit zur Steigerung der Datentransferrate zwischen den Prozessorkernen bei. Dazu ist im Cortex-A9 das "Modified Exclusive Shared Invalid" (MESI) Protokoll implementiert: Jeder 128-Bit langen Cache-Line werden zwei Statusbits zugeordnet, die einen der folgenden vier Zustände kennzeichnet [2]:

- **Modified** - Diese Cache-Line wurde lokal geändert, die im Hauptspeicher befindliche Kopie ist ungültig und der andere Kern enthält keine Version dieser Cache-Line.
- **Exclusive** - Dieser Cache ist der einzige, der diese Cache-Line enthält und ihr Wert ist kohärent mit dem Hauptspeicher.
- **Shared** - Beide Caches enthalten diese Cache-Line und ihr Wert ist kohärent mit dem Hauptspeicher. Ein Cache-Line wird immer dann als *Shared* markiert nachdem ein Kern lesend auf eine Cache-Line zugreift, die als *Modified* oder *Exclusive* im anderen Kern vorhanden war. Ein Schreibzugriff eines Kern verändert den Status der Line beim schreibenden Kern in *Modified* und in *Invalid* beim anderen.
- **Invalid** - Der Inhalt dieser Cache-Line ist nicht mehr aktuell und somit ungültig.

Ohne aktivierte SCU würde ein Datenwert, der zwischen den Kernen ausgetauscht werden soll, vom Datencache des Senders durch die Speicherhierarchie über den Hauptspeicher in

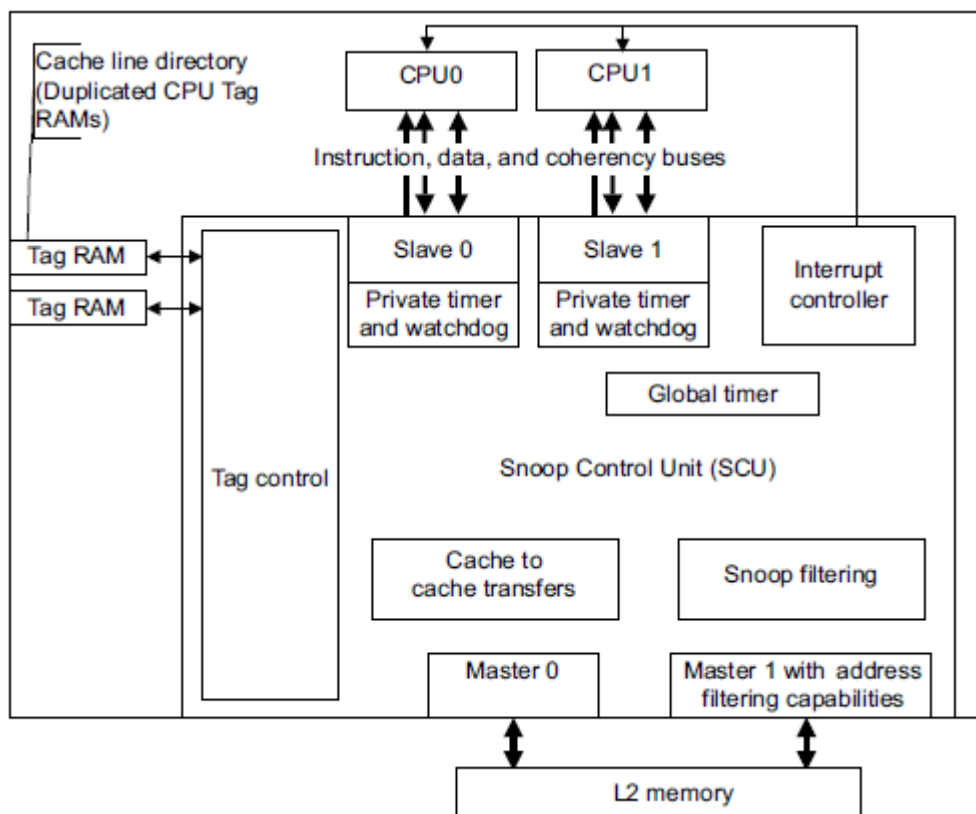


Abbildung 4.2: Top-Level-Diagramm des Cortex-A9 DualCore Prozessors mit den Funktionsblöcken der "Snoop Control Unit" (SCU) [2]

den Datencache des Empfängers gelangen. Mit aktivierter SCU überwacht diese die Cache-Line Zustandsbits beider Kerne (*bus snooping*) und stellt sicher, dass der Datenwert innerhalb des Prozessors vom Sender-Cache direkt in den Empfänger-Cache übertragen wird. Dieser Überwachungsmechanismus findet ausschließlich bei Speicherzugriffen statt, die durch die Ausführung von *exklusiven* Load- (LDREX) und Store- (STREX) Instruktionen initiiert werden, während die standardmäßigen LDR- und STR- Speicherzugriffe von der SCU ignoriert werden.

4.2 Memory Management Unit für virtuelle Speicherverwaltung

Zur Partitionierung von Multi-Tasking Anwendungen in den physikalischen Speicher des eingebetteten Systems dient die "Memory Management Unit" (MMU). Dabei erhält jeder Prozess einen eigenen virtuellen 4GB großen Adressraum, den die MMU-Hardware auf Bereiche des physikalischen Speichers abbildet (Abbildung 4.3). Der Vorteil des virtuellen Adressraumes

liegt darin, dass bei der Entwicklung einer Anwendung nicht auf die Speicheraufteilung des Endsystems geachtet werden muss. Bei einer deaktivierten MMU wird jede virtuelle Adresse eins-zu-eins in die gleiche physikalische Adresse übersetzt (*flat mapping*). Darüber hinaus ist eine aktivierte MMU für die SCU sowie zum Aufwecken des zweiten Prozessorkerns erforderlich [44].

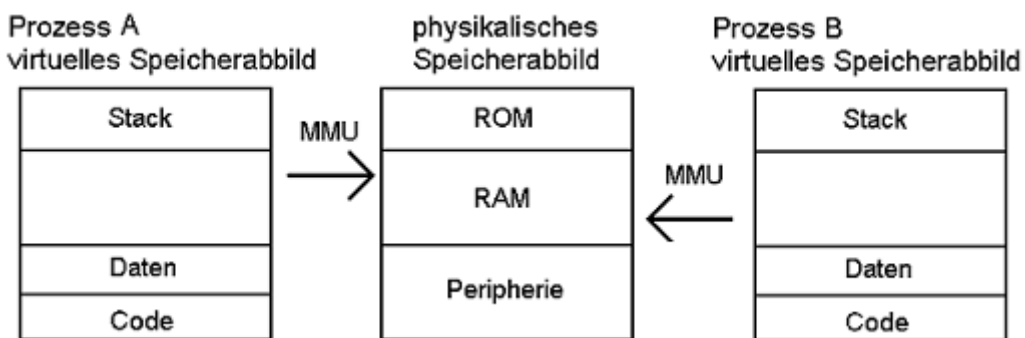


Abbildung 4.3: Virtuelle Speicherabbildung von zwei Prozessen in den physikalischen Speicher unter Verwendung der MMU.

Für die Adressübersetzung wird der 4GB große Adressraum in 4096 gleichgroße 1MB Sektionen aufgeteilt. Eine Seitentabelle (*level 1 page table*) enthält dazu 4096 x 32-Bit Einträge, von denen jeder entweder die Anfangsadresse einer 1MB-Sektion im physikalischen Speicher enthält oder auf eine weitere Seitentabelle (*level 2 page table*) verweist, welche die 1 MB-Sektion in kleinere Sektionen aufteilt. Die Position der L1-Seitentabelle befindet sich im "Translation Table Base Register" (TTBR), ein CP15-Kontrollregister. Führt ein Prozess einen Speicherzugriff auf eine virtuelle Adresse durch (Abbildung 4.4), werden die 12 hochwertigsten Bits [31:20] der Adresse auf die Anfangsadresse der L1-Seitentabelle addiert. Das Resultat ist die Adresse (First Level Descriptor Address) des korrespondierenden Eintrages. Die MMU liest die 12 höchstwertigen Bits des Eintrages und bildet daraus zusammen mit den Bits [19:0] der virtuellen Adresse die physikalische Adresse. Die beiden niederwertigsten Bits [1:0] kennzeichnen den Typ des Eintrages:

- **00** - Eintrag für nicht-abgebildete virtuelle Adressen, deren Zugriff eine Abort-Exception generiert.
- **01** - Eintrag enthält die Anfangsadresse einer L2-Seiten, welche die 1MB-Sektion weiter unterteilt.
- **10** - Eintrag enthält die Anfangsadresse einer 1MB-Sektion im physikalischen Speicher.
- **11** - Reserviert.

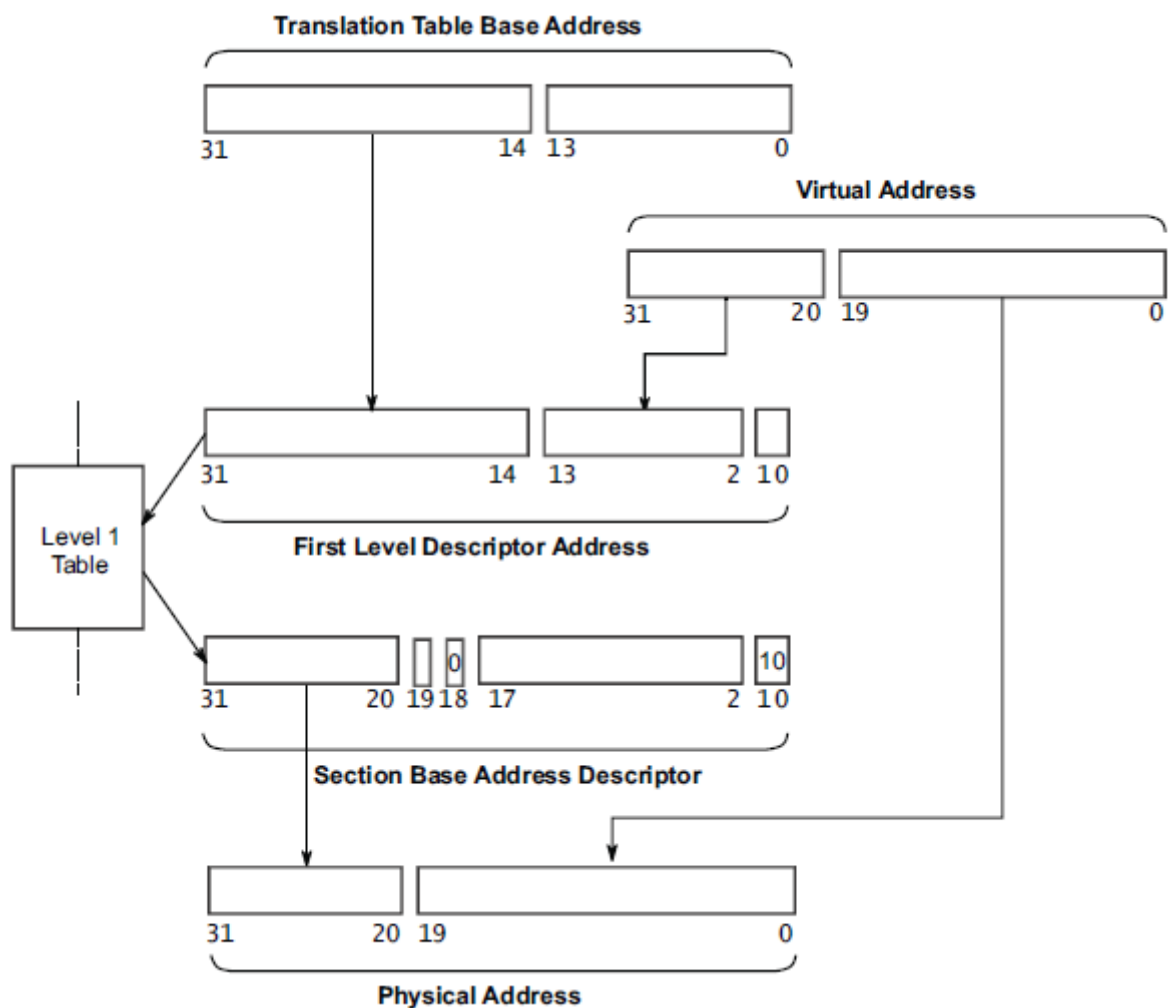


Abbildung 4.4: Übersetzungsvorgang von virtueller in eine physikalische Speicheradresse [5]

Bei einem Verweis auf eine L2-Seitentabelle werden Bits [19:12] der virtuellen Adresse als Offset zur Auswahl des L2-Eintrages verwendet. Für einen Speicherzugriff einer Anwendung wird somit bis zu dreimal auf den Speicher zugegriffen: Einmal auf die L1-Seitentabelle, einmal auf die L2-Seitentabelle und anschließend mit der übersetzten Adresse auf die Daten. Dabei dient der "Translation Lookaside Buffer" (TLB) als Cache für die MMU zur Reduzierung der Speicherzugriffe bei der Adressübersetzung. Die Bits der Seitentabellen-Einträge (Abbildung 4.5), die nicht an der Adressbildung beteiligt sind, legen fest welche Zugriffsrechte, Cache-Strategien und Sicherheitsattribute für die Speicherregion gelten:

- **Non-secure bit (NS)** - Kennzeichnet die Zugehörigkeit des Speicherbereichs in "Secure" (NS=0) oder "Normal" (NS=1). Bei Verwendung der *TrustZone* Sicherheitserweiterung, unterteilt dieses Bit den physikalischen Speicherbereich in zwei separate Adressräume und dient dazu, dass auf memory-mapped Ressourcen, die als "Secure" deklariert sind,

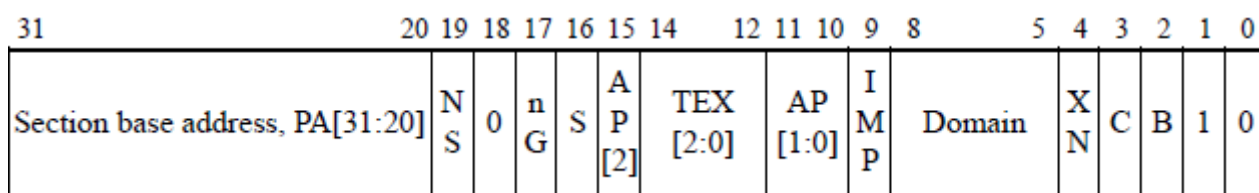


Abbildung 4.5: Format eines L1-Seitentabellen Eintrages [4]

nicht ohne Ausführen einer "Secure Monitor Call" (SMC) Instruktion zugegriffen werden kann.

- **not global bit (nG)** - Kennzeichnet die Zugehörigkeit des Speicherbereichs in "Global" (nG=0) oder "Prozess-spezifisch" (nG=1). Während für einen globalen Speicherbereich eine Adressübersetzung einzigartig ist, können für Prozesse unabhängige Adressübersetzungen parallel existieren. Die Kennzeichnung unterstützt die MMU bei der Auswahl welche Übersetzungen im TLB bleiben bzw. ersetzt werden.
- **Shareable bit (S)** - Wird von der SCU zur Erhaltung der L1-Cache Kohärenz genutzt. Die Inhalte von "Shareable" (S=1) Speicherregionen sind für alle Prozessorkerne identisch.
- **Access permissions bits (AP[2], AP[1:0])** - Kennzeichnen welche Zugriffsformen in den privilegierten oder im unprivilegierten User-Mode auf diesen Speicherbereich zulässig sind und welche zur Auslösung eines Zugriffsfehlers führen (Abbildung 4.6).

| APX | AP | Privileged | Unprivileged | Description |
|-----|----|------------|--------------|------------------------|
| 0 | 00 | No Access | No Access | Permission Fault |
| 0 | 01 | Read/Write | No Access | Privileged Access only |
| 0 | 10 | Read/Write | Read | No user mode write |
| 0 | 11 | Read/Write | Read/Write | Full Access |
| 1 | 00 | - | - | Reserved |
| 1 | 01 | Read | No Access | Privileged Read Only |
| 1 | 10 | read | Read | Read Only |
| 1 | 11 | - | - | Reserved |

Abbildung 4.6: Bit-Codierung der Zugriffsrechte in einem L1-Seitentabellen Eintrag [5]

- **Memory region attribute bits (TEX[2:0], C, B)** - Diese Bits kennzeichnen den Speichertyp sowie die angewendete Cache-Strategie für den Speicherbereich und werden im

nächsten Unterabschnitt "Instruktionsreihenfolge und Speicherbarrieren" ausführlich behandelt.

- **Implementation defined bit (IMP)** - Kann vom Anwender frei genutzt werden.
- **Domain bits** - Ordnet dem Speicher eine von 16 IDs zu. Für einen Prozess, der einer Domain-ID über das "Domain Access Control Register" (DACR), einem CP15-Kontrollregister, zugeordnet ist, ist es zulässig Speicheroperationen auszuführen, denen die Domain-ID als Parameter übergeben wird und die sich anschließend auf alle Speicherbereiche der entsprechenden Domain auswirken.
- **execute-never bit (NX)** - Aus Speicherbereichen mit gesetztem NX-Bit werden von der Prefetch-Pipelinestufe keine Instruktionen geladen. Dieser Mechanismus dient als Schutz vor der Ausführung von kompromittierten Daten.

4.3 Speicherbarrieren und Reihenfolge von Assembler-Instruktionen

Zur Steigerung des Instruktionsdurchsatzes werden die Assembler-Instruktionen vom Cortex-A9 nicht notwendigerweise in der Reihenfolge ausgeführt, in der sie im Instruktionsspeicher liegen. Betrachtet man beispielsweise die folgenden drei Instruktionen (Abbildung 4.7). Die erste

| | | |
|----------------|---|----------------|
| LDR RO, [R1] | → | LDR RO, [R1] |
| ADD RO, RO, #1 | | ADD R8, R8, #1 |
| ADD R8, R8, #1 | | ADD RO, RO, #1 |

Abbildung 4.7: Beispiel zur Umsortierung der Assembler-Instruktionen durch den Cortex-A9

Instruktion lädt das Register Ro mit dem Speicherinhalt, der sich an der Adresse befindet, die im Register R1 steht. In der zweiten Instruktion wird der Inhalt des Registers Ro anschließend um den Wert 1 inkrementiert und in der letzten Instruktion wird die gleiche Operation auf dem Register R8 durchgeführt. Befindet sich der Speicherblock zum Zeitpunkt des Lade-Zugriffs nicht im Cache, müsste der Prozessor warten bis der angeforderte Speicherinhalt aus dem Hauptspeicher geladen wurde, bevor mit der Ausführung der zweiten Instruktion fortgefahren werden kann. Der Prozessor darf die Reihenfolge der ersten beiden Instruktionen aufgrund ihrer Abhängigkeit nicht vertauschen, wohl aber die letzten beiden Instruktionen, da dort keine Abhängigkeiten bestehen. Auf diese Weise wird ein Teil der Wartezeit zur Ausführung unabhängiger Instruktionen genutzt.

Bei Speicherzugriffen auf memory-mapped I/O, CP15-Kontrollregister oder für die Ausführung von selbst-modifizierenden Code sind Umsortierungen unter Umständen nicht erwünscht. Zu diesem Zweck existieren drei Varianten vom Speichertypen: "Strongly-ordered", "Device" und "Normal". Instruktionen, die auf "Normal" (*weakly-ordered*) Speicherbereiche zugreifen, können vom Prozessor umsorrtiert werden. Bei den anderen beiden Typen garantiert der Prozessor die Ausführung in der Reihenfolge, in der sie im Instruktionsspeicher zueinander angeordnet sind. Der Unterschied zwischen "Strongly-ordered" und "Device" Speicherbereichen besteht darin, dass mit der Ausführung der nachfolgenden Instruktionen nach Schreibzugriffen auf "Strongly-ordered" Speicherbereiche bereits fortgefahren werden kann, bevor deren Datum den Hauptspeicher erreicht hat. Dagegen wird mit der Ausführung von nachfolgenden Instruktionen bei Schreibzugriffen auf "Device" Speicherbereiche erst dann fortgefahren, nachdem diese vollständig durchgeführt wurden. Die Zuordnung der Speichertypen erfolgt durch die MMU und wird zusammen mit der Cache-Strategie über die *Memory region attribute bits* ($TEX[2:0]$, C, B) gesteuert (Abbildung 4.8). "Strongly-ordered" und "Device" Speicherbereiche werden

| TEX | C | B | Description | Memory Type |
|-----|---|---|---|------------------|
| 000 | 0 | 0 | Strongly-ordered | Strongly-ordered |
| 000 | 0 | 1 | Sharable Device | Device |
| 000 | 1 | 0 | Outer and Inner Writethrough, no allocate on write | Normal |
| 000 | 1 | 1 | Outer and Inner Writeback, no allocate on write | Normal |
| 001 | 0 | 0 | Outer and Inner Non-cacheable | Normal |
| 001 | - | - | Reserved | - |
| 010 | 0 | 0 | Non-shareable device | Device |
| 010 | - | - | Reserved | - |
| 011 | - | - | Reserved | - |
| 1XX | Y | Y | Cached memory XX = Outer Policy YY = Inner Policy | Normal |

Abbildung 4.8: Bit-Codierung der Speicherattribute für Cache-Strategie und Speichertyp

niemals im Cache zwischengespeichert. Für "Normal" Speicherbereiche wird zwischen "Inner" (Level 1) und "Outer" (Level 2) Caches unterschieden. Für das Cache-Schreibverhalten sind zwei Strategien vorgesehen: Beim "Write-through" wird jeder Schreibzugriff auf den Cache und den Hauptspeicher ausgeführt, so dass diese Kohärent bleiben. Schreibzugriffe auf "Writeback"

Speicherbereiche werden zunächst ausschließlich auf den Cache ausgeführt und die Cache-Line mit den neuen Daten wird zu einem späteren Zeitpunkt in den Hauptspeicher bzw. in den Cache der nächsten Stufe geschrieben, wenn diese durch eine andere Cache-Line ersetzt wird oder der Cache explizit dazu durch *cache-flush* Operationen angewiesen wird.

Zur Synchronisation bei der (parallelen) Anwendungsausführung auf mehreren Prozessorkernen existieren drei Speicherbarrieren, um die keine Instruktion-Umsortierung stattfindet und die jeweils durch eine entsprechende ARMv7-Assembler-Instruktion aufgerufen werden:

- **DMB** (Data Memory Barrier) - Diese Instruktion garantiert, dass sämtliche Speicherzugriffe eines Kerns, die vor dem Aufruf dieser Barriere stehen, vor den Speicherzugriffen nach dem Aufruf, aus Sicht des anderen Kerns auch in dieser Reihenfolge beobachtet werden.
- **DSB** (Data Synchronization Barrier) - Entspricht der DMB und erzwingt zusätzlich das Anhalten der Pipeline des ausführenden Kerns bis alle Speicherzugriffe, die vor dieser Barriere stehen, die externe Speicherhierarchie durchlaufen haben und somit vollständig abgeschlossen sind.
- **ISB** (Instruction Synchronization Barrier) - Diese Instruktion führt zum Abarbeiten aller Instruktionen, die sich in der Pipeline des ausführenden Kerns befinden, so dass alle nachfolgenden Instruktionen neu aus dem Cache oder Hauptspeicher geladen werden. Sie findet Anwendung insbesondere zum Thread-Kontextwechsel auf einem Prozessorkern.

Zur Demonstration folgt eine Implementierung (Abbildung 4.9) eines Mutex, den ein Kern auf der OMAP4430-Plattform zum exklusiven Zugriff auf eine speichergebundene Ressource nutzt [24]. In dieser Implementierung befindet sich die Speicheradresse an der sich Mutex befindet in

```

1 lock_mutex:           ;Sprunglabel
2 LDREX r1, [r0]       ;Lade Register R1 mit dem Wert des Mutex
3 CMP r1, #0          ;Vergleiche den Mutex-Wert mit 0
4   WFENE              ;Ist der Wert ungleich 0, warte auf Signalisierung
5   BNE lock_mutex;Wiederhole den Vorgang durch Sprung zum Anfang
6 STREX r1, #-1, [r0];Ist der Wert gleich 0, schreibe -1 als neuen Wert
7   CMP r1, #0        ;Überprüfe, ob Schreibzugriff erfolgreich war
8   BNE lock_mutex;Wiederholen, falls Schreibzugriff nicht erfolgreich
9   DMB              ;Stelle sicher, dass nachfolgende Speicherzugriffe
                    ;erst nach erfolgreicher Belegung erfolgen

```

Abbildung 4.9: Assembler-Quellcode für die Subroutine zum Belegen eines Mutex. Die konditionalen Instruktionen, die sich auf die vorige Vergleichsoperation beziehen sind eingerückt und blau markiert. Der rote Pfeil kennzeichnet das erneute Wiederholen der Routine nach dem Ausführen der Sprunganweisungen.

Register R0 und ein Wert von -1 gilt als belegter Mutex, während ein Wert von 0 als unbelegt gilt. Die exklusive Lade-Operation (LDREX) in Zeile 2 signalisiert der "Snoop Control Unit" (SCU), den Mutex-Wert aus dem Cache des anderen Kerns zu lesen, sollte dieser neuer als der Wert im Hauptspeicher sein. In Zeile 3 wird der geladene Mutex-Wert mit 0 verglichen (CoMPare). Dieser Vergleich führt zum Setzen der ALU-Flags (Carry, oVerflow, Zero und Negative), so dass die nachfolgenden Instruktionen mit dem Zusatz NE (NotEqual) nur ausgeführt werden, wenn der Mutex-Wert ungleich 0 (also belegt) ist. In diesem Fall wechselt der Kern in den Ruhezustand (WaitForEvent), aus dem er durch einen Software-Interrupt aufgeweckt wird, den der andere Kern nach der Freigabe des Mutex durch das Ausführen einer SEV-Instruktion (SignalEvent) auslöst. Die Sprung-Instruktion (Branch) in Zeile 5 dient zum Wiederholen des Vorgangs und wird ebenfalls bedingt in Abhängigkeit des vorangegangenen Vergleichs ausgeführt. Sollte der Mutex-Wert den Wert 0 enthalten haben, wird mit der Ausführung in Zeile 6 fortgefahren.

Der exklusive Schreibzugriff (SToReEXclusive) schreibt eine -1 als neuen Mutex-Wert zurück in den Speicher und legt die Status-Information des Zugriffs von der SCU in Register R1 ab. Dabei gilt der Wert 0 als erfolgreich, während ein Wert von 1 bedeutet, dass diese Speicherstelle nicht zuvor von einer exklusiven Lade-Operation von der SCU als 'zu überwachen' markiert wurde. In diesem Fall wird der Vorgang ebenfalls durch einen bedingten Sprung in Zeile 8 zum Anfang wiederholt. War der Schreibzugriff und somit die Belegung des Mutex erfolgreich, wird der Vorgang mit einer "Data Memory Barrier" (DMB) abgeschlossen. Dadurch werden alle nachfolgenden Speicherzugriffe auf die durch den Mutex zu schützende Ressource von dem anderen Kern erst nach der Mutex-Belegung beobachtet. Ohne die Barriere, könnten *Hazards* oder Code-Umsortierungen dazu führen, dass Speicherzugriffe auf die Ressource vor der Mutex-Belegung stattfinden.

Für die Erweiterung der Implementierung zu einem eintrittsinvarianten (*reentrant*) Mutex, wird die CPU-ID anstelle der Werte 0 und -1 als Mutex-Wert verwendet. Das Auslesen der CPU-ID erfolgt durch die CP15-Instruktion: "MRC p15, 0, r1, co, co, 5". Nach Ausführung dieser Instruktion steht die CPU-ID in Register R1. Dadurch kann der CPU, der den Mutex bereits belegt hat, diesen ohne Freigabe oder Auftreten eines Deadlocks beliebig oft erneut belegen.

5 Inbetriebnahme: "Open Multimedia Application Platform 4430"

Die OMAP4430-Plattform (Abbildung 5.1) dient als Ziel-MPSoC für die Ausführung der Laserscanner-basierten Objekterkennung.

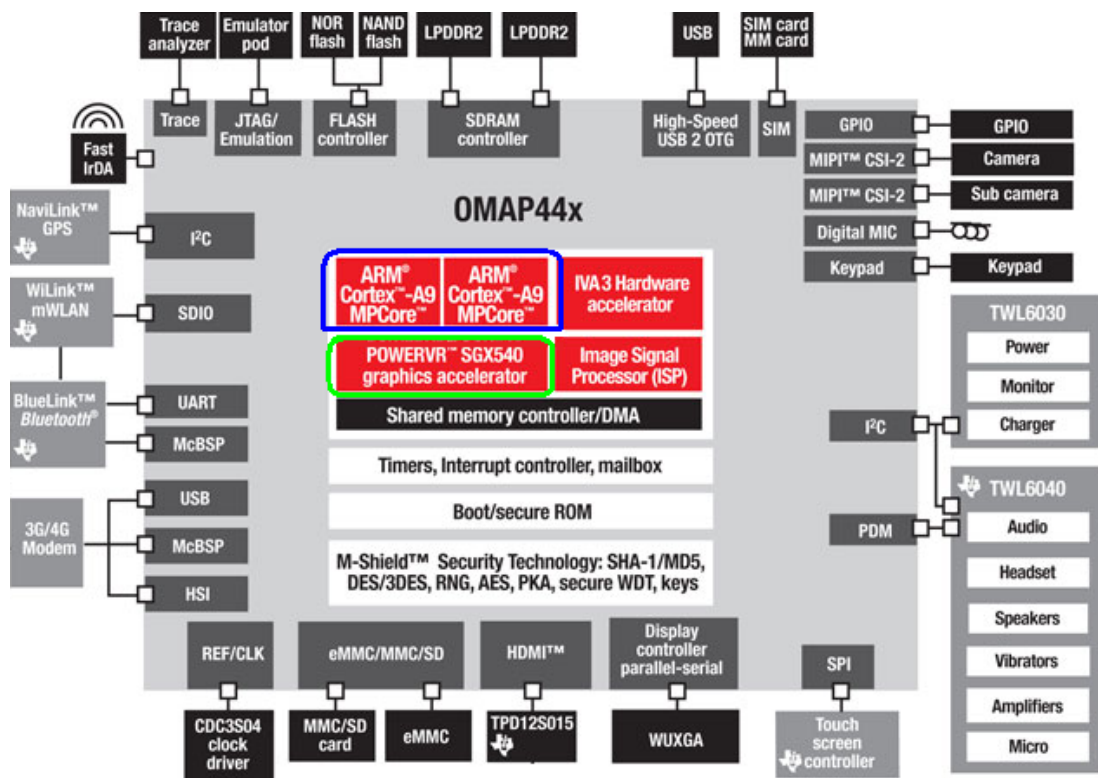


Abbildung 5.1: OMAP4-Komponenten mit integrierten Cortex-A9 1 GHz DualCore (blau), PowerVR HW-Grafikbeschleuniger (grün), 1 GB DDR2-RAM und den verfügbaren Peripherieschnittstellen [44].

5.1 DualCore-Initialisierung mit Bootloader

Die OMAP4430-Plattform verfügt über einen 48KB großen ROM-Speicher in dem der Startup-Code steht, der automatisch nach dem Einschalten der Versorgungsspannung ausgeführt wird. Zweck des Startup-Codes ist es, ein Boot-Image über USB, UART oder den SD-Kartenleser ins 64KB große statische RAM zu laden und dort auszuführen. Dieser Code im Boot-Image initialisiert die serielle RS232-Schnittstelle für Statusausgaben und lädt erweiterte "Multimedia Card" (MMC) Treiber für das FAT32-Dateisystem auf der SD-Karte. Anschließend kopiert er das Betriebssystemabbild ins dynamische RAM und startet es von dort aus. Die Auswahl über welche Schnittstelle das Boot-Image geladen wird, erfolgt durch die Belegung von 6 Input-Pins mit 3,3 KOhm Widerständen auf dem Pandaboard (Anhang A). Im Auslieferungszustand des Pandaboards (SYSBOOT[5:0] = "000101") wird das Boot-Image als "Master-Loader" (MLO) Datei im Wurzelverzeichnis der aktiven FAT32-Partition mit der Bezeichnung "BOOT" auf der SD-Karte über den Kartenleser geladen und ausgeführt (Abbildung 5.2). Die vollständige Präparierung des SD-Karten Dateisystems mit Android wird in Kapitel 6.1 beschrieben.

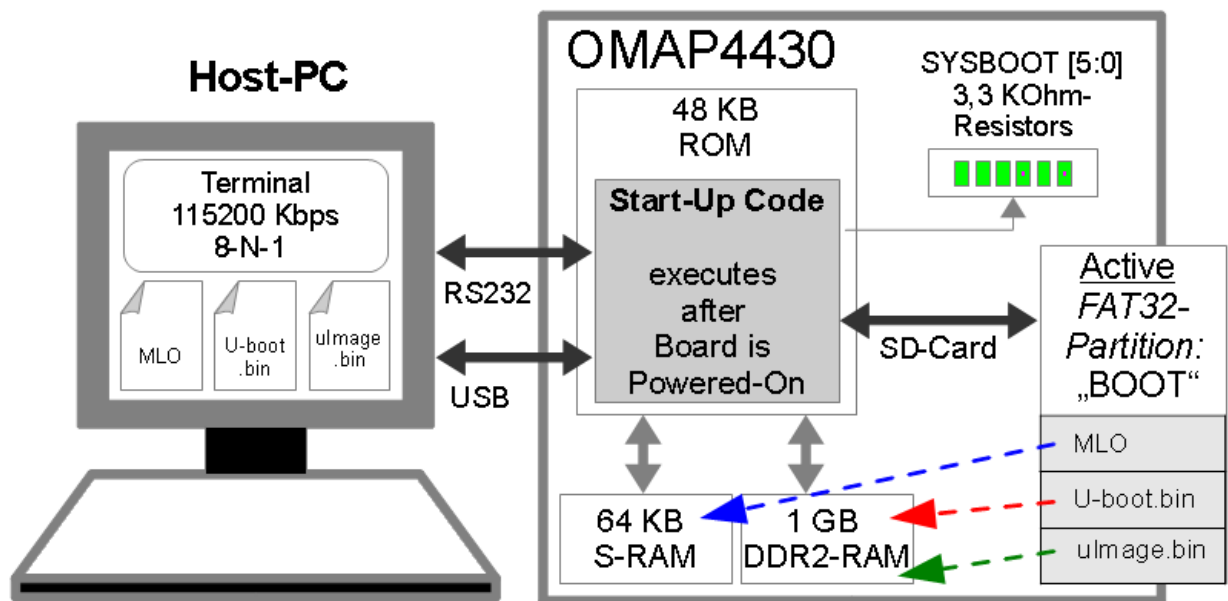


Abbildung 5.2: Mehrstufer Bootloader-Vorgang bei dem die SYSBOOT-Konfiguration die SD-Karte als Boot-Image Quelle bestimmt: 1. Xloader-MLO durch Start-Up Code in das statische RAM (blau) 2. U-boot durch Xloader in das dynamische RAM (rot) 3. Linux-Kernel durch U-boot in den Arbeitsspeicher (grün)

Nach der Inbetriebnahme der OMAP4430-Plattform wird der Bootvorgang als auch die Anwendung von einem Cortex-A9 Kern ausgeführt. Der andere Kern befindet sich währenddessen in einem stromsparenden "Wait For Event" (WFE) Modus und führt keine Instruktionen aus

(Abbildung 5.3). Zum Aufwecken schreibt der erste Kern (Master CPU) die Speicheradresse, an der die erste Instruktion für den zweiten Kern (Slave CPU) steht, in das memory-mapped `AUX_CORE_BOOT_1` Register und führt anschließend eine "Signal Event" (SEV) Instruktion aus. Der zweite Kern verlässt damit den WFE-Modus und springt an die Adresse im Register. Damit dieser Sprung erfolgen kann, aktiviert der erste Kern zuvor den Watchdog-Timer, die L1-Caches, die MMU und die SCU. Das `AUX_CORE_BOOT_0` Register dient dem ersten Kern zur Signalisierung, dass die Initialisierungen abgeschlossen sind bevor der zweite Kern den Sprung zur Adresse ausführt.

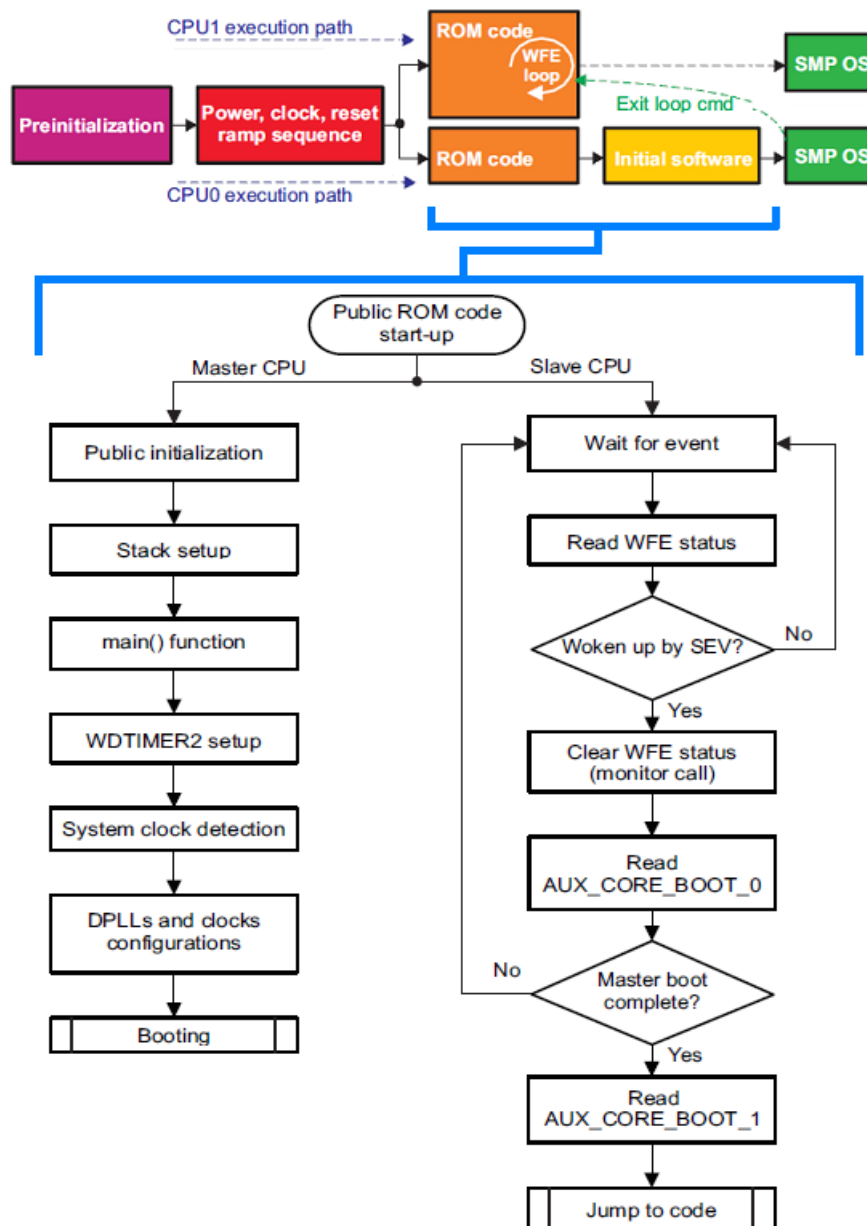


Abbildung 5.3: Start-Up Sequenz des DualCore-Prozessors auf der OMAP4430-Plattform [44].

5.2 PowerVR SGX540 GPU-beschleunigte Visualisierung

Die Ergebnisse der Objekterkennung werden von der Android-App in Echtzeit visualisiert. Durch Aktivieren der "Hardware accelerated" Option im Manifest-File des Android Projektes (Kapitel 6.2), werden die Zeichenoperationen nicht wie im Software-basierten Zeichenmodus von der CortexA9-DualCore CPU, sondern zu deren Entlastung von der *PowerVR-SGX540* "Graphics Processing Unit" (GPU) ausgeführt (Abbildung 5.4). Auf der GPU-Hardware wird CPU-

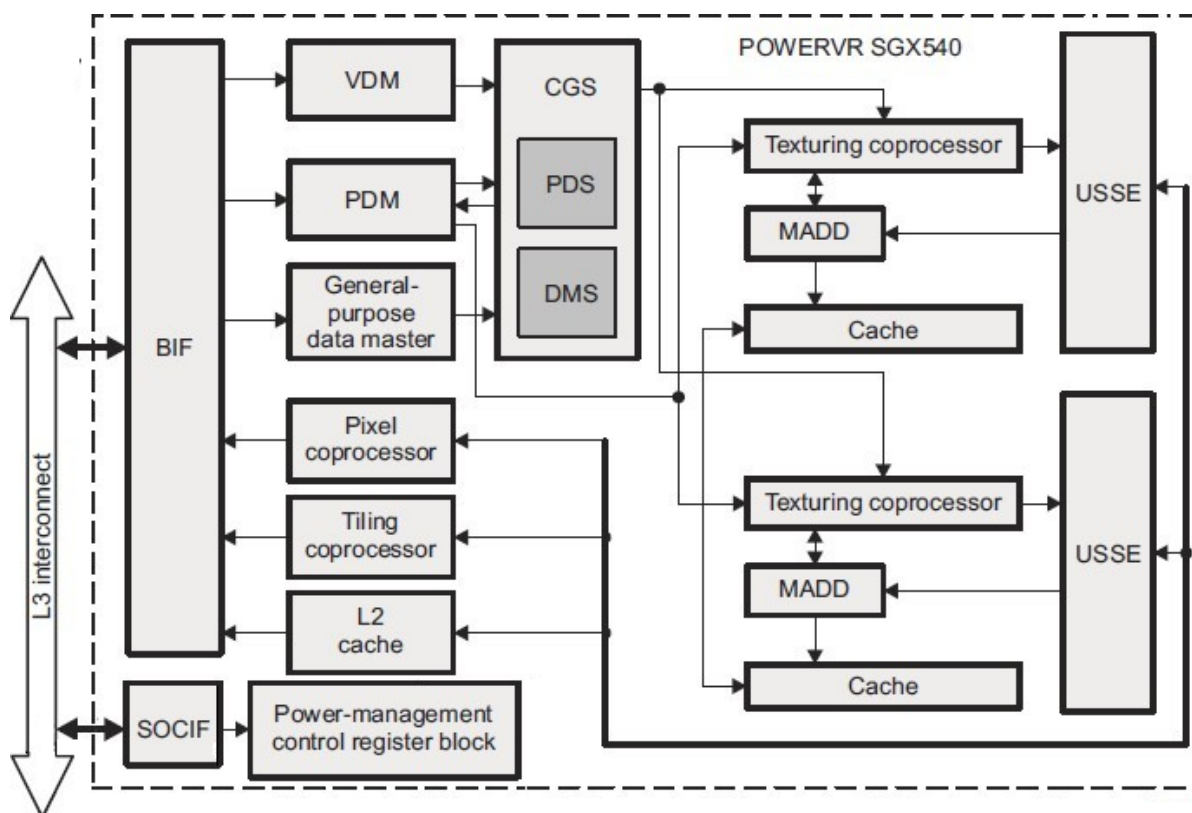


Abbildung 5.4: Funktionsblock-Übersicht der *PowerVR-SGX540* GPU auf der OMAP4430-Plattform [30].

unabhängig eine Micro-Kernel Betriebssystem-Software ausgeführt, an die eine auszuführende Grafikoperation, hier als *Task* bezeichnet, vom Android Betriebssystem übergeben wird. Die "Universal Scalable Shader Engine" (USSE) besteht im SGX540-Model aus vier *superscalar graphics pipelines*, denen jeweils ein *Task* zugeordnet ist und die sich jeweils aus einem "Thread Scheduler" und einer "Execution Unit" zusammensetzt. Ein "Thread Scheduler" verfügt über 16 GPU-Threads von denen er zu einem Zeitpunkt bis zu 4 gleichzeitig aktiv setzt und auf die "Execution Units" verteilt. Eine abgearbeitete *Task* erzeugt zur Signalisierung einen GPU-intern *Interrupt*, der vom μ Kernel empfangen wird und dem GPU-eigenen "Direct Memory Access"

(DMA) Controller zum Schreiben der neuen Pixelinformationen aus dem 2048 x 2048 Pixel großen Framebuffer in den Arbeitsspeicher veranlasst.

Zur Veranschaulichung der Thread-basierten parallelisierten Bearbeitung einer Task durch die GPU, wird der Ablauf beispielhaft anhand des Zeichnens einer Linie zwischen zwei gegebenen 2D-Punktkoordinaten ($x_1 = 0 ; y_1 = 5$) und ($x_2 = 7 ; y_2 = 2$). Aus den Koordinaten folgt in X-Richtung der Steigungsfaktor m über den jedem X-Wert von 0 bis 7 ein Y-Wert $y(x)$ zugeordnet wird:

$$m = \frac{2 - 5}{0 - 7} = \frac{-3}{-7} = 0,43 \quad \text{und} \quad y(x) = y_1 + x * m = x * 0,43$$

Unter Verwendung der GPU werden die acht zu berechnenden Y-Werte nicht sequentiell, sondern parallel von den vier Pipeline-Threads berechnet (Abbildung 5.5). Die Aufgabenverteilung

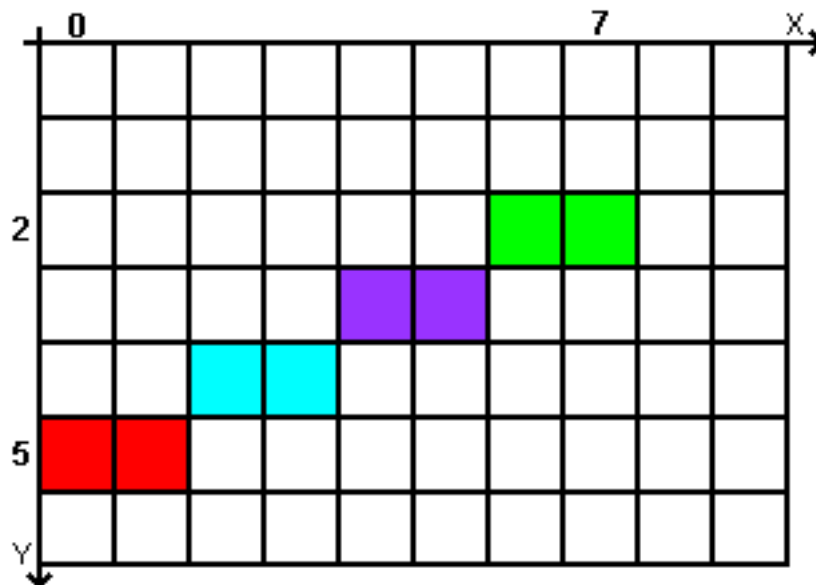


Abbildung 5.5: Koordinatensystem-Ausschnitt mit den von der GPU zu identifizierenden Pixel beim Zeichnen einer Linie zwischen den beiden Punkten ($x_1 = 0 ; y_1 = 5$) und ($x_2 = 7 ; y_2 = 2$). Die Task wird auf die vier aktiven Pipeline-Threads gemäß der Farbgebung aufgeteilt und gleichzeitig bearbeitet.

erfolgt über die Zuordnung der X-Werte auf die Thread-ID t_{ID} von 0 bis 3 durch:

$$x(t_{ID}) = x_1 + \frac{x_2 - x_1 + 1}{4} * t_{ID} = 2 * t_{ID}$$

Ein weiterer Vorteil der Hardware-beschleunigten 2D-Zeichenoperationen gegenüber dem Software-basierten Modus resultiert durch das "Tiling Based Rendering" (TBR). Ohne TBR

verursacht jede Veränderung im Framebuffer ein unmittelbares Neuzeichnen aller darstellbaren Objekte, die sich mit dem veränderten Bereich überschneiden. Eine Farbänderung einer Button-Schaltfläche innerhalb eines Rahmens führt zum Beispiel zum Neuberechnen des gesamten Rahmenbereiches, auch wenn dieser sich nicht verändert hat und auch wenn die Button-Schaltfläche selbst verdeckt und zu diesem Zeitpunkt nicht sichtbar ist. Mit TBR-Einsatz wird der Framebuffer durch eine Rasterung in *Kacheln* (engl. Tiles) aufgeteilt. Zu jedem *Kachel-Bereich* wird im Arbeitsspeicher eine Liste mit den darstellbaren Objekten in diesem Bereich geführt, sortiert nach Ihrer Position entlang der Z-(Tiefen)Achse. Die GPU zeichnet bei Veränderungen eines Objektes nur die beteiligten *Kachel-Bereiche* neu und verarbeitet dabei nur die Objekte, die in der Liste oberhalb des veränderten Objektes liegen.

5.3 PWM-Signalgenerierung durch externen AVR- μ C

An dem SoC-Fahrzeug sind als Aktoren zwei Motoren zur Fortbewegung angebracht: Einer der Motoren erzeugt den Antrieb, während der andere mit der Stellung der Lenkachse den Lenkeinschlag steuert. Angesteuert werden die Motoren über eine im Modellbau übliche Pulsweitenmodulation (PWM) mit einer Periodendauer von 10ms bis 20ms und einer Impulslänge mit einem High-Pegel von 1ms bis 2ms (Abbildung 5.6). Die Spannung der PWM beträgt 4-6V. Die OMAP4-Plattform auf dem Pandaboard verfügt über keine spezialisierte PWM-Peripherie,

| Impulslänge | Drehrichtung Antrieb | Stellung der Lenkung |
|-------------|---|--|
| 1ms | Rückwärts mit maximaler Geschwindigkeit | Maximaler Ausschlag links |
| 1ms-1,5ms | Rückwärts, proportional zur Impulslänge | Ausschlag links, proportional zur Impulslänge |
| 1,5ms | Stop | Mitte |
| 1,5ms-2ms | Vorwärts, proportional zur Impulslänge | Ausschlag rechts, proportional zur Impulslänge |
| 2ms | Rückwärts mit maximaler Geschwindigkeit | Maximaler Ausschlag rechts |



Abbildung 5.6: Auswirkung der Impulslänge des PWM-Signals zur Ansteuerung der Motoren auf Antrieb und Lenkung [1].

so dass diese sich nach Kapitel 22.2.2.1 des OMAP4430-Handbuchs (S. 4187) über die General-Purpose Timer *GPTIMER8* bis *GPTIMER11* erzeugen, da diese Timer PWM-Signale auf den Pins direkt ausgeben. Diese Pins werden jedoch von der USB-Peripherie belegt [38], die für die Kamera verwendet wird, um die Fahrspur aufzunehmen. Zusätzlich sind die Timer-Anschlüsse nicht auf den Expansion Connector herausgeführt. Der TWL6030 "Power Management Companion

Device", der unter anderem die 1.8V OMAP4430-Versorgungsspannung bereit stellt, verfügt über zwei PWM-Ausgänge, die jedoch auf dem Pandaboard nicht mit der Platine verbunden sind. Da es sich bei dem TWL6030 um ein BGA-Gehäuse handelt sind diese Anschlüsse nicht von außen zugänglich.

Die realisierte Lösung ist die Generierung der PWM, ausgelagert an einen externen Chip, der über einen Bus mit dem OMAP4430 verbunden ist. Zum Einsatz kommt ein verfügbarer AT90CAN128 AVR-Mikrocontroller [7] auf einer Adapterplatine, die die PWM und I²C-Anschlüsse des AVR zugänglich macht. Da der AVR mit einer 5V Spannung arbeitet, wird dessen PWM-Ausgang direkt an den PWM-Eingang des Servo-Motors angeschlossen (Abbildung 5.7). Für die Verbindung mit dem mit 1,8V arbeitendem OMAP4430 wird ein Pegelwandler

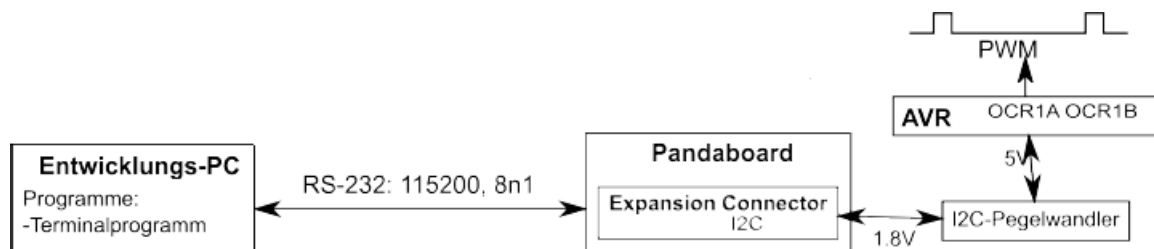


Abbildung 5.7: Die PWM wird über einen AVR-Mikrocontroller an den Pins OCR1A/OCR1B erzeugt, wobei ein I²C-Pegelwandler für die Anpassung der Spannungsdifferenzen zwischen AVR und Pandaboard eingesetzt wird [1].

verwendet. Zum Einsatz kommt dafür ein PCA 9517 "Level translating I²C-bus repeater" [36]. Der PCA 9517 verbindet einen I²C-Bus mit einer Spannung an der A-Seite von 0,9V bis 5,5V mit einem I²C-Bus an der B-Seite mit einer Spannung von 2,7V bis 5,5V; Der 1,8V OMAP4430 wird also an die A-Seite, der 5V AVR an die B-Seite des PCA 9517 angeschlossen. Da mit den beiden 1kOhm Widerständen R74 und R75 bereits PullUp-Widerstände am Pandaboard für den I²C-Bus vorhanden sind, werden nur auf der AVR-Seite zwei Widerstände angelötet (Abbildung 5.8).

Der OMAP4430 verfügt über vier I²C Controller, der vierte I²C-Controller ist zusammen mit der 1,8V Referenzspannung und der Masse mit Anschlüssen an dem Steckverbinder J3 des Pandaboard verbunden, die über den Pegelwandler mit dem AVR gekoppelt werden. Die beiden Motoren werden an die PWM-Anschlüsse OCR1A (Pin PB5) und OCR1B (PinPB6) des AVR-Mikrocontroller angeschlossen. Für die Erzeugung der PWM wird der Timer 1 des AVR mit der Betriebsart "Fast-PWM" konfiguriert und das Register ICR1 als Endwert für den Zähler genommen. Der AVR-Mikrocontroller ist mit 16MHz getaktet, der Vorteiler des Timer 1 wird auf 64 gesetzt, um den Timer mit 250kHz zu takten. Damit ergibt sich für die Erzeugung der PWM-Periodendauer von 20ms für den Endwert des Timers ein Wert von $250kHz * 20ms = 5000$.

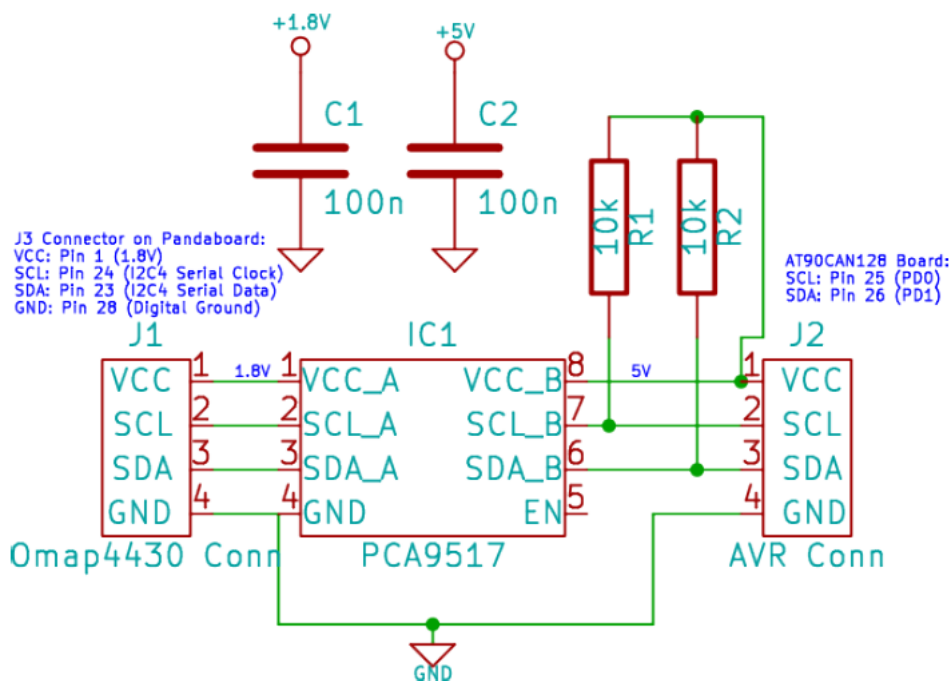


Abbildung 5.8: Schaltplan: I²C-Pegelwandler PCA 9517 zu der Umsetzung der Spannungsdifferenz zwischen Pandaboard (1,8 Volt) und AT90CAN128 (5 Volt). Auf der AVR Seite wurden PullUp-Widerstände angelötet [1].

Für einen 1ms langen Impuls ergibt sich entsprechend ein Wert von 250 und für 2ms ein Wert von 500.

Damit passt der Wertebereich von 250 bis 500 in ein Byte und wird als Versatz von der Mittelstellung mit einem vorzeichenbehafteten Byte zum AVR gesendet. Um eine PWM mit 1ms zu erzeugen (Linke maximale Stellung des Servos) wird also ein Byte mit einem Wert von -125 gesendet und für eine PWM mit 2ms (Rechte maximale Stellung des Servos) ein Byte mit einem Wert von +125. Zuvor muss die Registernummer gesendet werden, in die die Impulslänge geschrieben werden soll. Register 1 steht für den PWM-Ausgang *OCR1A* und Register 2 für den PWM-Ausgang *OCR1B*. Die Implementierung der I²C-Bus Ansteuerung mit Android unter Verwendung des "Native Development Kit" wird in Kapitel 6.3 beschrieben.

6 Konfiguration und Entwicklungsumgebung für Linaro Android 4.0.3

Die Installationsabläufe des Android 4.0.3 Betriebssystems von Linaro und der Entwicklungswerkzeuge werden in diesem Kapitel beschrieben. Für die Durchführung wurden die folgenden Hard- und Softwarekomponenten verwendet:

- Pandaboard-Entwicklungsboard REV-A2 [38].
- 5 Volt Power-Supply mit 5.5 x 2.1 x 10mm Center Positive Barrel DC-Ausgangsanschluss.
- USB zu RS232-Seriell Adapterkabel mit PL-2303 Treiber-CD für Windows-XP.
- Cat 5 TwistedPair-Kabel gemäß Fast-Ethernet Standard.
- SecureDigitalHC-Karte der Geschwindigkeitsklasse 4 mit 8GB-Speicherplatz.
- Windows-XP SP3 Host-PC mit SD-Kartenleser, freiem USB-Port und Internetzugang.
- Virtuelle Maschine VirtualBox [37] mit Linux-Ubuntu 11.04 Image [46].
- Linaro Android 4.0.3 *Landing-Build* [34].
- "Android Development Tools" als Eclipse-Plugin zusammen mit dem Android NDK [23].

Im Mai 2011 gründeten Ingenieure von ARM, Freescale Semiconductor, IBM, Samsung, ST-Ericsson und Texas Instruments die gemeinnützige Linaro-Organisation zur Förderung von Quellcode-offenen Innovationen für ARM-SoCs. Die Android-*Landing*-Implementierung von Linaro besteht aus dem den Betriebssystem-Kernel von Texas Instruments und schnitt auf dem Pandaboard in Benchmark-Tests stabiler und rechenleistungsfähiger [11] ab, als die Android-Variante aus dem "Android Open Source Project" von Google.

6.1 Formatierung der SD-Karte mit Betriebssystemabbild

Das Kopieren des Betriebssystemabbildes auf die SD-Karte erfolgt mit den "Linaro Image Tools", die für Linux-basierte Betriebssysteme verfügbar sind und auf dem Windows-Host durch Verwendung der frei verfügbaren VirtualBox von Oracle zusammen mit der Ubuntu Linux-Distribution als Gastbetriebssystem wie folgt ausgeführt werden:

1. Herunterladen der VirtualBox-Installationsdatei und des Ubuntu-Image.
2. Ausführen der VirtualBox-Installationsdatei auf dem Host-PC und Erzeugen einer virtuellen Festplatte über die Schaltfläche "Neu".
3. Auswahl des von Linux-Ubuntu im Installationsassistenten als Gastbetriebssystem und dem Image als Installationsquelle mit Übernahme der unveränderten Default-Einstellungen.
4. Einrichten des Gastbetriebssystems über die Schaltfläche "Starten", die den Installationsprozess initiiert, in dessen Verlauf Zeitzone, Tastaturlayout und Benutzername angegeben werden.
5. Die Installation schließt mit einem Neustart des Gastbetriebssystems ab (Abbildung 6.1).

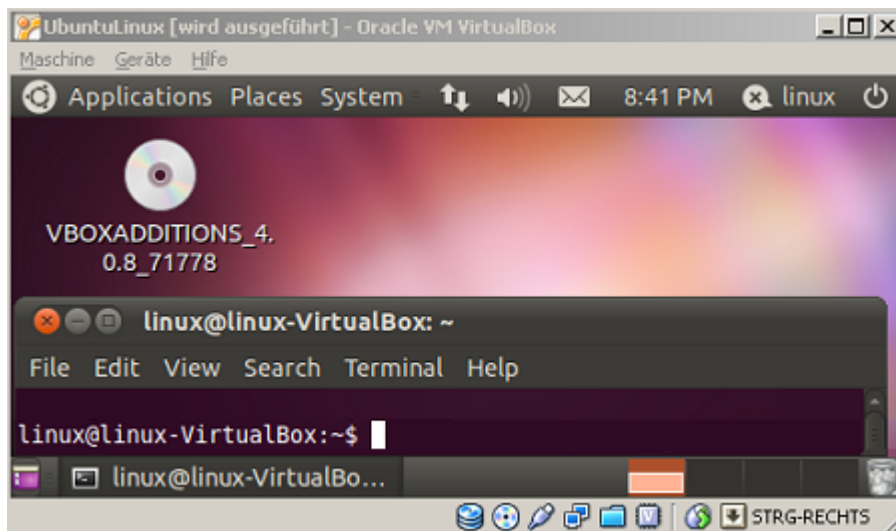


Abbildung 6.1: Terminal-Shell zur Eingabe der Befehlsketten im Ubuntu-Gastbetriebssystem, das in einer virtuellen Maschine auf dem Host-PC ausgeführt wird.

Ein vorhandenes Filesystem einschließlich des "Master Boot Record" auf der SD-Karte ist vor dem Ausführen der Linaro Image-Tools zu löschen, zum Beispiel mit dem "GParted Partition

Editor" [21]. Nach dem Einhängen der Karte in das Gastbetriebssystem ist deren Laufwerksbezeichnung durch den "dmesg" Konsolenbefehl zu ermitteln und anstelle der "/dev/sdX" Zeichenkette in den nachfolgenden Shell-Terminal Befehlsketten einzusetzen:

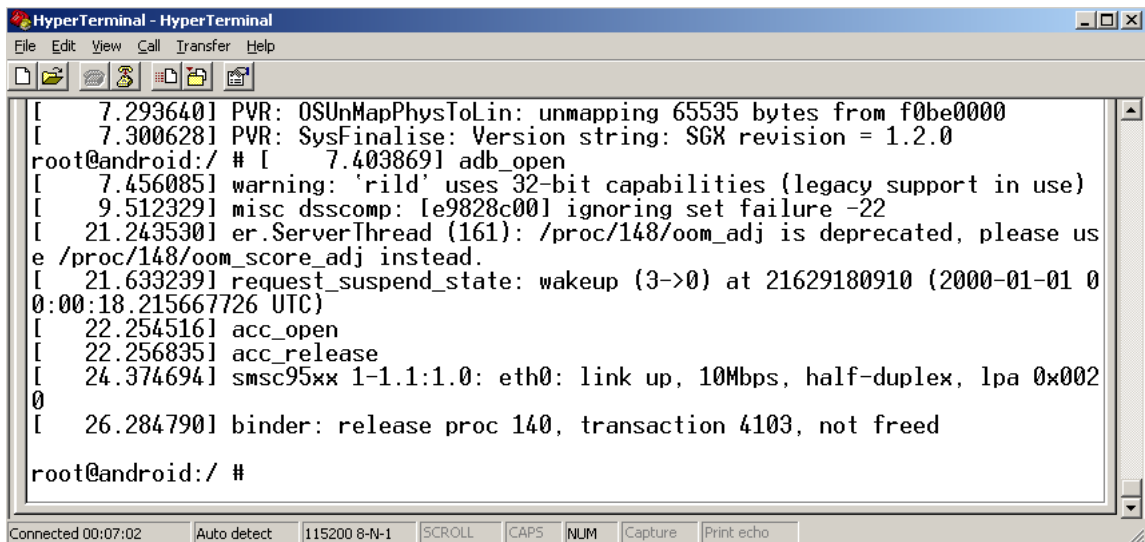
1. Herunterladen des Android-Images (*URL* jeweils ersetzen durch:
`http://snapshots.linaro.org/android/linaro-android/landing-panda/53/target/product/pandaboard`):
`"wget URL/boot.tar.bz2 URL/system.tar.bz2 URL/userdata.tar.bz2"`
2. Installation der Linaro Image-Tools: `"sudo add-apt-repository ppa:linaro-maintainers/tools"`
3. `"sudo apt-get update"`
4. `"sudo apt-get install linaro-image-tools"`
5. Partitionierung des SD-Karte: `"sudo linaro-android-media-create -mmc /dev/sdX -dev panda -system system.tar.bz2 -boot boot.tar.bz2 -userdata userdata.tar.bz2"`
6. Hinzufügen der proprietären Programmbibliotheken von Google durch ein Shell-Skript:
`"wget http://releases.linaro.org/12.04/android/images/panda-ics-gcc47-tilt-tracking-blob/install-binaries.sh"`
7. `"chmod a+x install-binaries.sh"`
8. `"./install-binaries.sh /dev/sdX2"`

Durch Betätigung der Space-Taste mit der an das Ende der Lizenzvereinbarung gesprungen wird und mit der "I ACCEPT" Eingabe ist die Erstellung der SD-Karte abgeschlossen.

Die SD-Karte wird in den Kartenleser des Pandaboards gesteckt und das Pandaboard über das USB zu RS232-Seriell Adapterkabel mit dem Entwicklungs Host-PC verbunden. Im nächsten Schritt wird die Hyper-Terminal Anwendung auf dem Host-PC aufgerufen und eine neue Verbindung erstellt. Die Auswahl der virtuellen COM-Ports richtet sich nach dem verwendeten USB-Port, an dem das Adapterkabel angeschlossen ist und setzt unter Windows XP die Installation der PL-2303 Treiber-CD voraus. Die Einstellungen für die RS232-Übertragung mit dem Pandaboard lauten:

- BAUD-RATE: 115200
- DATA-BITS: 8
- PARITY: Keine / none
- STOP-BITS: 1
- FLOW CONTROL: Kein / none

Nach dem Anschließen der Spannungsversorgung an das Pandaboard werden die Konsolenausgaben beim Android-Bootvorgang in der Hyperterminal-Anwendung angezeigt (Abbildung 6.2).



```
HyperTerminal - HyperTerminal
File Edit View Call Transfer Help
[ 7.293640] PVR: OSUnMapPhysToLin: unmapping 65535 bytes from f0be0000
[ 7.300628] PVR: SysFinalise: Version string: SGX revision = 1.2.0
root@android:/ # [ 7.403869] adb_open
[ 7.456085] warning: 'rild' uses 32-bit capabilities (legacy support in use)
[ 9.512329] misc dsscomp: [e9828c00] ignoring set failure -22
[ 21.243530] er.ServerThread (161): /proc/148/oom_adj is deprecated, please use
 /proc/148/oom_score_adj instead.
[ 21.633239] request_suspend_state: wakeup (3->0) at 21629180910 (2000-01-01 0
0:00:18.215667726 UTC)
[ 22.254516] acc_open
[ 22.256835] acc_release
[ 24.374694] smsc95xx 1-1.1:1.0: eth0: link up, 10Mbps, half-duplex, lpa 0x002
0
[ 26.284790] binder: release proc 140, transaction 4103, not freed

root@android:/ #
```

Abbildung 6.2: Hyper-Terminal Anwendung auf dem Windows Host-PC mit den RS232-Port Ausgaben vom Pandaboard nach Abschluss des Android-Bootvorgangs.

Ist der Bootvorgang abgeschlossen wird das Betriebssystem über den Hyper-Terminal durch die folgenden Befehlszeilen zur Android-App Entwicklung über die Ethernet-Schnittstelle konfiguriert:

1. Schreibzugriffe auf die Systempartition freischalten:
"mount -o rw,remount -t yaffs2 system /system"
2. IP-Adresse und Netzmaske der Ethernet-Schnittstelle setzen:
"ifconfig etho up 192.168.2.88 netmask 255.255.255.0"
3. Android-Debug-Bridge *Deamon* reinitialisieren: "stop adbd"
"setprop service.adb.tcp.port 5555"
"start adbd"
4. Lese- und Schreibzugriffe auf das I²C-dev-Interface freischalten: "chmod 777 /dev/i2c-4"

Die im zweiten Schritt gewählte IP-Adresse und Netzmaske sind durch jede andere Auswahl austauschbar, die zum lokalen Subnetz des Host-PCs gehören. Ersatzweise zur Ethernet-Schnittstelle ist die SW-Entwicklung über den USB Mini-B Stecker des Pandaboards durch folgende Befehle zu aktivieren:

1. "echo 0 > /sys/class/android_usb/androido/enable"
2. "echo 0451 > /sys/class/android_usb/androido/idVendor"
3. "echo D101 > /sys/class/android_usb/androido/idProduct"
4. "echo adb > /sys/class/android_usb/androido/functions"
5. "echo 1 > /sys/class/android_usb/androido/enable"

Die Vorteile der Entwicklung über die Ethernet-Schnittstelle im Vergleich zur USB-Schnittstelle sind:

- Kalter Neustart des Pandaboards durch Trennung genau einer Spannungsversorgung. Die durch 500mA begrenzte Stromaufnahme des Pandaboards über die Mini-B USB-Schnittstelle ist zum fehlerfreien Betrieb alleine nicht ausreichend, verhindert aber einen Neustart des Pandaboards durch Entfernen des Netzteils, so dass zum Neustart in der USB-Konfiguration eine zusätzliche Steckverbindung abzuziehen und erneut anzuschließen ist.
- Nebenläufige SW-Entwicklung auf mehreren Ziel-Plattformen durch IP-Adressierung. Die USB-Konfiguration erlaubt die App-Entwicklung von einem Host-PC auf genau einem Pandaboard. Durch die Ethernet-Infrastruktur können von einem Host-PC aus Android-Anwendungen auf mehrere Pandaboards verteilt und ausgeführt werden ohne Änderung der physikalischen Verbindungen. Entsprechend kann die SW-Entwicklung auf einem Pandaboard von jedem Host-PC im IP-Subnetz erfolgen.
- Betriebssystem-unabhängige Konfiguration ohne USB-Treiber. Die Erkennung des Pandaboards als ein entwicklungsfähiges Android USB-Device erfordert zusätzliche Konfigurationseinträge, die vom gewählten Host-Betriebssystem abhängig sind (siehe Kapitel 6.2) und bei der Ethernet-Variante wegfallen.

6.2 Eclipse-Plugin "Android Development Tools" auf Windows Host-PC

Die Software-Entwicklung der Android-App erfolgt unter Verwendung der Eclipse-IDE [45]. Über den Menüpunkt "Help" und Untermenüpunkt "Install New Software..." erfolgt die Installation der Android-Entwicklungswerkzeuge, deren neuster Stand vom *Google Repository*: "<https://dl-ssl.google.com/android/eclipse/>" heruntergeladen wird (Abbildung 6.3).

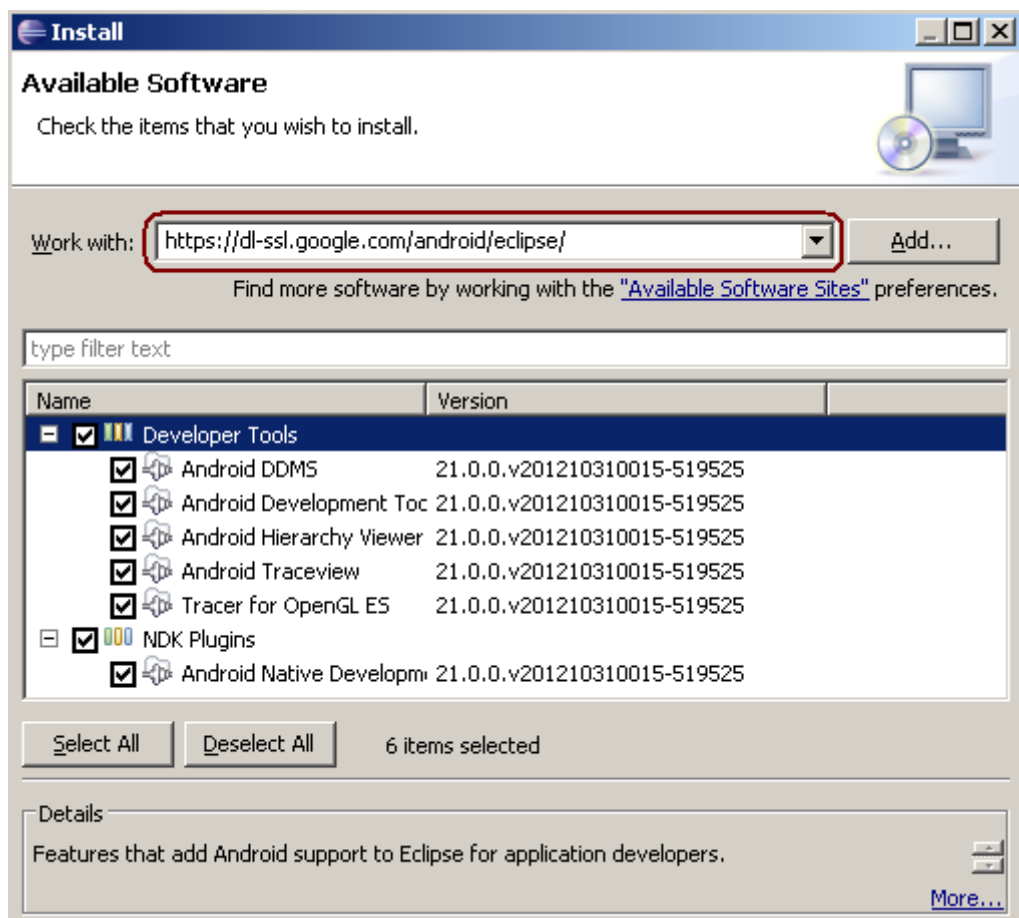


Abbildung 6.3: Installation des "Android Development Tools" Eclipse-Plugins. Die SW-Module werden durch die Angabe des *Google Repository* (rot) ausgewählt und aktualisiert.

Nach dem anschließenden Neustart der Eclipse-IDE öffnet sich der "Android SDK Manager" zur Auswahl der Entwicklungsplattform (Abbildung 6.4). Diese enthält Android Betriebssystemversions-spezifische API-Programmibliotheken und die vom dem Host-PC Betriebssystem abhängige "Android Debug Bridge" (ADB) Werkzeugpalette.

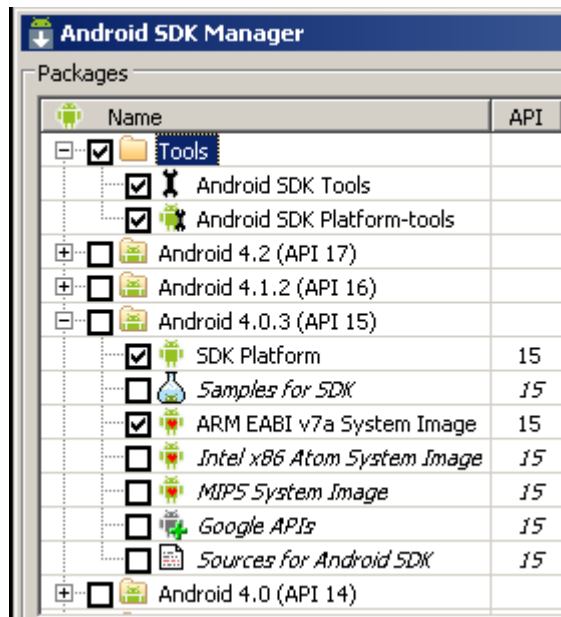


Abbildung 6.4: Auswahl der Android Entwicklungsplattform im SDK-Manager zu der Betriebssystemversion 4.0.3 (API 15) auf dem Pandaboard.

Für die Entwicklungs-Konfiguration über die USB-Schnittstelle ist an dieser Stelle zusätzlich das "Google USB Driver" Paket auszuwählen. Im SDK-Installationsunterverzeichnis: "/extras/google/usb_driver/" wird in diesem Fall automatisch die Datei *android_winusb.inf* angelegt. In diese Datei sind auf einem 32-bit Windows Host-PC unterhalb des *[Google.NTx86]* Eintrages, auf einem 64-bit Windows unterhalb des *[Google.NTamd64]* die folgenden Zeilen hinzuzufügen:

```

%SingleAdbInterface% = USB_Install, USB\VID_0451&PID_D101
%CompositeAdbInterface% = USB_Install, USB\VID_0451&PID_D102&MI_01
%CompositeAdbInterface% = USB_Install, USB\VID_0451&PID_D106&MI_02
%CompositeAdbInterface% = USB_Install, USB\VID_0451&PID_D107&MI_03
%SingleAdbInterface% = USB_Install, USB\VID_0451&PID_FFFFE
%CompositeAdbInterface% = USB_Install, USB\VID_0451&PID_FFFE&MI_01
%SingleAdbInterface% = USB_Install, USB\VID_0451&PID_D022
%CompositeAdbInterface% = USB_Install, USB\VID_0451&PID_D022&MI_01
%CompositeAdbInterface% = USB_Install, USB\VID_0451&PID_D10A&MI_01

```

Unter Verwendung der Ethernet-Konfiguration ist zum Entwickeln und Ausführen eines Android-Projektes die Ziel-Plattform auszuwählen. Dies erfolgt auf dem Host-PC im SDK-Installationsunterverzeichnis: "/platform-tools/" durch den Aufruf der Befehlszeile: "adb connect 192.168.2.88".

6.3 "Native Development Kit" zur I²C-Bus Steuerung

Mit dem I²C-dev-Interface lässt sich von einem Programm aus auf den I²C-Bus zugreifen [14]. Bei dem I²C-dev-Interface handelt es sich um Dateien im Gerätedateisystem /dev, pro I²C-Controller des OMAP4430 existiert eine Datei. Der vierte I²C-Controller, der hier verwendet wird, wird durch die Datei /dev/i2c-4 dargestellt und ist im Kernel auf eine Taktfrequenz von 400kHz eingestellt. Auf diese Datei wird zur Ansteuerung des AVR (Kapitel 5.3) mit nativem Maschinenbefehle zur Manipulation von Dateien zugegriffen.

- `open()`: Öffnen der Gerätedatei.
- `read()`: Lesen von Datenströmen von dem I²C-Gerät.
- `write()`: Schreiben von Datenströmen an das I²C-Gerät.
- `close()`: Schließen der Gerätedatei.
- `ioctl()`: Setzt die Adresse des I²C-Gerätes an den die `read()` und `write()` Operationen gehen. Mit dem Befehl wird die I²C-Startbedingung ausgeführt. Das Makro `I2C_SLAVE` ist in der Header-Datei `linux/i2c.h` definiert.

Zur deren Ausführung werden Anteile der Android Anwendung von der *Dalvik*-Java virtuellen Maschine ausgelagert und als Assembler/C Anweisungen explizit in eine Programmbibliothek kompiliert. Hierzu wird das Android "Native Development Kit" (NDK) [23] entpackt und ein "jni" (*Java Native Interface*) Unterverzeichnis im Android-Projekt erstellt. In diesem Verzeichnis sind vier Textdateien mit folgendem Inhalt zu erstellen:

- Auswahl der Ziel-Architektur. **Application.mk**: "APP_ABI := armeabi-v7a"
- Compiler/Linker-Konfigurationsparameter. **Android.mk**:
"LOCAL_PATH := \$(call my-dir)
include \$(CLEAR_VARS)
LOCAL_MODULE := hpec
LOCAL_SRC_FILES := hpec.c
LOCAL_LDLIBS += -llog
include \$(BUILD_SHARED_LIBRARY)"
- Batchskript zum Build-Vorgang wobei *NDK* mit dem Ort des NDK-Stammverzeichnisses zu ersetzen ist. **RunMakeBuild.bat**: "start *NDK*/ndk-build"
- C-Quellcode. **hpec.c**: (Abbildung 6.5)

```

1 #include <sys/ioctl.h>
2 #include <fcntl.h>
3 #include <linux/i2c.h>
4 #include <jni.h>
5 #include <android/log.h>
6
7 #define ADDR (0x50>>1)
8
9 void set_pwm(char chan, char val) {
10     int file;        // File-Descriptor
11     char buf[2];    // Send-Buffer
12
13     file = open("/dev/i2c-4", O_RDWR); // Open character device file
14     if (file < 0) {
15         __android_log_print(ANDROID_LOG_ERROR, "I2C", "open failed\n");
16     }
17     if (ioctl(file, I2C_SLAVE, ADDR) < 0) { // Set I2C slave address
18         __android_log_print(ANDROID_LOG_ERROR, "I2C", "ioctl failed\n");
19     }
20     buf[0] = chan;  // PWM-Output channel 1 or 2
21     buf[1] = val;  // Value from -127 (minimal) to 127 (maximal)
22     if (write(file, buf, 2) != 2) {
23         __android_log_print(ANDROID_LOG_ERROR, "I2C", "write failed\n");
24     }
25     close(file); // Close character device file
26 }
27
28 jint Java_haw_hpec_HPECActivity_FromJNI(JNIEnv* env, jobject thiz, jint jchan, jint jval)
29     set_pwm(jchan, jval); // Call plain C function
30 }

```

Abbildung 6.5: Native C Anweisungen zur Kommunikation mit dem AVR-Mikrocontroller über I²C. In zwei Bytes wird zuerst die PWM-Ausgangskanal Auswahl und danach die Impulslänge über die I²C-Verbindung gesendet.

Nach der Ausführung der Batchskript-Datei befindet sich die Programmbibliothek als *Shared-Object Library* im Unterverzeichnis des Android Projekts: `/obj/local/armeabi-v7a/libhpec.so`. Der Aufruf der Bibliotheksfunktion erfolgt von der Android-App zur Laufzeit in zwei Schritten:

1. Laden der *Shared-Object Library* beim Start der App innerhalb der `onCreate()` Aktivitäts-Methode durch: `"System.loadLibrary("hpec");"`.
2. Aufruf im Rahmen der Objekterkennung der durch: `"public native int FromJNI(int chan, int val);"` deklarierten Aktivitäts-Methode.

Beim Modifizieren oder Hinzufügen von nativem Funktionen ist die Namensbezeichnung zu beachten. Die Funktionsdeklaration im C-Quellcode muss der folgenden Namenskonvention genügen:

```
"[ReturnType] Java_[PackageNameFirstSegment]_[PackageNameSecondSegment]
_[CallerJavaClassName]_[NativeJavaFunctionName](JNIEnv* env, jobject thiz
, [CommaSeperatedParameterList]) {...}"
```

In der mit einem Oszilloskop durchgeführten Messung der PWM zur Validierung werden die Sollzeiten eingehalten (Abbildung 6.6).

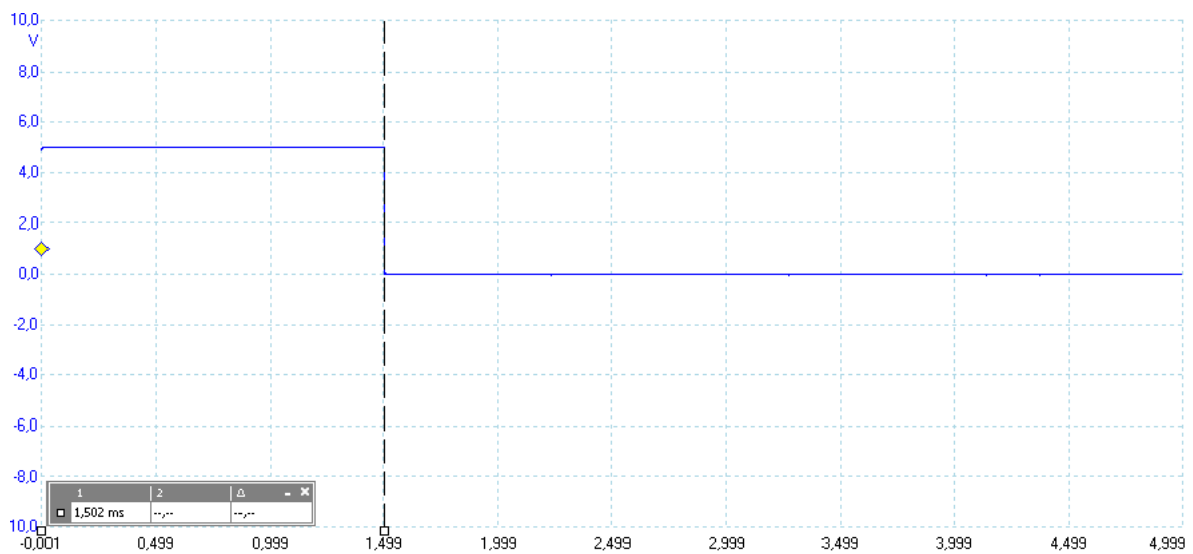


Abbildung 6.6: Von dem AVR-Mikrocontroller generierte PWM mit einer gemessenen Impulslänge von 1,5 Millisekunden [1].

7 System-Integration und Software-Implementierung

Unter dem Einsatz der bisher vorgestellten Technologien (Abbildung 7.1) realisiert und integriert die entwickelte Android-Anwendung eine Erfassung der Laserscanner-Umgebung, die sich nach potenziellen Hindernissen für das SoC-Fahrzeug orientiert und in diesem Kapitel vorgestellt wird.

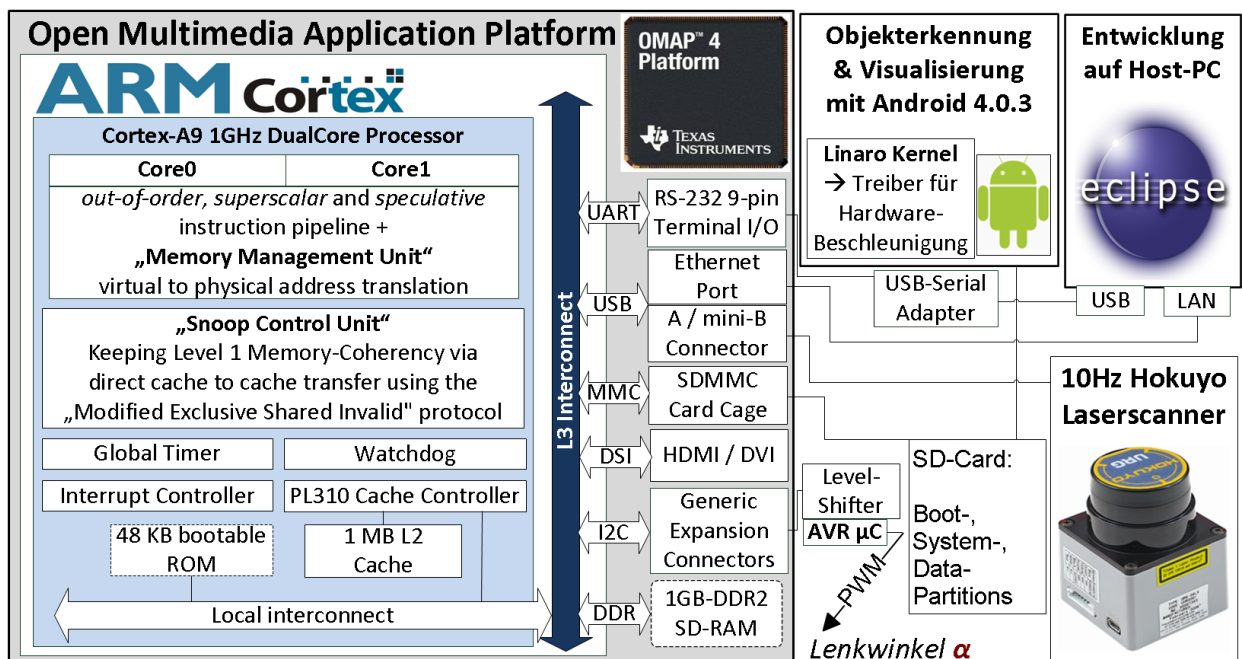


Abbildung 7.1: Übersicht der eingesetzten und vorgestellten Technologien im Rahmen der Laserscanner-basierten Objekterkennung.

Im autonomen Betrieb löst und koordiniert die Objekterkennungs-Software die folgenden Teilaufgaben:

- Messdatenerfassung vom *Hokuyo*-Laserscanner über die USB-Schnittstelle (Kapitel 7.1).
- Objektsegmentierung aus den dekodierten Messwerten nach dem 3-Punkte Schwellwertverfahren (Kapitel 7.2).
- Stabilisierung des dreieckig-approximierten Hindernis-Modells für ein Segment mit dem "Random Sample Consensus" (RANSAC) Verfahren zur Bereinigung von Messwerten außerhalb eines Messtoleranzbereiches (Kapitel 7.3).
- Visualisierung der erfassten Umgebung durch GPU-beschleunigte Grafikoperationen mit den identifizierten Objekten in Echtzeit (Kapitel 5.2).
- Native I²C-Bus Ansteuerung eines externen AVR- μ C zur PWM-Signalgenerierung für die Adaption des Lenkwinkels als Vorbereitung von Ausweichmanövern (Kapitel 5.3 & 6.3).
- Mutex-basiertes "Symmetrisches Multiprozessorsystem" (SMP) Synchronisationskonzept für die prioritätsgesteuerte Aufgabenpartitionierung durch SW-Threads auf den DualCore-CPU zur Laufzeit (Kapitel 7.4).

Implementiert ist die Android-App in der Java-Klasse *HPECActivity* für die "Dalvik Virtual Machine" (DVM). Im Unterschied zur "Java Virtual Machine" (JVM), für die jede Java-Klasse separat in eine *.class* Bytecode Datei übersetzt wird, werden alle Klassen einer Android-App in genau eine *classes.dex* Datei zusammengepackt. Ein weiterer Unterschied zwischen den virtuellen Maschinen besteht darin, dass für jede Android-App eine eigene Instanz der DVM erzeugt wird und dem Rechnermodell einer Registermaschine entspricht, während eine einzige JVM-Instanz sämtliche Java-Prozesse ausführt und bei der es sich um einen Kellerautomaten handelt. Für Threads einer JVM stehen betriebssystemunabhängige Prioritäten von 1 (niedrigste) bis 10 (höchste) zur Verfügung, die die JVM auf die Betriebssystem-bedingten Prioritäten abbildet. Die statische Android API-Funktion: "android.os.Process.setThreadPriority()" weist einem Thread systemweit einen *LinuxKernel*-basierten Prioritätswert im Bereich von -20 (höchste) bis +20 (niedrigste) zu und wird für die prioritäts-gesteuerte SMP-Partitionierung verwendet.

Agitationsstart und -Terminierung der Anwendung richten sich nach dem vom Android Betriebssystemkontext vorgegebenen *Aktivitätslebenszyklus*, der für jede Aktivität einer App bindend ist. Die von der *Activity*-Klasse geerbten Ereignismethoden werden dabei mit den Objekterkennungs-Routinen überschrieben (Abbildung 7.2).

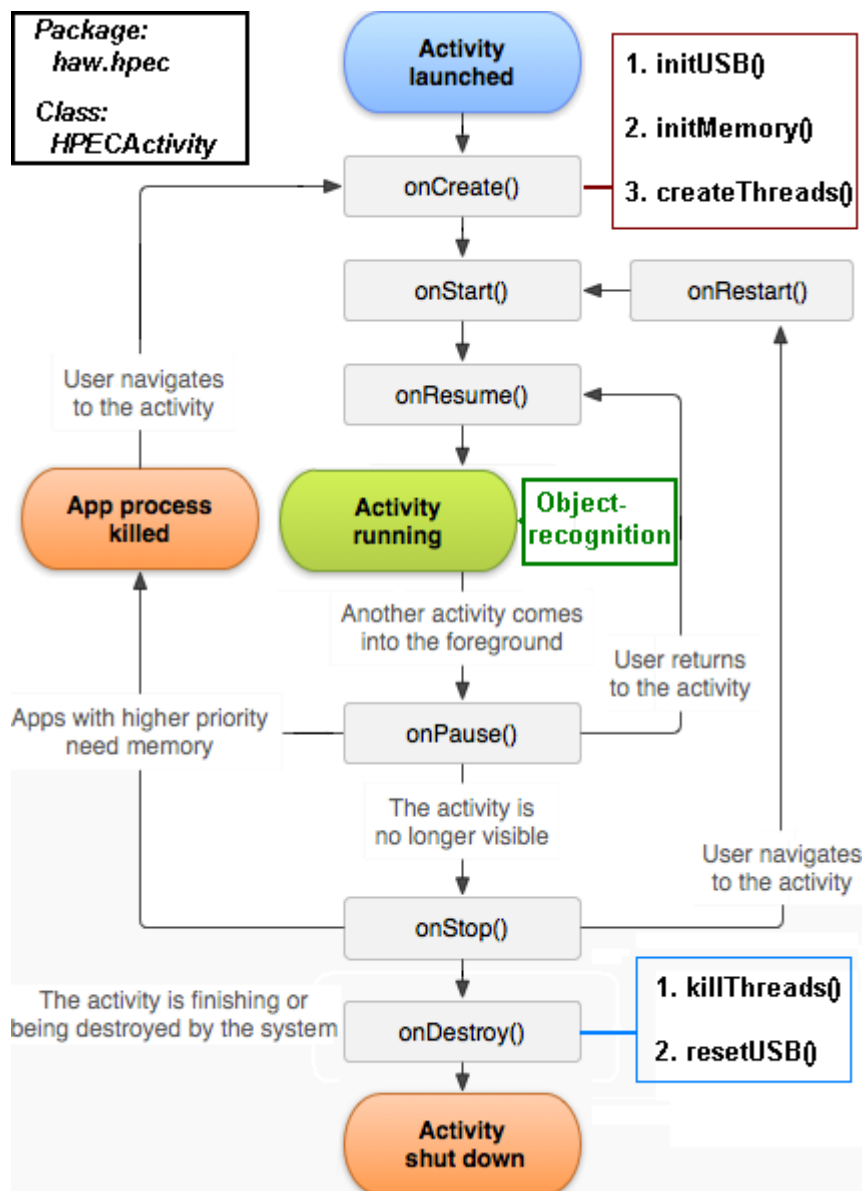


Abbildung 7.2: Lebenszyklus der HPEC-Aktivität mit den Funktionsaufrufen zur Initialisierung (rot) und Ressourcenfreigabe (blau). [24]

Dem Eclipse-Projekt zur Anwendung liegt die folgende Verzeichnisstruktur zur Grunde:

"/": Enthält die `AndroidManifest.xml` (Anhang B), in der die Betriebssystem-Version der Zielplattform und Anwendungsübergreifende Zugriffsberechtigungen festgelegt werden. Hier sind die beiden Attribute "Debuggable" und "Hardware accelerated" auf `true` gesetzt.

"src/": Quellcode Verzeichnis in Package-Namen unterteilt, die aus mindestens zwei Segmenten bestehen. Die Anwendung ist als Java-Klasse in `HPECActivity.java` im Package `haw.hpec` implementiert.

"gen/": Von der Entwicklungsumgebung generierter Quellcode wird hier abgelegt. In *R.java* sind den "User Interface" (UI) Ressourcen, die mit dem integrierten Layout-Editor erstellt wurden, ID-Konstanten zugeordnet, über die der Ressourcenzugriff zur Laufzeit erfolgt.

"bin/": Enthält die zur Übertragung auf Android ausführbare Anwendung in Form eines komprimierten "Application Package File" (APK) Paketes.

"jni/": Nativer C-Quellcode zur I²C-Bus Ansteuerung mit Build-Batchscript (Kapitel 6.3).

"libs/": Aus dem "jni/" Verzeichnis heraus kompilierte Programmbibliothek für die Ziel-Architektur *armeabi-v7a/libhpec.so*.

"res/": Enthält die Android-Ressourcen UI-Layout Beschreibung *layout/main.xml* (Anhang C), formatierte Zeichenketten-Konstanten *values/strings.xml* und die Icon-Grafikdatei *drawable/ic_launcher.png*.

7.1 USB-Schnittstellenzugriff auf den *Hokuyo URG-04LX* Laserscanner

Da die Android Anwendung auf den Laserscanner als USB-Device zugreift, ist die Zugriffsberechtigung auf das USB-Device anzufordern (Abbildung 7.3).

Beim Aufruf der *requestPermission()* Funktion (Zeile 10) wird die Zustimmung des Benutzers durch Bestätigung eines Nachrichten-Fensters verlangt. Ohne am Pandaboard angeschlossenen HDMI-Bildschirm und Maus oder Tastatur, sind die Eingabeereignisse zur Bestätigung über die Android-Konsole zu emulieren. Wahlweise erfolgt der Zugriff vom Host-PC auf die Konsole über die Hyper-Terminal Anwendung oder durch den Aufruf der Befehlszeile: "adb shell". Die Konsolenbefehle zum Emulieren der Eingabeereignisse lauten:

1. Entfernen der Tastatursperre: "input keyevent 82"
2. Aktivieren der 'Use by default' Box: "input keyevent 66"
3. Zweif Tabulator-Ereignisse zum Markieren der 'OK' Schaltfläche: "input keyevent 61
input keyevent 61"
4. Betätigen der 'OK' Schaltfläche: "input keyevent 66"


```

1 import android.hardware.usb.*; // USB Host API (since Android 3.1)
2
3 // Used to enumerate and communicate with connected USB devices
4 UsbManager usbManager = (UsbManager) getSystemService(Context.USE_SERVICE);
5 // Represents a connected USB device
6 UsbDevice usbDevice = null;
7 // Find the Hokuyo-LaserScanner (USB-VendorId: 5585)
8 for( UsbDevice usbd : usbManager.getDeviceList().values() ) {
9     if( usbd.getVendorId() == 5585 && (usbd.getProductId() == 0) ) {
10         usbManager.requestPermission( usbd, PendingIntent.getBroadcast(this, 0,
11             new Intent("com.android.example.USE_PERMISSION"), 0) );
12         usbDevice = usbd; // Found Hokuyo-LaserScanner
13     }
14 }
15 // Represents a connection to the device, which transfers data on endpoints.
16 UsbDeviceConnection usbCon = usbManager.openDevice(usbDevice);
17 // Request an USB interface, which defines a set of functionality
18 usbCon.claimInterface(usbDevice.getInterface(1), true);
19 // An interface can have one or more endpoints,...
20 UsbEndpoint usbOUT = usbDevice.getInterface(1).getEndpoint(0);
21 // ...and has input and output endpoints for two-way communication
22 UsbEndpoint usbIN = usbDevice.getInterface(1).getEndpoint(1);
23 // M:continuous, D:3byte-encoding, start:128, end:640, clustercount:1, skip:0
24 byte[] startScanMScommand = ("MD0128064001000\n").getBytes();
25 // Performs a bulk transaction on the given endpoint
26 usbCon.bulkTransfer(usbOUT, startScanMScommand, startScanMScommand.length, 0);

```

Abbildung 7.3: Quellcode-Ausschnitt mit dem Ablauf des initialen USB-Laserscanner Zugriff.

Gemäß der USB 2.0 Spezifikation verfügt ein USB-Device über mindestens ein oder mehr *Interfaces*, die voneinander unabhängige Funktionalitäten bereitstellen und jeweils mindestens einen bidirektionalem *Control-Endpoint* mit der Adresse 0 zur Kommunikation besitzen. Den weiteren USB-Endpunkten ist die Richtung ihrer Datenübertragung aus Sicht des USB *Host-Controller* bzw. der OMAP4430-Plattform, entweder lesend (IN) oder schreibend (OUT) fest zugeordnet. Es existieren vier Typen von Endpunkten:

- **Isochron:** Transport der Daten erfolgt mit festgelegten, garantierten Datenrate, die vom *Host-Controller* reserviert wird. Gegenwärtig unterstützt die Android-API keinen Zugriff auf isochrone USB-Endpunkte.
- **Interrupt:** Diese Endpunkte teilen dem *Host-Controller* mit nach welchem maximalen Zeitabstand die nächste Abfrage nach neuen Daten erfolgen soll. Sie dienen bei Eingabegeräten zur Übertragung von bis zu 64 Byte großen Datenpaketen mit bestimmaren Verfügbarkeitszeitpunkten.

- **Control:** Die initiale Kommunikation beim Anschließen eines USB-Device zum Bereitstellen der Informationen über weitere Endpunkte, erfolgt über diesen Endpunkt. Jeder Datenaustausch wird in beide Richtungen bestätigt und im Fehlerfall wiederholt.
- **Bulk:** Der Laserscanner besitzt für jede Transportrichtung jeweils einen dieser niedrig-priorisierten und prüfsummengesicherten Endpunkte und die Datenübertragung erfolgt gemäß des SCIP 2.0 Protokolls im ASCII-Zeichenformat.

7.2 Verwendung des seriellen SCIP 2.0 Kommunikationsprotokolls

Der Laserscanner URG-o4LX verwendet im Auslieferungszustand als Kommunikationsprotokoll das serielle und Zustands-lose SCIP 1.1 Protokoll. Für das SoC-Fahrzeug wurde der Laserscanner durch ein Firmware-Update mit der SCIP 2.0 Version ausgestattet, da durch das Update die bisherigen vier Kommandos mit neun zusätzlichen Kommandos ergänzt wurden. Der Laserscanner genügt dem USB "Communication Device Class" (CDC) Standard, so dass keine Betriebssystem-spezifischen Treiber zum Senden und Empfangen der Befehle notwendig sind. Der Laserscanner und die OMAP4430-Plattform tauschen vordefinierte Kommandos aus, die in einem festgelegten Kommunikationsformat übermittelt werden (Abbildung 7.4).

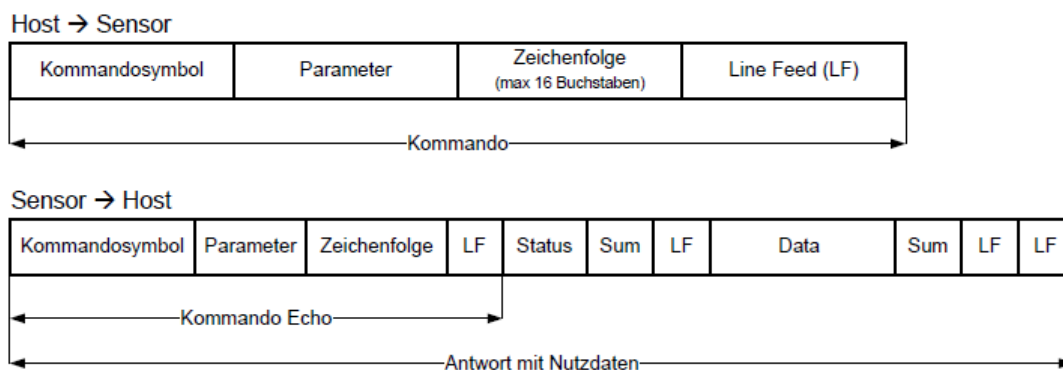


Abbildung 7.4: Aufbau des eingesetzten SCIP 2.0 Kommunikationsformats [28].

Die OMAP4-initiierte Kommunikation beginnt immer mit einem 2 Byte großen Kommandosymbol gefolgt von weiteren Parametern, wobei ein oder zwei abschließende Linefeeds das Kommunikationsende kennzeichnen. Empfängt der Laserscanner ein Kommando, wird dieses durch ein Echo des gesendeten Pakets und durch die angeforderten Nutzdaten bestätigt. Die Anzahl der maximal 64 Byte großen Nutzdaten-Blöcke ist abhängig von der im Kommando

konfigurierten Menge an Entfernungswerten und deren Kodierung. Durch einen internen Puffer im Laserscanner werden sequentiell gesendete Kommandos zwischengespeichert und durch ein FIFO-basiertes Verfahren nacheinander verarbeitet. Von der OMAP4430-Plattform empfangende Nutzdaten werden ebenfalls intern zwischengespeichert und der Objekterkennungssoftware in bis zu 32 KByte großen Paketen zur Verfügung gestellt.

Der Laserscanner URG-o4LX überträgt die Entfernungswerte mit einer Auflösung von 1mm, wofür zur Datenreduktion und zur Minimierung der Übertragungszeit eine Kodierung eingesetzt wird. Die 2Byte Kodierung hat eine maximale Länge von 12Bit und die 3Byte Kodierung eine maximale Länge von 18 Bit. Die Kodierung erfolgt durch die Trennung der oberen und unteren 6 Bits und dem anschließenden Addieren von 30H, was die kodierten Daten in den darstellbaren Bereich des ASCII-Formats verschiebt (Abbildung 7.5).



Abbildung 7.5: Kodierung der übertragenden Entfernungswerte mit 2 Byte oder 3 Byte [28].

Für die Objekterkennung werden die angeforderten Entfernungswerte mit der 3 Byte Kodierung übertragen, die gegenüber der 2 Byte Kodierung Entfernungsmesswerte von über 4.095mm zulässt. Zum Einschalten des Laserscanners, Konfigurieren und Starten der Entfernungsmessung sendet die OMAP4430-Plattform einmalig eine ASCII kodierte Zeichenkette, die mit den Buchstaben M und D eingeleitet wird (Abbildung 7.6).

Sind die gewünschten Messwerte größer als 64 Byte, dann werden mehrere Nutzdaten-Blöcke versendet, wobei im letzten durch zwei aufeinanderfolgende *Line Feed* das Ende der Übertragung gekennzeichnet wird. Es folgt die Beschreibung der Packet-Inhalte:

"M" (4dH) - Vereinigt die Funktionalität von Einschalten des Laserscanners (BM-Kommando) und Anfordern von Messdaten (GD-Kommando).

"D" (44H) - Abstandswerte werden mit 3Byte und nicht mit 2Byte (MS-Kommando) kodiert, so dass auch Abstandswerte bis zu 5.600mm übermittelt werden.

Starting Step / End Step - Liegen im Intervall von 44 und 725. Der Endschritt muss stets größer sein als der Startschritt. Es ist darauf zu achten, dass die natürlichen Zahlen ebenfalls als

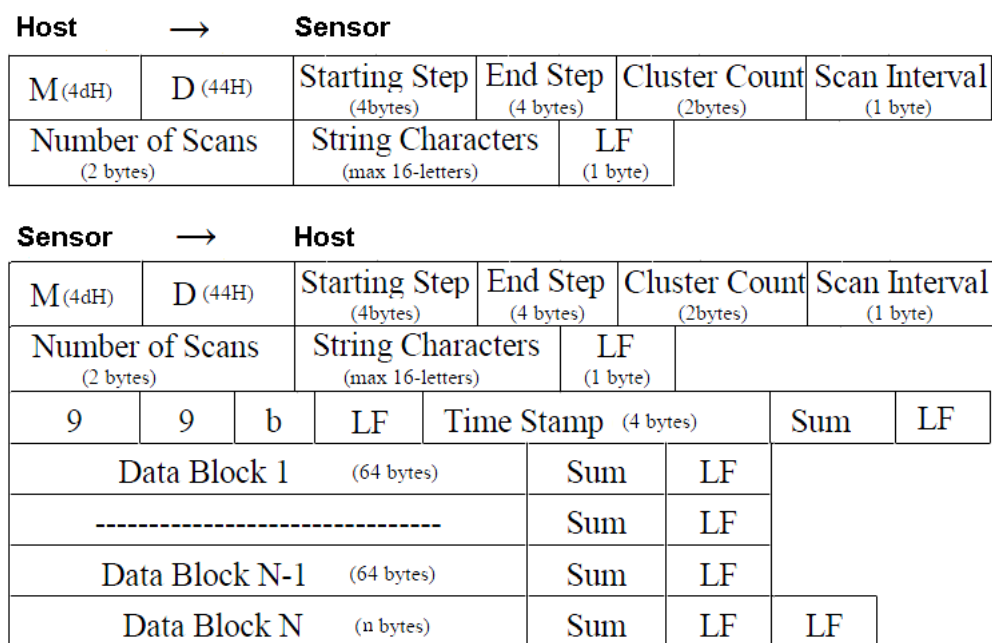


Abbildung 7.6: Ablauf des Anfordern von Messwerten mit Aufbau des MD-Kommandos und der Antwort vom USB-Laserscanner.

ASCII Äquivalent versendet werden, da sonst der Laserscanner diese nicht interpretieren kann. Für die Objekterkennungssoftware werden jeweils "0128" (30H, 31H, 32H, 38H) und "0640" (30H, 36H, 34H, 30H) verwendet, so dass der 180° Frontbereich des SoC-Fahrzeugs abgedeckt wird.

Cluster Count - Bestimmt die Winkelauflösung des Laserscanners und dient der Datenreduktion, wobei z.B. bei einem Cluster Count von "03" der geringste von drei aufeinanderfolgenden Messwerten als Abstand gesendet wird und sich eine Winkelauflösung von $0,35^\circ * 3 = 1,05^\circ$ ergibt. Um die maximale Winkelauflösung des Laserscanners zu nutzen wurde in der Software-Implementierung "01" (30H, 31H) gewählt.

Scan Interval - Gibt die Anzahl der zu überspringenden Rotierungen des Laserscanners an, so dass sich eine Scanperiode von $100ms * ScanInterval$ ergibt. Zur Nutzung der maximalen Scanfrequenz von 10Hz wird dieser Wert auf "0" (30H) gesetzt.

String Character - Ist ein freies 16 Byte großes Feld, welches beispielsweise zur Vergabe von Sequenznummern verwendet wird. Bei der Implementierung wird dieses Feld nicht genutzt, da die Kommandos sequentiell gesendet und somit auch geordnet verarbeitet werden.

LF - Zeilenvorschub *Line Feed* zur Formatierungs- und als Trennzeichen, welches durch die Escape-Sequenz "\n" (0AH) implementiert wird.

"99b" (39H, 39H, 62H) - Status, welcher zur fehlerfreien Empfangsbestätigung im Echo vom Laserscanner auf "oob" oder zur Kennzeichnung von nachfolgenden Messdaten auf "99b" gesetzt wird. Andere Werte entsprechen Fehlercodes wie z.B. eine ungültige Parameterwahl oder einer Hardware-Fehlfunktion des Laserscanners.

Time Stamp - Jedes Nutzdaten-Paket enthält einen vom Laserscanner-internen Timer gesetzten 4Byte-Zeitstempel mit einer Auflösung von 1ms, der nach dem gleichen Kodierungsschema der Abstandswerte (vgl. Abb.7.5) maximal 24Bit lang ist.

Sum - Checksumme, die durch das Addieren der Bytes seit dem letzten *Line Feed* berechnet wird. Die niederwertigsten 6 Bits dieser Summe werden ebenfalls nach dem Kodierungsschema der Abstandswerte in ein ASCII-Zeichen konvertiert wird.

Die Größe eines Paketes mit Roh-Messdaten beträgt bei 512 Abstandswerten unter Verwendung der 3 Byte-Kodierung 1.535 Bytes, die vom Laserscanner alle 100 Millisekunden gesendet werden. Durch den Paket-Header mit den durch das SCIP 2.0 Protokoll definierten Trennzeichen innerhalb des Paketes wächst die Größe auf 1.616 Bytes.

7.3 Objekterkennung mit RANSAC

Bei der Objekterkennung wird zur Datenreduktion jedes Segment in einem Segmentierungsdurchlauf durch den ersten (*First*) und letzten (*Last*) Begrenzungspunkte sowie dem Punkt mit der geringsten Entfernung (*Nearest*) zum Hindernis identifiziert (Abbildung 7.7).

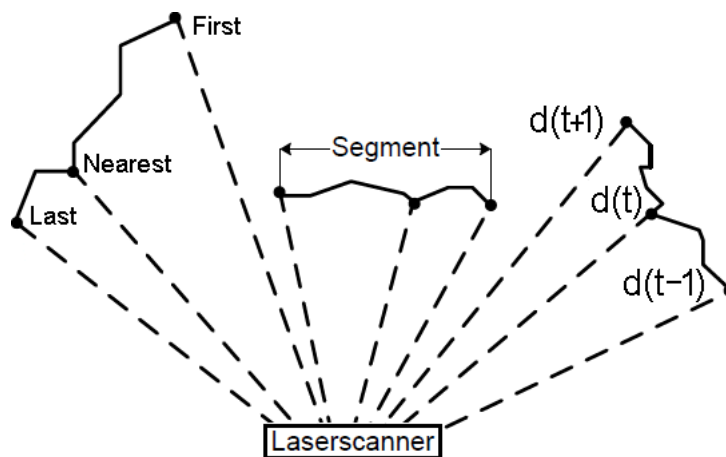


Abbildung 7.7: Bei der Identifizierung der Segmente durch drei Punkte wird jedem Punkt ein Scanschritt t und der dazugehörige Abstandswert d_t zugeordnet

Eine vollständige Segmentierung der Umgebung innerhalb eines Scanvorgangs ist gegeben, wenn jeder Abstandswert eindeutig einer zusammenhängenden Regionen zugeordnet ist. Der aktuell gemessene Distanzwert d_t eines Scanschritts $t \in [0; 512]$ wird sukzessive mit der Distanz des vorigen Abstandswertes d_{t-1} verglichen. Benachbarte Abstandswerte gehören zum gleichen Segment, genau dann wenn der Betrag der Differenz der beiden Messwerte $|d_t - d_{t-1}|$ einen Schwellwert $s(d_t)$ nicht überschreiten (Abbildung 7.8).

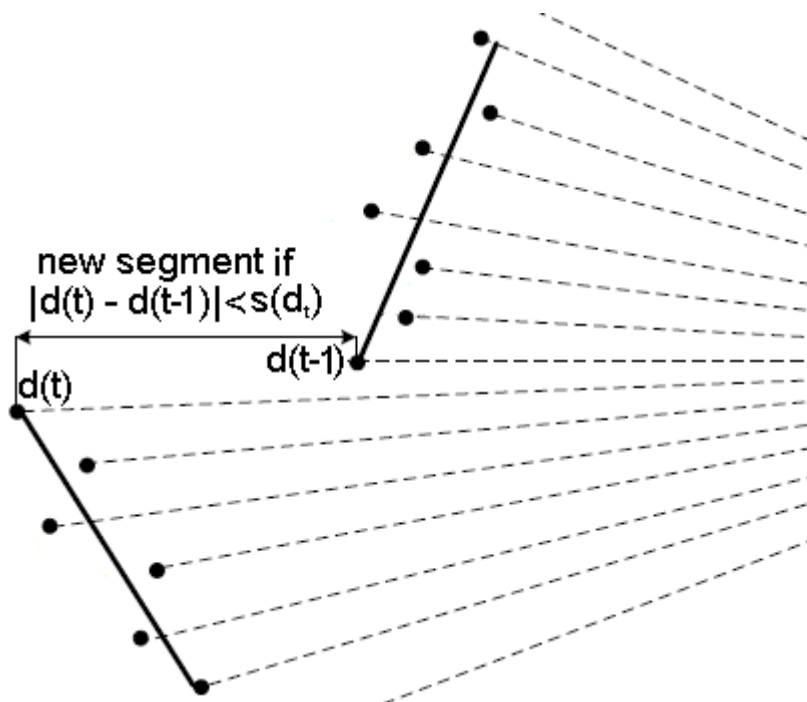


Abbildung 7.8: Generierung der Segmente durch Vergleichen der absoluten Differenz zweier aufeinanderfolgender Abstandswerte $|d_t - d_{t-1}|$ mit einem Schwellwert $s(d_t)$ in Abhängigkeit der aktuell gemessenen Scanschritts t .

Die Wahl eines statischen Schwellwertes in der Vorarbeit führte dazu, dass mehrere zusammenstehende Objekte in Entfernungen von unterhalb 1,5 Meter als ein zusammenhängendes Segment identifiziert wurden und ein einzelnes Objekt, das sich mehr als 2,5 Meter entfernt befindet, auf mehrere Segmente aufteilt wurde. Im Ausblick der Vorarbeit wird ein Anfangsschwellwert s von $20mm$ mit zunehmender Entfernung d in diskreten 0,5 Meter Schrittweiten im Durchschnitt um $16,5mm$ erhöht (vgl. Abb. 2.3). Für die kontinuierliche Berechnung des Schwellwertes erfolgt eine lineare Interpolation durch die entsprechenden Entfernungswerte. Der aktuell gemessene Abstandswert d_t gehört genau dann zum Segment des vorigen Wertes

d_{t-1} wenn der Betrag ihrer Differenz kleiner ist als der berechnete Schwellwert $s(d_t)$ für den sich die folgende Geradengleichung ergibt:

$$s(d_t) = m_{\Delta} \cdot d_t + b_{offset} = \frac{\Delta s_{average}}{\Delta d_{const}} \cdot d_t + s_0 = \frac{16,5mm}{500mm} \cdot d_t + 20mm = 0,033 \cdot d_t + 20mm$$

Die Definitionsmenge der Geradengleichung ergibt sich dabei aus den Messwertbereich des Laserscanners zu $d_t \in [20mm; 5.600mm]$ und damit für den Schwellwert die Abbildungsmenge $s(d_t) \in [20,66mm; 184,8mm]$. Mit der zusätzlichen Bedingung, dass ein detektiertes Segment aus mindestens vier Messwerten besteht, sind die 512 Abstandswerte in maximal 128 Segmente gruppierbar und die Daten-Arrays entsprechend dimensioniert.

"RANDOM SAMPLE CONSENSUS" (RANSAC)

Als Eckpunkte eines Dreiecks angewendet, erscheint jedes segmentierte Objekt aus Sicht des Fahrzeugs Keil-förmig. Die drei Punkte sind somit die Modellparameter, der zum Fahrzeug gerichteten Hindernisfront, die es auszuweichen gilt (Abbildung 7.9).

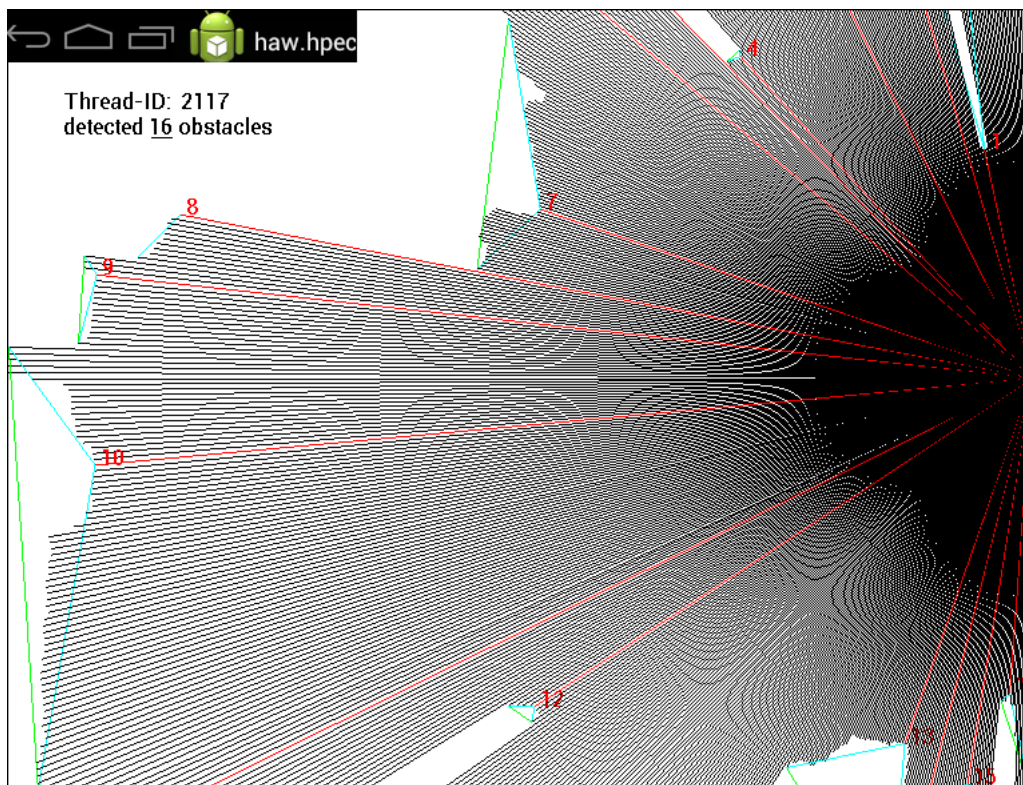


Abbildung 7.9: Visualisierung eines Auswertungszyklus mit der in rot dargestellten Segment-Identifikationsnummer und dem jeweils dichtesten Abstandswert der erkannten Objekten. Die grüne Linie verläuft jeweils vom ersten bis zum letzten erkannten Segmentabstandswert.

Das RANSAC [8] Ausgleichsverfahren dient zur Bereinigung der Modellparameter von Messdaten-Ausreißern. Die Annahme, dass die drei ermittelten Werte keine Ausreißer sind wird überprüft, indem die Distanz aller Messwerte eines Segmentes zu den beiden Geradenabschnitten des Modells bestimmt wird. Je dichter die Messwerte an der berechneten Hindernisfront liegen, desto wahrscheinlicher enthielten zur Modellberechnung gemessenen Werte keine Ausreißer. Zum Vergleich werden pro Segment drei weitere dieser sog. *Consensus Sets* berechnet: Einmal mit dem zweiten Segmentpunkt anstelle des ersten, mit dem vorletzten Segmentpunkt anstelle des letzten und abschließend mit sowohl mit Verwendung des zweiten und vorletzten zusammen. Aus diesen vier Keil-förmigen Hindernismodellen wird das *Consensus Set* ausgewählt, dessen Summe über die Distanzen der Abstandswerte zu den Geradenabschnitten am geringsten ist (Abbildung 7.10).

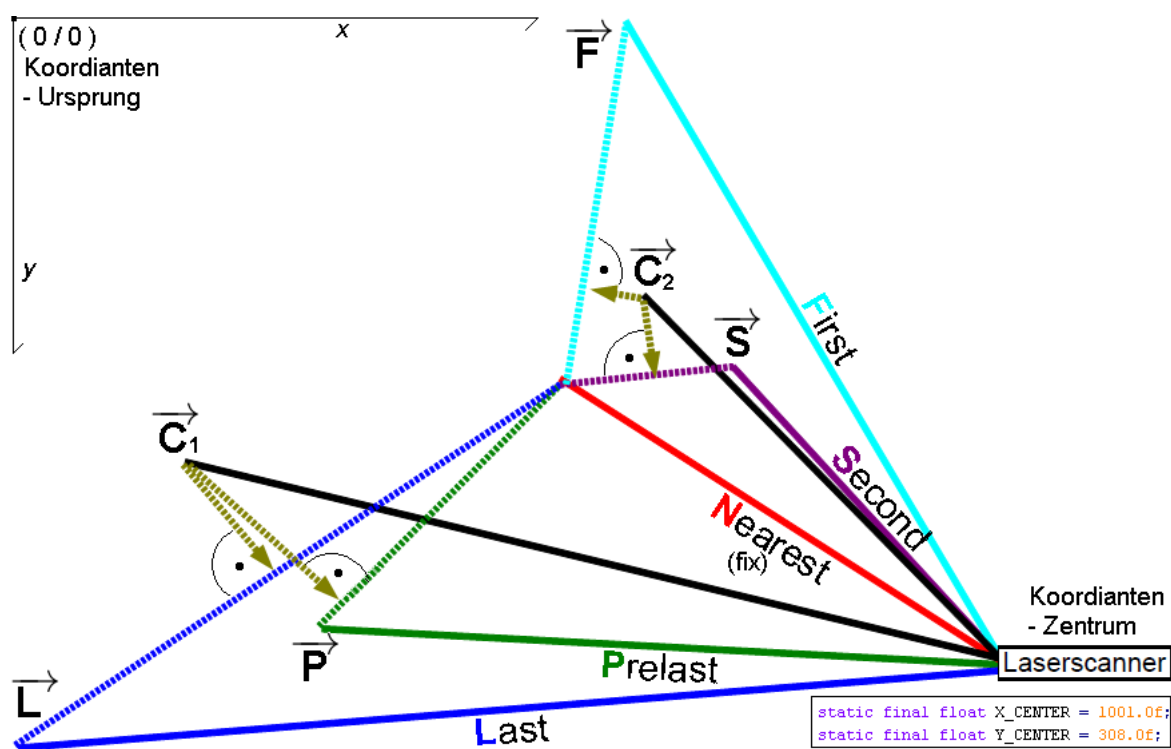


Abbildung 7.10: Ein Segment wird beschrieben durch den jeweils dichtesten zugehörigen Abstandswert (rot) und zwei Geraden, die vom jeweils dichtesten Abstandswert einmal zum ersten (hellblau) oder zweiten (magenta) und einmal zum letzten (dunkelblau) oder vorletzten (grün) Abstandswert verlaufen. Ausgewählt wird das Geraden-Paar mit dem geringsten euklidischen Distanzwerten (gelb) zu den gemessenen Abstandswerten (schwarz) innerhalb eines Segments.

Die Auswahl der beiden Randpunkte eines Segmentes sind störungsanfällig und wirken sich signifikant auf das resultierende Dreieck-Modell aus. Zur Stabilisierung der detektierten Seg-

mente wird mit geprüft, ob die Auswahl des zweiten statt ersten und vorletzten statt letzten Segmentpunktes zu einem *wahrscheinlicherem* Modell des Hindernis führt. Auf jedes identifizierte Segment wird das RANSAC-Verfahren angewendet:

1. Berechne die Gerade vom ersten *First*- zum dichtesten *Nearest*-Segmentpunkt und summiere die Abstände aller Messpunkte innerhalb des Segmentes von dieser Geraden.
2. Wiederhole Schritt 1 dreimal unter Austausch des ersten Segmentpunktes jeweils durch den zweiten *Second*-, letzten *Last*- und vorletzten *Prelast*-Segmentpunkt.
3. Wähle als endgültige Modellparameter für den ersten Segmentpunkt aus *First* und *Second* den mit der geringeren Abstandssumme und für den letzten Segmentpunkt aus *Prelast* und *Last* ebenfalls den mit der geringeren Abstandssumme.

Der Abstand zwischen einem Messpunkt und einer der vier ermittelten Geraden bzw. Hindernisfront berechnet sich aus der euklidischen Distanz, die von dem Vektor gespannt wird, der vom Messpunkt $\vec{C} := (C_x/C_y)$ senkrecht auf den Vektor \vec{TI} zeigt, der parallel zur Hindernisfront-Geraden ausgerichtet verläuft und die Gerade somit repräsentiert:

$$\vec{TI} := \vec{N} - \vec{I} \quad \text{mit} \quad I \in \{F, S, P, L\}$$

Das Skalarprodukt, definiert für zwei Dimensionen durch: $\vec{A} * \vec{B} = (A_x \cdot B_x) + (A_y \cdot B_y)$, aus diesem noch unbestimmten Vektor, der senkrecht auf den Richtungsvektor \vec{TI} zeigt, beträgt dann genau 0:

$$(\vec{C} - (\vec{N} + \vec{TI} \cdot K_I)) * \vec{TI} = \vec{0}$$

Durch Auflösen der Vektor-Gleichung in X/Y Koordinatensystem-Komponenten und anschließendem Umstellen der Gleichung nach dem Faktor K_I , ergibt dieser sich zu jedem der 512 Abstandswert-Koordinatenpaare (C_x/C_y) als die Lösung der vier Varianten des folgenden Gleichungssystems:

$$K_I = \frac{C_x \cdot TI_x + C_y \cdot TI_y - (TI_x \cdot N_x) - (TI_y \cdot N_y)}{TI_x^2 + TI_y^2} \quad \text{mit} \quad I \in \{F, S, P, L\}$$

Der Abstand D_I zwischen einem Messwert und einer der vier Geraden beträgt dann:

$$D_I = \sqrt{(C_x - (K_I \cdot TI_x + N_x))^2 + (C_y - (K_I \cdot TI_y + N_y))^2} \quad \text{mit} \quad I \in \{F, S, P, L\}$$

7.4 Mutex-basiertes Thread-Synchronisationskonzept

Mit einer Periodendauer von 100 Millisekunden liefert der *Hokuyo*-Laserscanner neue Messwerte. Der blockierende USB-Lesevorgang zur Messdatenerfassung erfolgt auf einem CPU-Kern, der vom Betriebssystem-Scheduler verwaltet wird und anderen Anwendungen zur Verfügung steht bis die Messdaten im USB-Empfangspuffer vorliegen. Parallel dazu führt der andere CPU-Kern unter Verwendung des gemeinsamen Speichers die Objektsegmentierung und Visualisierung mit den aktuellen Messdatensatz durch. Daraus resultieren zwei kritische Regionen, deren Zugriffe jeweils durch einen SW-Mutex vom Typ:

`java.util.concurrent.locks.ReentrantLock`

auf genau einen Thread zu einem Zeitpunkt begrenzt wird (Abbildung 7.11).

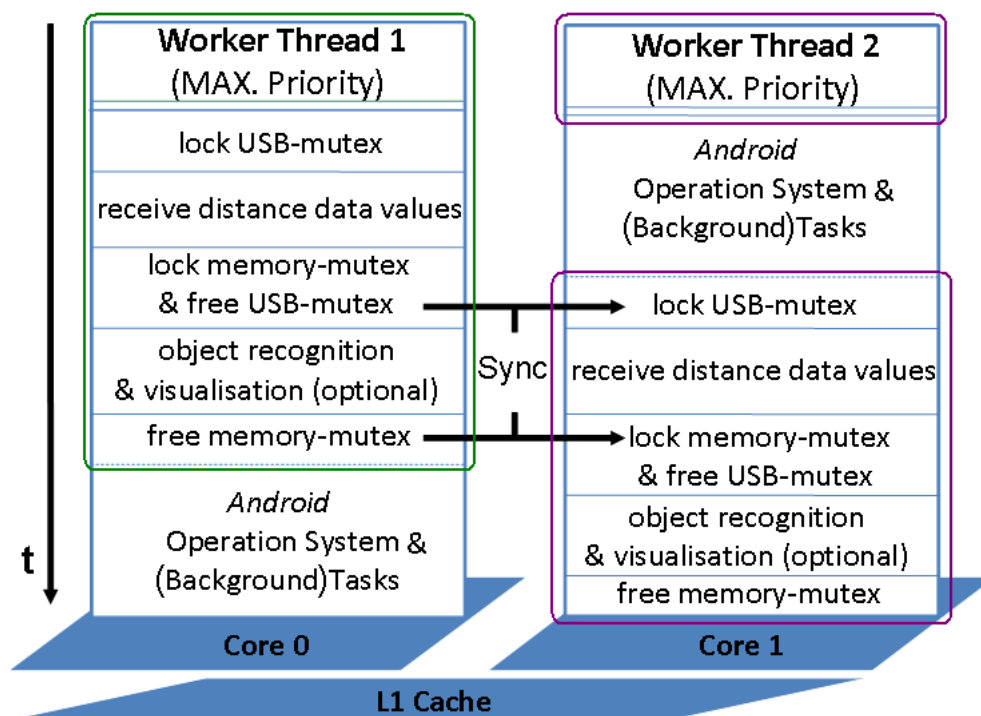


Abbildung 7.11: Partitionierungskonzept der Laserscanner-basierten Objekterkennung: Zwei Instanzen identischer Threads mit höchster Priorität synchronisieren sich über SW-Mutexe beim Zugriff auf die USB-Schnittstelle zur Messdatenerfassung und für die Objekt-Segmentierung auf dem gemeinsamen Speicher.

Die *HPECAActivity* Java-Klasse implementiert über ihren Methodenrumpf:

```
"public void run() {...}"
```

das *Runnable-Interface* und beschreibt darin das Verhalten eines *WorkerThreads*. Nach der Initialisierung des Laserscanners werden davon zwei Thread-Instanzen erzeugt und gestartet.

Diese weisen sich die höchste *LinuxKernel*-basierte Systemweite Priorität von -20 zu. Einem höchst-priorisiertem Thread garantiert Android eine CPU-Zuteilung wenn diese aus dem blockierenden in den aktiven Zustand wechseln. Dieser Wechsel erfolgt sobald neue Entfernungswerte lese-bereit vorliegen. Bis zum Abschluss der Objektsegmentierung mit diesen Messdaten bleibt ein *WorkerThread* ohne Unterbrechung aktiv und ihm wird aufgrund der Priorität der zugeteilte CPU-Kern nicht von einem Betriebssystemprozess entzogen.

Durch Aufrufen der Android-API [24] Funktion:

`android.os.SystemClock.elapsedRealtime()`

wird die durchschnittliche Dauer eines vollständigen Segmentierungsdurchlaufs mit Visualisierung in Millisekunden gemessenen (Tabelle 7.1).

| | | |
|------------------------|--------------------|---------------------|
| MIT RANSAC | 2 Segmente | 12 Segmente |
| 1 WorkerThread | 25 +/- 2 ms | 50 +/- 5 ms |
| 2 WorkerThreads | 29 +/- 4 ms | 55 +/- 10 ms |
| OHNE RANSAC | 2 Segmente | 12 Segmente |
| 1 WorkerThread | 23 +/- 2 ms | 45 +/- 5 ms |
| 2 WorkerThreads | 27 +/- 4 ms | 50 +/- 10 ms |

Tabelle 7.1: Die durchschnittliche jeweils aus 100 Segmentierungsdurchläufen gemessene Dauer einer Segmentierung mit Angabe der Messstreuungen um den Durchschnittswert.

Die im RANSAC Verfahren von der *Vector Floating Point* Einheit durchgeführten Fließkomma-Arithmetik Berechnungen führen demnach zu einer Verzögerung, die innerhalb der Mess-toleranz liegt. Unabhängig von der Anzahl der identifizierten Objekte beträgt die Dauer für Dekodierung und Visualisierung der 512 Abstandswerte konstant 20 Millisekunden. Start- und Endpunkt der Zeitmessung schließen die Mutex-Operationen mit ein. Beim Einsatz eines *WorkerThreads* ist die Segmentierungsdauer ohne Synchronisierung aufgrund der wegfallenden Operationen des Betriebssystems-Schedulers geringer. Ein `unlock()`-Aufruf hat dann keine Thread-übergreifende Auswirkung zum korrespondierenden `lock()`-Aufruf des Mutex.

8 Zusammenfassung und Ausblick

Mit *Texas Instruments* "Open Multimedia Application Platform 4430" als "Symmetrische Multiprozessorsystem" MPSoC-Plattform wurde die vom *Hokuyo URG-04LX* Laserscanner maximal-erreichbare Messgenauigkeit bei der Portierung eines 3-Punkte Schwellwertverfahrens zur Segmentklassifizierung eingesetzt. Der 1 GHz Cortex-A9 Zweikern-Prozessor mit "Vector Floating Point" Einheit gehört zur siebten Version der ARM Prozessorarchitektur und bietet der implementierten Objekterkennungs-Software Assembler-Instruktionen zur virtuellen Speicher-verwaltung durch eine "Memory Management Unit" und unterstützt mit der "Snoop Control Unit" die Interprozessor-Synchronisation unter Erhaltung der Level1-Cache Kohärenz.

Zur Ergebnis-Visualisierung der erkannten Umgebungsobjekte entlastet die *PowerVR-SGX540* "Graphics Processing Unit" den Cortex-A9 bei der Darstellung und beschleunigt diese durch paralleles Ausführen der Zeichen-Operationen. Die Software-Implementierung erfolgte als *Android-App* unter dem Android 4.0.3 Betriebssystem mit modifiziertem Linux-Kernel von *Linaro*, welches in einem mehrstufigen Bootvorgang von einer präparierten SD-Karte geladen wird und die OMAP4-Peripherie initialisiert. Zur SW-Entwicklung und Fehlersuche wurde die *Eclipse* Entwicklungsumgebung auf einem *Windows* Host-PC durch Installieren des "Android Development Tools" Plugins von *Google* erweitert.

Wahlweise über die USB- oder Ethernet-Schnittstelle wird die Anwendung in Bytecode für die "Dalvik Virtual Machine" kompiliert, auf die OMAP4430-Plattform übertragen und dort ausgeführt. Ein externer AVR- μ C wird zur PWM-Signalgenerierung über einen I²C-Bus durch den Aufruf *nativer* C-Bibliotheks Anweisungen angesteuert. Im Lebenszyklus-Einstiegspunkt der "High Performance Embedded Computing" *Aktivität* erfolgt die initiale Laserscanner-Kommunikation über die USB 2.0 Schnittstelle als ASCII-kodierte *bulk-endpoint* Transaktion gemäß des seriellen SCIP 2.0 Protokolls.

Zwei höchst-priorisierte *WorkerThread* Instanzen synchronisieren sich über SW-Mutexe, erfassen die Messdaten vom Laserscanner im 100 Millisekunden Intervall und gruppieren die dekodierten Abstandswerte in zusammenhängende Segmentregionen. Anfang und Ende eines so zu identifizierten Objektes werden durch Vergleich benachbarter Abstandswert-Differenzen

mit einem Schwellwert bestimmt, der kontinuierlich in Abhängigkeit der aktuell gemessenen Entfernung berechnet wird. Das durch den ersten, letzten und dichtesten Messpunkt eines detektierten Segmentes verlaufende Dreieck approximiert das Hindernis-Modell. Mit dem "Random Sample Consensus" (RANSAC) Verfahren erfolgt zur Stabilisierung des resultierenden Modells die Auswahl aus vier Anfangs- und Endpunkt-Paaren. Die Summe über alle Distanzen zwischen den Messpunkten innerhalb eines Segments zur Hindernisfront-Geraden wird dabei als Kriterium für den Gütegrad des Modells verwendet. Jeweils die Gerade mit dem geringeren Summenwert wird aufgrund der höheren Messwert-Übereinstimmung als endgültiges Modell ausgewählt.

Ausblick

Geplant ist die Integration der Laserscanner-basierten Objekterkennung mit der autonomen Fahrspurführung (Abbildung 8.1) unter einem Hardware/Software Co-Design [22] auf die *Zynq-7000 Extensible Processing Platform* [55], auf der sich neben FPGA-Ressourcen zur Software-Ausführung der baugleiche *Cortex-A9* Zweikernprozessor der OMAP4430-Plattform befindet (Anhang D).

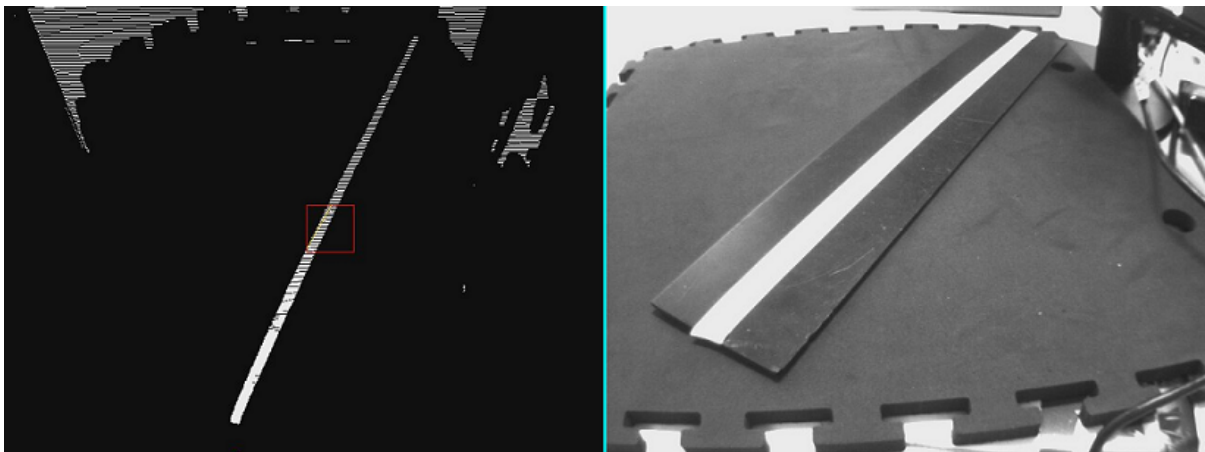


Abbildung 8.1: Kameraausgabe aus der Fahrspurführungs-Pipeline auf der "Zynq-7000 All Programmable SoC" Plattform (rechts) und nach Hough-Transformation zur Geradenerkennung mit Binarisierung und projektiver Transformation zur Korrektur der perspektivischen Verzerrung (links).

Für Betriebssystem-unabhängige Debug- und Überwachungsfunktionalitäten über ein JTAG-Interface, die bis auf CPU-Registerebene reichen, wird das "ARM Development Studio 5" (DS-5) als Software-Entwicklungsumgebung eingesetzt (Abbildung 8.2).

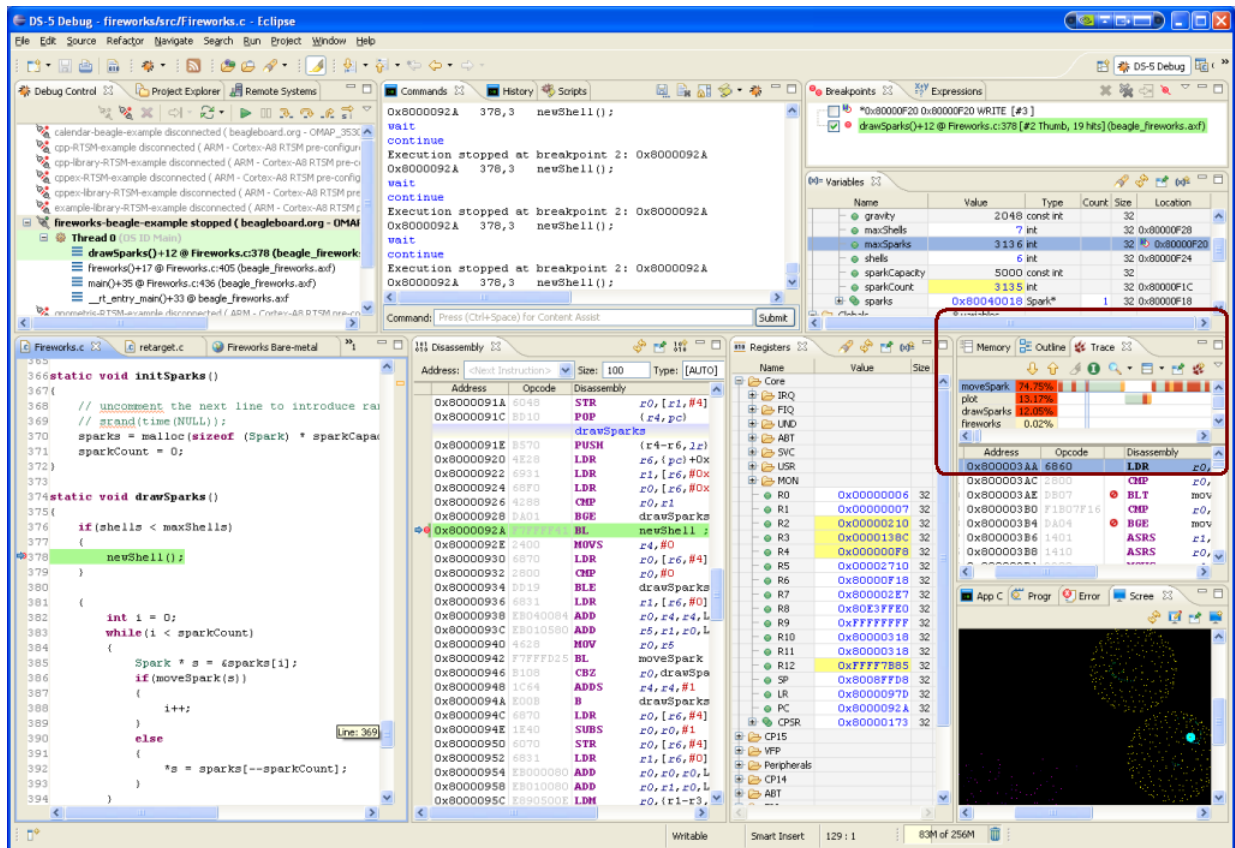


Abbildung 8.2: Im Trace-View (rot) der Eclipse-basierten DS-5 Tool-Chain wird die Simulationszeit prozentual dargestellt, die der Cortex-A9 anteilig für die in C- oder Assembler-Code geschriebene (Unter-)Funktionen und Interrupt Service Routinen einer Animation verbraucht. [6]

Literaturverzeichnis

- [1] ANDRESEN, Erik: *Inbetriebnahme einer ARM Cortex-A9 Dualcore-Plattform und Erzeugung von PWM-Signalen für die Ansteuerung von Aktoren*, Hochschule für Angewandte Wissenschaften Hamburg, Projektbericht, 2012
- [2] ARM® LIMITED: *Cortex-A9 MPCore Technical Reference Manual*. Website. 2010. – URL http://infocenter.arm.com/help/topic/com.arm.doc.ddi0407f/DDI0407F_cortex_a9_r2p2_mpcore_trm.pdf. – Zugriffsdatum: 27.08.2012
- [3] ARM® LIMITED: *Cortex-A9 Technical Reference Manual*. Website. 2010. – URL http://infocenter.arm.com/help/topic/com.arm.doc.ddi0388f/DDI0388F_cortex_a9_r2p2_trm.pdf. – Zugriffsdatum: 17.08.2012
- [4] ARM® LIMITED: *ARM Architecture Reference Manual*. Website. 2011. – URL <https://silver.arm.com/download/download.tm?pv=1164925>. – Zugriffsdatum: 18.08.2012
- [5] ARM® LIMITED: *Cortex-A Series Programmer's Guide*. Website. 2011. – URL <https://silver.arm.com/download/download.tm?pv=1143190>. – Zugriffsdatum: 19.08.2012
- [6] ARM® LIMITED: *ARM Development Studio 5*. Website. 2012. – URL <http://www.arm.com/products/tools/software-tools/ds-5/index.php>. – Zugriffsdatum: 15.11.2012
- [7] ATMEL® CORPORATION: *AVR AT90CAN128 Produktseite*. Website. 2012. – URL <http://www.atmel.com/devices/at90can128.aspx>. – Zugriffsdatum: 24.11.2012
- [8] BAKARDZHIEV, Angel K.: *Objektbox-Detektion auf Basis von Laserscannerdaten*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2012
- [9] BORGES, G.A. ; ALDON, M.: *Line Extraction in 2D Range Images for Mobile Robotics*. 2006
- [10] BURKHARDT, T. ; FEINÄUGLE, A. ; FERICEAN, S. ; FORKL, A.: *Lineare Weg- und Abstandssensoren: Berührungslöse Messsysteme für den industriellen Einsatz*. Süddeutscher Verlag, 2004. – ISBN 978-3-93788-907-8

- [11] CNXSOFT - EMBEDDED SOFTWARE DEVELOPMENT: *Linaro Android Puts Stock Android To Shame on TI Pandaboard (OMAP4430)*. 2012. – URL <http://www.cnx-software.com/2012/06/03/linaro-android-puts-stock-android-to-shame-on-ti-pandaboard-omap4430>. – Zugriffsdatum: 23.10.2012
- [12] DAIMLER CHRYSLER®: *Systembeschreibung ETC Deutschland*. DCSMM / GE. 2003. – URL <http://www.dcl.hpi.uni-potsdam.de/teaching/mobilitySem03/slides/tollcollect.pdf>. – Zugriffsdatum: 03.08.2012
- [13] DAMM, Werner ; ACHATZ, Reinhold ; BEETZ, Klaus ; BROY, Manfred ; DAEMBKES, Heinrich ; GRIMM, Klaus ; LIGGESMEYER, Peter: *Nationale Roadmap Embedded Systems*. Springer, 2010. – 67–136 S. – ISBN 978-3642149016
- [14] DELVARE, Jean: *Linux I2C dev-Interface*. Website. 2012. – URL <http://www.kernel.org/doc/Documentation/i2c/dev-interface>. – Zugriffsdatum: 27.11.2012
- [15] DIN 70000: *Straßenfahrzeuge, Fahrzeugdynamik und Fahrverhalten, Begriffe (ISO 8855:1991, modifiziert)*. 1994
- [16] DIN VDE 0848 - TEIL 2: *Sicherheit in elektromagnetischen Feldern - Schutz von Personen im Frequenzbereich 30 kHz bis 300 GHz*. 1991
- [17] EICHLER, J.: *Laser: Bauformen, Strahlführung, Anwendungen*. Springer, 2010. – ISBN 978-3-64-210461-9
- [18] FRAUNHOFER-INSTITUT FÜR LASERTECHNIK: *Presseinformation - Femtosekundenlaser*. 2011. – URL http://www.ilt.fraunhofer.de/content/dam/ilt/de/documents/Pressemeldungen/pm2011/PM_fs-Laser_fuer_MPQ.pdf. – Zugriffsdatum: 01.08.2012
- [19] FU, Guoqiang ; CORRADI, P. ; MENCIASSI, A. ; DARIO, P.: *An Integrated Triangulation Laser Scanner for Obstacle Detection of Miniature Mobile Robots in Indoor Environment*. Mechatronics, IEEE/ASME Transactions. 2011
- [20] FUERSTENBERG, K. ; SCHULZ, R.: *Laserscanner for Driver Assistance*. 3rd International Workshop on Intelligent Transportation, Hamburg, Ibeo Automotive Systems GmbH. 2006
- [21] GEDAK, Curtis: *Gnome Partition Editor*. Website. 2012. – URL <http://gparted.sourceforge.net>. – Zugriffsdatum: 07.12.2012

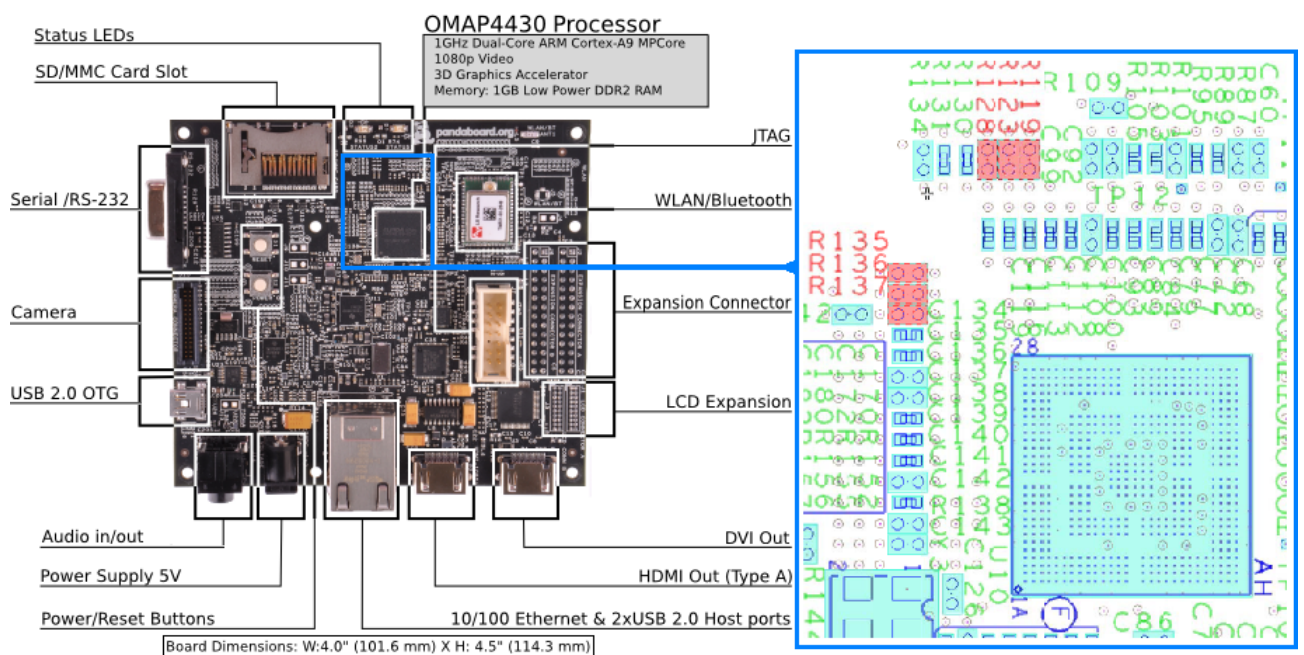
- [22] GÖHRINGER, Diana ; HÜBNER, Michael ; BENZ, Michael ; BECKER, Jürgen: A Design Methodology for Application Partitioning and Architecture Development of Reconfigurable Multiprocessor Systems-on-Chip. In: *18th IEEE Annual International Symposium* (2010)
- [23] GOOGLE® INC.: *Android Developer Tools*. Website. 2012. – URL <http://developer.android.com/sdk/>. – Zugriffsdatum: 17.12.2012
- [24] GOOGLE® INC.: *Android Developers*. Website. 2012. – URL <http://developer.android.com>. – Zugriffsdatum: 16.08.2012
- [25] GOULD, R. G.: *Light Amplification by Stimulated Emission of Radiation*. The Ann Arbor Conference on Optical Pumping. 1959
- [26] GSCHWANDTNER, M. ; KWITT, R. ; UHL, A. ; PREE, W.: *BlenSor: Blender Sensor Simulation Toolbox*. Springer, 2011. – ISBN 978-3-642-24030-0
- [27] HELLA KGAA HUECK® UND CO: *Technical Information Electronics - Driver Assistance Systems*. 2007. – URL http://www.hella.com/produktion/HellaUSA/WebSite/MiscContent/Download/AutoIndustry/Electronics/TI_ADAS_GB_TT_07.pdf. – Zugriffsdatum: 02.08.2012
- [28] HOKUYO AUTOMATIC® CO.: *Communication Protocol Specification For SCIP2.0*. 2006. – URL http://www.hokuyo-aut.jp/02sensor/07scanner/download/data/URG_SCIP20.pdf. – Zugriffsdatum: 30.08.2012
- [29] IBEO AUTOMOTIVE SYSTEMS® GMBH: *ALASCA User Manual*. 2006
- [30] IMAGINATION TECHNOLOGIES LIMITED®: *PowerVR Series5 IP Core*. Website. 2012. – URL http://www.imgtec.com/powervr/sgx_series5.asp. – Zugriffsdatum: 08.12.2012
- [31] K., Kneubühl F. ; W., Sigrist M.: *Laser*. Vieweg + Teubner, 2008. – ISBN 978-3-83-510145-6
- [32] KIRSCHKE, Marco: *FPGA-basierte MPSoC-Plattform zur Integration eines Antikollisions-systems in die Fahrspurführung eines autonomen Fahrzeugs*, Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, 2012
- [33] KUMAR, Akash ; CORPORAL, Henk ; MESMAN, Bart: *Multimedia Multiprocessor Systems: Analysis, Design and Management (Embedded Systems)*. Springer, 2012. – ISBN 978-9400733497
- [34] LINARO - OPEN SOURCE SOFTWARE FOR ARM SoCs: *Android 4.0.3 Landing Build*. 2012. – URL <http://releases.linaro.org/12.01/android/leb-panda>. – Zugriffsdatum: 07.12.2012

- [35] MOTOR NEWS: *Hella - Die automatische Abstandsregelung (ACC = Adaptive Cruise Control) erstmals in Serie*. 2007. – URL http://www.motornews.eu/cms/front_content.php?idcatart=4306. – Zugriffsdatum: 04.08.2012
- [36] NXP SEMICONDUCTORS®: *Level translating I2C-bus repeater PCA9517*. Website. 2012. – URL http://www.nxp.com/products/interface_and_connectivity/i2c/i2c_bus_repeaters_hubs_extenders/PCA9517AD.html. – Zugriffsdatum: 25.11.2012
- [37] ORACLE®: *VirtualBox*. Website. 2011. – URL <http://www.virtualbox.org>. – Zugriffsdatum: 22.09.2012
- [38] PANDABOARD: *PandaBoard System Reference Manual*. Website. 2011. – URL http://pandaboard.org/sites/default/files/board_reference/A1/Panda_Board_Spec_DOC-21010_REV0_6.pdf. – Zugriffsdatum: 28.08.2012
- [39] PASCOAL, J. ; MARQUES, L. ; ALMEIDA, A.T. de: Assessment of Laser Range Finders in risky environments. In: *Intelligent Robots and Systems. IEEE/RSJ International Conference (2008)*, S. 3533–3538
- [40] REIF, K.: *Sensoren im Kraftfahrzeug*. Vieweg + Teubner, 2010. – ISBN 978-3-83481-315-2
- [41] SEUSS, J.: *Sensoren für Fahrerassistenzsysteme - Von Komfort zu Sicherheitssystemen*. 15. Aachener Kolloquium Fahrzeug- und Motorentechnik. 2006. – URL http://www.aachen-colloquium.com/pdf/Vortr_Nachger/2006/Seuss.pdf. – Zugriffsdatum: 05.08.2012
- [42] SPETH, J.: *Videobasierte modellgestützte Objekterkennung für Fahrerassistenzsysteme*. Cu-villier Verlag Göttingen, 2010. – ISBN 978-3-86955-317-7
- [43] SPIES, M.: *Automobile Lidar Sensorik: Stand, Trends und zukünftige Herausforderungen*. Advances in Radio Science 4. 2006
- [44] TEXAS INSTRUMENTS®: *Open Multimedia Application Platform 4430*. Website. 2011. – URL <http://focus.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?contentId=53243&navigationId=12843&templateId=6123>. – Zugriffsdatum: 21.08.2012
- [45] THE ECLIPSE FOUNDATION: *Eclipse Integrated Development Environment*. Website. 2012. – URL <http://eclipse.org/>. – Zugriffsdatum: 27.12.2012
- [46] UBUNTU FOUNDATION: *Ubuntu*. Website. 2011. – URL <http://www.ubuntu.com>. – Zugriffsdatum: 14.11.2012

- [47] VERLEYE, Bart ; HENRI, Pierre ; WUYTS, Roel: Implementation of a 2D Electrostatic Particle in Cell algorithm in Unified Parallel C with dynamic load-balancing. In: *Parallel Computing* (2011)
- [48] WEHR, A. ; LOHR, U.: *Airborne laser scanning - an introduction and overview*. ISPRS Journal of Photogrammetry and Remote Sensing 54. 1999
- [49] WILKEN, Heiko: *Laserscanner-basierte Objekterkennung für ein Fahrspurführungssystem auf einer Multiprozessor FPGA-Plattform*, Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, 2012
- [50] WINNER, H.: *Mit aktiven Sensoren das Kfz-Umfeld erfassen: Funktion und Leistungsfähigkeit von Radar und Co*. Information Technology. 2007
- [51] WINNER, H. ; HAKULI, S. ; WOLF, G.: *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Vieweg + Teubner, 2009. – ISBN 978-3-83-480287-3
- [52] WUYTS, Roel ; MEERBERGEN, Karl ; COSTANZA, Pascal: Reactive Rebalancing for Scientific Simulations running on ExaScale High Performance Computers. In: *Parallel Computing* (2011)
- [53] XILINX® INC.: *Spartan-3A DSP FPGA Family Data Sheet*. Website. 2010. – URL http://www.xilinx.com/support/documentation/data_sheets/ds610.pdf. – Zugriffsdatum: 23.08.2012
- [54] XILINX® INC.: *System Generator*. Website. 2012. – URL <http://www.xilinx.com/tools/sysgen.htm>. – Zugriffsdatum: 22.08.2012
- [55] XILINX®, INC.: *Zynq-7000 All Programmable SoC*. Website. 2012. – URL <http://www.xilinx.com/products/silicon-devices/epp/zynq-7000>. – Zugriffsdatum: 20.08.2012

Anhang A

Position der Widerstände für die SYSBOOT[5:0] Pins (rot) auf dem Pandaboard, deren Belegung die Bootloader-Reihenfolge steuert:



Anhang B

Android-Manifest *manifest.xml* zum HPEC Eclipse-Projekt:

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="haw.hpec"
4     android:versionCode="1"
5     android:versionName="1.0" >
6
7     <uses-sdk android:minSdkVersion="15" />
8     <uses-permission android:name="android.permission.INTERNET" />
9     <uses-permission android:name="android.permission.WAKE_LOCK" />
10
11     <application android:debuggable="true" android:hardwareAccelerated="true">
12         android:icon="@drawable/ic_launcher"
13         android:label="@string/app_name" >
14         <activity
15             android:label="@string/app_name"
16             android:configChanges="orientation|screenSize|uiMode|keyboardHidden"
17             android:name=".HPECActivity" >
18             <intent-filter >
19                 <action android:name="android.intent.action.MAIN" />
20
21                 <category android:name="android.intent.category.LAUNCHER" />
22             </intent-filter>
23         </activity>
24     </application>
25
26 </manifest>
```

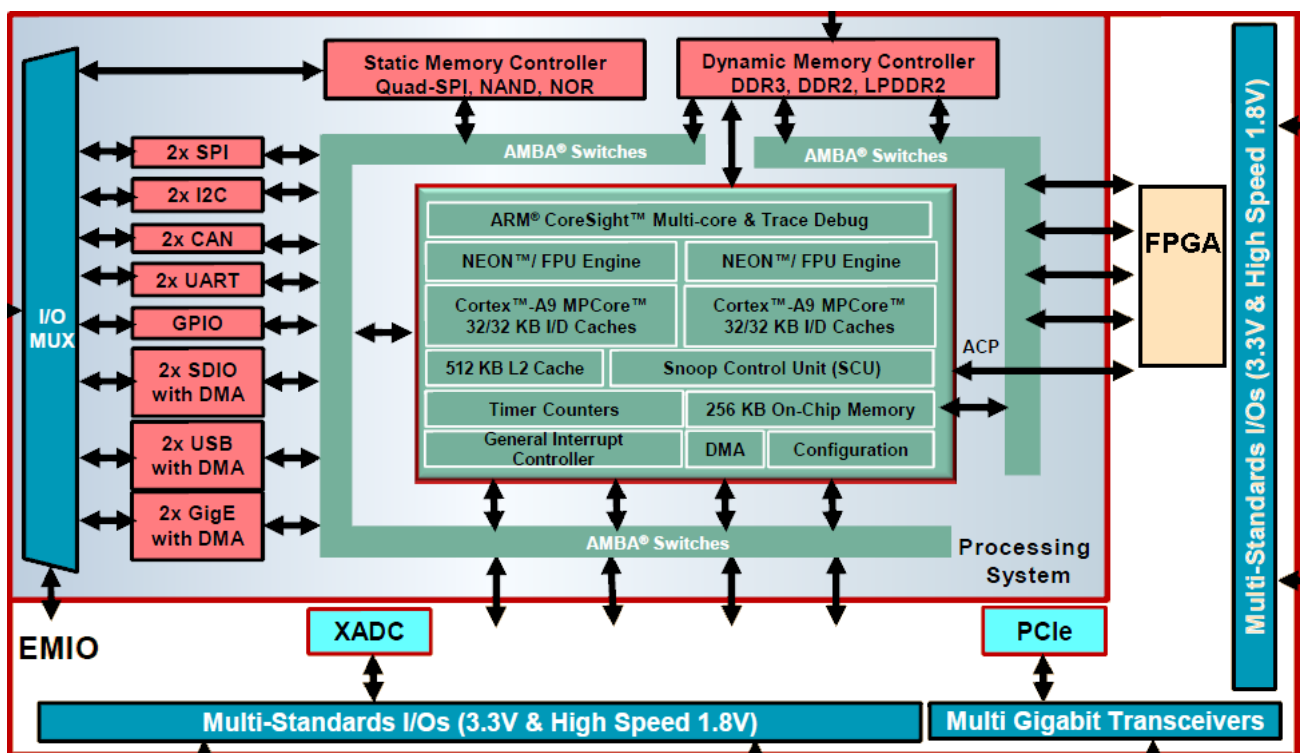
Anhang C

Android GUI-Layout *main.xml* zum HPEC Eclipse-Projekt:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
3   android:layout_width="fill_parent"
4   android:layout_height="fill_parent" >
5
6     <SurfaceView
7       android:id="@+id/surfaceView1"
8       android:layout_width="1001dp"
9       android:layout_height="616dp"
10      android:layout_x="0dp"
11      android:layout_y="0dp" />
12
13 </AbsoluteLayout>
```

Anhang D

Architektur der FPGA-basierten *Zynq-7000 Extensible Processing Platform* MPSoC-Plattform von *Xilinx*, die über den identischem Cortex-A9 Zweikern-Prozessor verfügt [55].



Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 14. Februar 2013

Andre Jestel
