Fehlertolerante Radarmessungen durch Austausch von Messwerten für hochautomatisiertes Fahren

Markus Ulbricht, Rizwan Tariq Syed, Herman Jalli Ng, Mario Schölzel*, Milos Krstic*

IHP
Im Technologiepark 25
15236 Frankfurt (Oder)

*Universität Potsdam
Am neuen Palais 10
14469 Potsdam

E-Mail:[ulbricht | syed | ng | schoelzel | krstic]@ihp-microelectronics.com

ABSTRACT

Im Bereich des autonomen Fahrens müssen Sensoren und die dazugehörige Datenverarbeitung höchste Ansprüche an Zuverlässigkeit und Fehlertoleranz erfüllen. Gleichzeitig sind jedoch auch möglichst geringe Leistungsaufnahme und minimale Herstellungskosten Ziele des Designs. In diesem Beitrag stellen wir einen Ansatz vor, der diese Ziele, basierend auf einer von uns entwickelten Sensorplattform, sowie verschiedenen Testarten und dem Austausch von Messwerten, erfüllen kann.

SCHLÜSSELWÖRTER

Autonomes Fahren, FMCW, BPSK, Redundanz, Akzeptanztest, m-aus-n System, Zuverlässigkeit, Fehlertoleranz

EINFÜHRUNG

Die steigende Anzahl von Assistenzsystemen und die ambitionierten Pläne für mehr Autonomie stellen hohe Anforderungen an die Fehlertoleranz von digitalen Komponenten im Bereich des hochautomatisierten Fahrens. Um diese Anforderungen zu erfüllen, werden häufig sogenannte *dual-*, *triple-* oder *double-dual-modular redundancy* Systeme eingesetzt, bei denen einzelne Systemkomponenten einfach, zweifach oder sogar gruppenweise kopiert und deren Ausgänge verglichen werden. Damit ist man in der Lage, auftretende Fehler erkennen und teilweise korrigieren zu können ([1], [2]). Dieser Ansatz lässt sich zwar vergleichsweise einfach implementieren und liefert gute Ergebnisse, bedeutet aber hohe zusätzliche Kosten im Stromverbrauch und in der Hardware. Weitergehende Forschung ist notwendig, um ähnlich effektive Möglichkeiten mit besserem Kosten-Nutzen-Verhältnis zu liefern.

Als Teil des EMPHASE Projekts ([3]) entwickeln wir die sogenannte *smart reconfigurable sensor* (SRS) Plattform, die Abstandsmessungen und Kommunikation über das gleiche Radar-Frontend durchführen kann. Wir wollen diese Eigenschaften nutzen, um Messdaten zwischen den verschiedenen Verkehrsteilnehmern auszutauschen und darüber Fehler erkennen und ggf. korrigieren zu können. Ziel ist es, somit auf die Erzeugung von Referenzwerten durch redundante Komponenten verzichten zu können und dadruch Strom und bestenfalls sogar Hardware einzusparen, ohne dabei Einbußen in der Fehlererkennungs- und -korrekturrate zu verursachen.

Um dies näher zu ergründen, wird im nächsten Abschnitt der Aufbau und die Funktionsweise der SRS Plattform eingehender betrachtet. Daran schließt sich die Erläuterung unseres Vorhabens, wie Fehler mit Hilfe des Austauschs von Messwerten erkannt und korrigiert werden sollen. Dazu zuerst eine kurze Einführung über die Abstandsmessung per Radar und welche Werte gemessen werden. Anschließend legen wir die Klassifizierungsmöglichkeiten der Messwerte als Referenzwerte zur Fehlererkennung dar und betrachten Sonderfälle, die beim Messen auftreten können. Zum Ende des Abschnitts folgt eine Beschreibung der Implementierungen die zum Test des vorgeschlagenen Ansatzes und seiner Bewertung vorgesehen sind. Im nächsten Abschnitt erfolgt die Auflistung bisheriger Ergebnisse und anschließend Zusammenfassung und Ausblick.

DIE SRS PLATFROM

Der schematische Aufbau der SRS Plattform ist in Abbildung 1 gegeben. Sie besteht im Wesentlichen aus drei Domänen (von rechts nach links): der Sensor Domäne, der *analog digital converter* (ADC) Domäne und der *digital signal processor* (DSP) Domäne. Erstere umfasst dabei einen 79GHz Radar Sensor, der, je nach Ansteuerung, entweder als *frequency modulated continuous wave* (FMCW) Radar zur Entfernungsbestimmung genutzt wird oder per *binary phase shift keying* (BPSK) Daten zur Kommunikation auf das Radarsignal moduliert ([4],[5]). In der ADC Domäne werden die komplexen analogen Werte vom Sensor in digitale Werte für die weitere Datenverarbeitung in der DSP Domäne umgewandelt. Die DSP Domäne besteht im Wesentlichen aus einer konfigurierbaren Anzahl von identischen Tensilica Xtensa DSP Prozessoren (drei in Abbildung 1), die zu einem zur Selbstreparatur fähigen *m-aus-n System* verbunden sind. Im vorliegenden Fall müssen sie drei Tasks ausführen: die Bestimmung der Abstandsmenge (Task **A** - mehr dazu im nächsten Abschnitt), die notwendigen Schritte zur Übertragung der Daten mit Hilfe des Radar

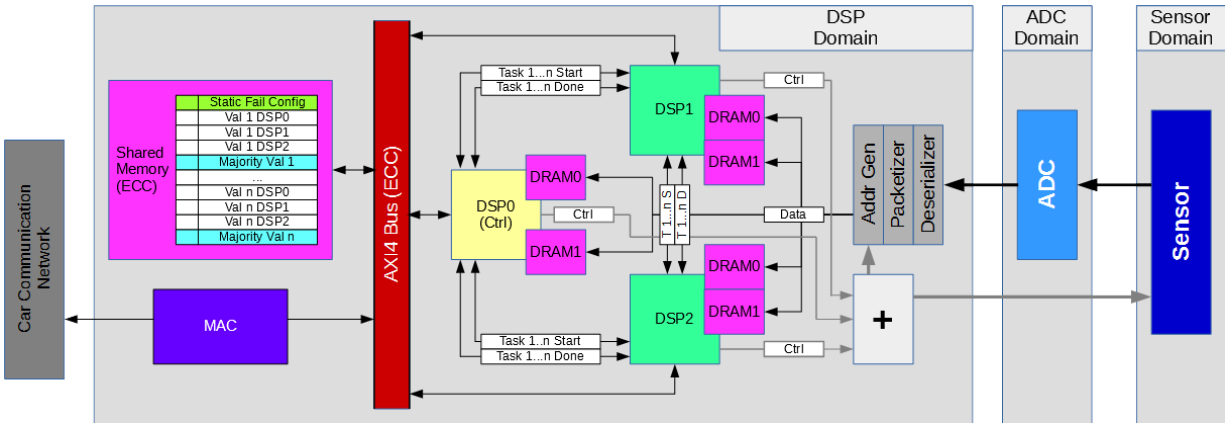

Abbildung 1 Schematischer Aufbau der SRS Komponente

Signals, wie *forward error correction* (FEC), Filtern, Timing-Recovery etc. (Task **D**) und die Kontrolle dieser Abläufe (Task **K**). Task **K** hat weiterhin die Möglichkeit, das System zu rekonfigurieren, Messergebnisse zu vergleichen und einen Mehrheitsentscheid durchzuführen. Dies versetzt ihn in die Lage, als Voter für auf mehreren DSP parallel ausgeführte Task **A** zu fungieren und damit Fehler erkennen und ggf. korrigieren zu können.
Zur Entwicklung eines Prototyps wird die DSP Domäne im Rahmen des Projekts zuerst auf einem FPGA implementiert. Dies birgt einerseits natürlich den Nachteil, dass die Taktfrequenzen mit voraussichtlich deutlich unter 200MHz wesentlich geringer ausfallen werden als bei einem echten ASIC, was deutliche Auswirkungen auf das zeitliche Verhalten des Systems haben wird. Andererseits erlaubt es aber große Freiheiten bei der Gestaltung des *m-aus-n Systems*, was die Anzahl der DSP und ihre Auslastung betrifft. Somit sind wir in der Lage, das System in Bezug auf Timing und Leistungsaufnahme direkt zu vergleichen, wenn die Redundanz aktiv rechnet, passiv wartet oder komplett von der Stromversorgung getrennt ist.

FEHLERERKENNUNG UND –KORREKTUR DURCH AUSTAUSCH VON MESSWERTEN

Wie bereits angedeutet, ist es bei der Fehlertoleranz im Automobilbereich üblich, eine Komponente zu vervielfachen und die Ausgangswerte zu vergleichen, um aufgetretene transiente Fehler sofort erkennen und, bei Mehrheitsentscheiden, korrigieren zu können. Dies ist auch bei der SRS Komponente durch die parallele Ausführung des Task **A** auf mehreren DSP möglich. Ziel ist es jedoch, den Aufwand an benötigter Energie und Hardware bei gleichem Maß an Fehlertoleranz zu verringern. Aus diesem Grund schlagen wir vor, Referenzwerte über das vorhandene Radar-Frontend zwischen den Fahrzeugen auszutauschen, statt sie über zusätzliche Hardware selbst zu generieren. Die folgenden Abschnitte betrachten nun den Messvorgang selbst, sowie die Fähigkeiten zur Fehlererkennung und möglichen Einsparungen an Energie und Hardware dieses Ansatzes näher.

Abstandsmessungen per Radar

Die Abstandsmessungen werden mit Hilfe eines FMCW Radars durchgeführt. Für den Messvorgang modulieren wir eine in 4 µs um 4 GHz ansteigende Frequenzrampe auf das 78 GHz Basissignal. Daraufhin kann mit Hilfe der Frequenzdifferenz zwischen dem originalen und dem vom entfernten Objekt reflektierten Signal und anschließender Fourier Transformation die Entfernung zum Objekt bestimmt werden. Das Ergebnis einer solchen Messung ist exemplarisch in Abbildung 3 dargestellt. Erkennbar sind drei Maxima, die für jeweils ein reflektierendes Objekt in 10, 20 und 25 Metern Entfernung stehen. Die daraus resultierende Menge an Abstandswerten

$$A = \{ 10, 20, 25 \}$$

und ihre Mächtigkeit

$$|A| = 3$$

sind jedoch die einzigen Resultate, die der Messvorgang liefert. Informationen über die Position der Objekte oder der jeweiligen Abstände zwischen ihnen erhalten wir nicht.
Die geringe Komplexität der Messergebnisse erleichtert jedoch die Überprüfung mit geteilten Werten anderer Verkehrsteilnehmer, da nicht erst aufwändig Vektoren transformiert oder Koordinaten ermittelt werden müssen. Die Überprüfung kann, wie auch der Vergleich mit redundanten Werten, direkt auf Ebene der DSP erfolgen und sollte deshalb zügig vonstatten gehen und andere Systemkomponenten nicht belasten.

Akzeptanztests zur Fehlererkennung

Geteilte Messwerte werden im Gegensatz zu redundant erzeugten Werten kleinere Abweichungen zu den selbst gemessenen Abständen besitzen. Deshalb kann man sie nicht direkt gleichsetzen, sondern muss sie mit Hilfe eines Akzeptanztests auf Richtigkeit untersuchen. Dieser kann jedoch eine redundante Komponente ersetzen, falls er die empfangenen Messwerte mit gleicher oder höherer Wahrscheinlichkeit korrekt als richtig oder falsch klassifiziert, wie dies bei einem Vergleich zu redundant erzeugten Werten geschehen würde ([6]). Dies lässt sich anhand des Beispiels in Abbildung 2 erläutern.
Angenommen, die Fahrzeuge F1 und F2 messen während ihrer Task **A** die jeweiligen Abstandsmengen

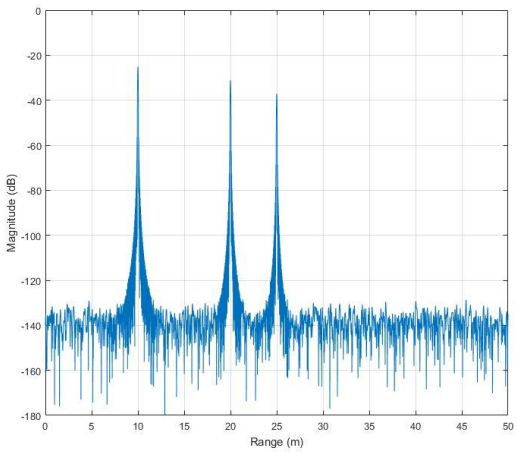

Abbildung 3 Simuliertes Messergebnis des Radars

A1 = { a1, a2, a4 } und
A2 = { a1, a3, a5 }.

Anschließend broadcastet F2 diese Menge *A2* über sein Radar und F1 vergleicht ihn in seinem Task **K** mit *A1*. Folgende Klassifizierungen lassen sich unterscheiden:

- **True positive:** Bei jedem Vergleich gibt es exakt ein Element beider Mengen, das übereinstimmt. Dies ist der gegenseitig voneinander gemessene Abstand *a1*.
- **True negative:** Der Vergleich aller anderen Elemente *a2...a5* ergibt keine Übereinstimmung, da es sich um den Abstand zu anderen Objekten handelt (F3 und F4).

Sind beide Bedingungen erfüllt, war die Messung mit einer gewissen Wahrscheinlichkeit richtig. Diese Wahrscheinlichkeit steigt mit jedem weiteren Vektor, der Empfangen und richtig klassifiziert wird (beispielsweise die Messwerte von F3 und F4).
Jedoch können auch andere Fälle eintreten:

- **False positive:** In Einzelfällen kann es dazu kommen, dass mehr als ein Punkt übereinstimmt: Zum Einen, falls F3 und F4 tatsächlich denselben Abstand zu F1 und F2 haben. Zum Anderen, durch fehlerhafte Berechnung und somit fälschlich übereinstimmende Werte.
- **False negative:** Weiterhin ist es auch möglich, dass selbst *a1* nicht übereinstimmt. Beispielsweise durch kurzzeitige Blockade der Messung eines Fahrzeuges (und damit die einseitige Messung von *a1*) oder fehlerhafte Berechnung.

Durch entsprechend hohe Messgenauigkeit und eine Beobachtung der Werte über einen längeren Zeitraum sollten sich jedoch zufällig falsche Ergebnisse von durch falsche Berechnungen verursachte Fehleinschätzungen unterscheiden lassen. Das heißt, dass als **false positive** und **false negative** klassifizierte Messungen mit hoher Wahrscheinlichkeit auf einen Fehler in der Berechnung hinweisen, somit die oben genannte Bedingung aus [6] erfüllen und deswegen zur Fehlererkennung und –korrektur in der Hardware genutzt werden können.
Oft wird es außerdem der Fall sein, dass nicht alle Punkte klassifizierbar sind. So zum Beispiel, wenn F3 und F4

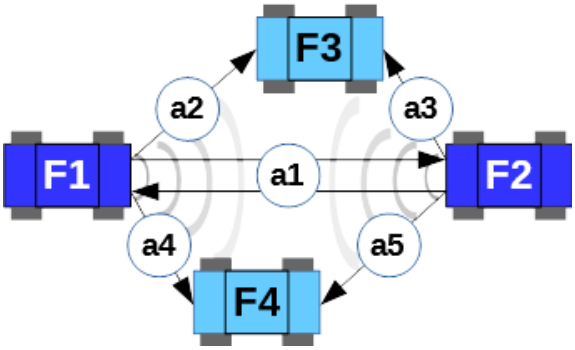

Abbildung 2 Angenommene Verkehrssituation

keinen oder einen Sensor ohne Kommunikationsmöglichkeit besitzen. Ob nun die Verifizierung von nur einem von insgesamt vier Messpunkten ausreicht, um die gesamte Messung als richtig oder falsch zu klassifizieren, ist zu diesem Zeitpunkt noch fraglich. Jedoch sollte ein Vektor mit ausschließlich **true positives** und **true negatives** bei entsprechend hoher Genauigkeit ein sehr deutliches Indiz für korrektes Messen sein.
Zwei weitere Sonderfälle sind das komplette Fehlen von Referenzwerten (wenn keine anderen Verkehrsteilnehmer in der Nähe sind) und der Empfang (absichtlich) falscher Werte. Beides klärt sich schnell, wenn weitere Verkehrsteilnehmer mit korrekten Ergebnissen in die Nähe kommen. Ist dies nicht der Fall, wird der Einsatz von redundanter Hardware zur Erzeugung eigener Referenzwerte aus Sicherheitsgründen unumgänglich sein. Sie könnte beispielsweise im Normalfall von der Stromversorgung getrennt und nur im Bedarfsfall aktiviert werden. Wir wollen die beiden Fälle in den weiteren Betrachtungen jedoch erst einmal zurück stellen um das Potential des vorgeschlagenen Ansatzes besser einschätzen zu können.

Implementierungen

Um die Möglichkeiten zur Fehlererkennung und die Einsparungen an Strom und eventuell Hardware ermitteln zu können, sind eine *fail safe* und eine *fail operational* Implementierung der SRS Komponente vorgesehen. *Fail safe* bedeutet dabei, dass zusätzlich zum originalen Wert immer zumindest ein Referenzwert vorliegt, mit dem Unterschiede in der Berechnung der Werte festgestellt werden können. Dies ermöglicht das Auslösen von speziellen Maßnahmen im Fehlerfall, die das System in einen sicheren Zustand (safe state) versetzen ([7]). Die *fail safe* Implementierung wird dabei aus drei DSP cores bestehen, die in drei unterschiedlichen Modi betrieben werden (siehe Abbildung 4 oben). Im ersten Modus (oben links) führen zwei DSP den Task **A** parallel aus und der dritte Task **K**. Damit erhalten wir ein einfaches DMR System, mit dessen Hilfe der Basiswert zu Fehlererkennung und Stromverbrauch ermittelt wird. Dass Task **K** dabei selbst fehlerfrei abläuft werden wir mit verschiedenen Methoden der Software Fehlertoleranz sicherstellen ([8]). Dies soll in diesem Beitrag jedoch nicht betrachtet werden.
Im zweiten Modus (Abbildung 4 oben mitte) führt ein Core den Task **A** zum Messen, einer den Task **K** als Kontrollinstanz und der dritte den Task **D** aus, mit dessen

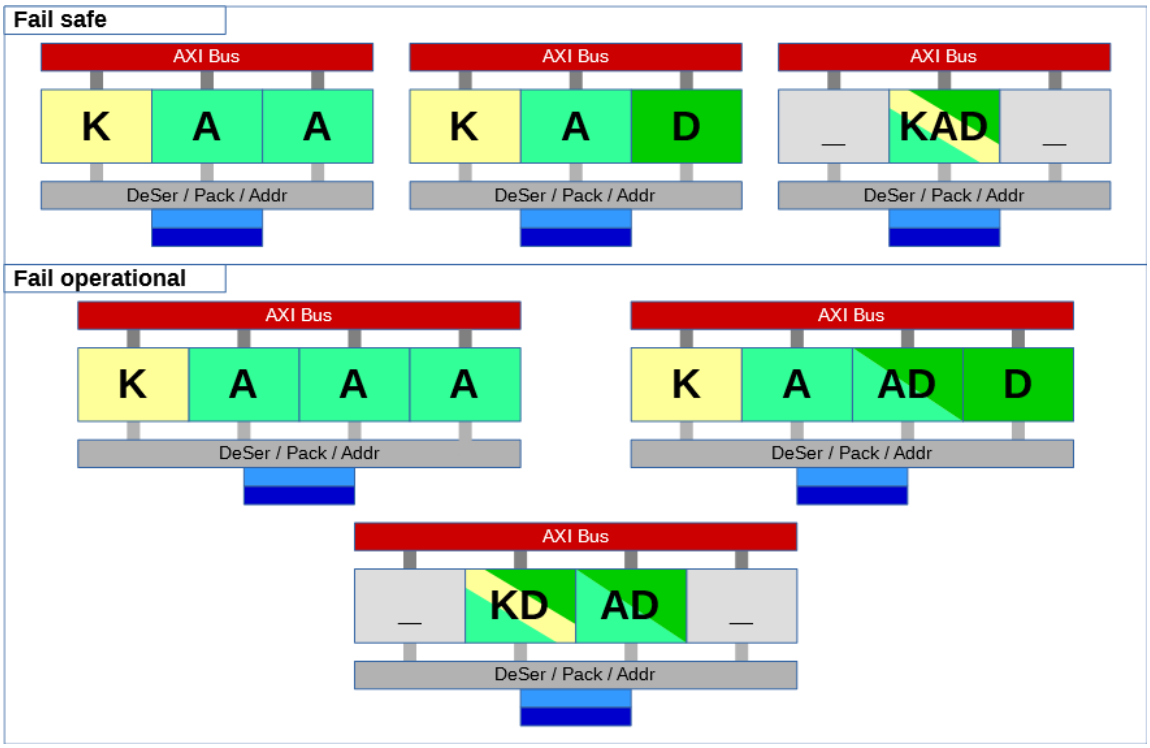

Abbildung 4 Fail Safe und Fail Operational Implementierungen

Hilfe er Referenzwerte empfängt. Hier wird sich zeigen, ob durch Task **D** neue Fehler aktiviert werden, wie gut sie mit dem vorgeschlagenen Ansatz erkennbar sind und welche Unterschiede es im Stromverbrauch zwischen den Tasks gibt. Der dritte Modus (oben rechts) wird zur Untersuchung der maximalen Einsparungen eingesetzt, indem sämtliche noch vorhandene (zeitliche) Redundanz genutzt und die Tasks auf so wenig DSP wie möglich ausgeführt werden. Die restlichen, freien DSP werden dann entweder deaktiviert und als kalte Redundanz genutzt oder, falls sie nicht mehr benötigt werden, aus dem System entfernt.

Fail operational bedeutet, dass das System selbst im Fehlerfall funktionsfähig bleiben muss, da es keinen „safe state“ gibt ([7]). Um dies zu erreichen, werden zusätzlich zum Original zwei weitere Kontrollwerte benötigt, mit deren Hilfe über Mehrheitsentscheid das wahrscheinlich richtige Ergebnis gefunden werden kann. Die *fail operational* Implementierung (Abbildung 4 unten links) wird im Basisfall drei Mal Task **A** parallel ausführen und besitzt damit die notwendigen drei Werte. Beim zweiten Modus tauschen wir ein bis zwei Task **A** durch Task **D** aus und versuchen Referenzwerte von mehreren Verkehrsteilnehmern in den Akzeptanztest einfließen zu lassen, um einen möglichen Einfluss auf dessen Robustheit zu überprüfen. Modus 3 (unten mitte) hilft wieder, durch Konzentration der Aktivität auf so wenige DSP wie möglich, den minimalen Aufwand an Strom und Hardware zu bestimmen.

VORLÄUFIGE ERGEBNISSE

Die Implementierung der Tasks befindet sich noch in einem zu frühen Stadium, um präzise Aussagen über das Timing machen zu können. Geplant ist, dass Task **A** und **D** innerhalb von 10 ms nacheinander abgearbeitet werden können; Task **K** läuft dazu parallel. Die Implementierung von Task **A** ist soweit abgeschlossen und besitzt durch die Hardwarebeschleuniger auf den G3 und G6 DSP eine sehr kurze Ausführungszeit. Die Funktionalität von Task **D** ist auch komplett, jedoch in Bezug auf das Timing noch nicht optimiert. Task **K** wurde noch nicht implementiert. Die genauen Laufzeiten auf zwei verschiedenen Konfigurationen von G3 DSP – Typ mit single precision floating point unit (SPFPU) und max mit double precision floating point unit (DPFPU) - bei einer simulierten Taktrate von 150 MHz sind in Tabelle 1 aufgelistet:

Tabelle 1 Laufzeiten von Task A und D

DSP core	Task A	Task D
G3 Typ (SPFPU)	0,55ms	68,7ms
G3 Max (DPFPU)	0,51ms	25,3ms

Die Bestimmung der zusätzlichen Kosten für die Hardware lässt etwas Interpretationsspielraum. Die Hardwarebeschleuniger für Task **A** sind bereits vorhanden und können für Task **D** weitestgehend mit genutzt werden. Zusätzliche Blöcke zur Beschleunigung von Task **D** würden zwar die Hardware insgesamt vergrößern, können aber schwerlich als extra Aufwand für den vorgeschlagenen Ansatz betrachtet werden, da die Kommunikationsfähigkeit des Task **D** eine Grundfunktion des Systems darstellt. Die Einsparungen werden sich jedoch eindeutig zeigen lassen, falls der vorgeschlagene Ansatz so gut funktioniert, dass weniger redundante Hardware ins System eingebaut werden muss.

In Bezug auf die Einsparungen im Strom können nur erste, vorsichtige Schätzungen gemacht werden. Die Energie des Systems E_{SRS} berechnet sich aus der aktiven

Energie der ausgeführten Tasks (E_K, E_A, E_D) innerhalb des 10ms dauernden Messzyklus und der Ruheenergie der DSP E_R, die während dieser Zeit allein dadurch entsteht, dass die DSP nicht von der Stromversorgung entkoppelt sind. Das heißt für die *fail safe* Implementierung im ersten Modus mit einem ausgeführten Task **K**, zwei ausgeführten Task **A** und drei aktiven DSP:

$$E_{SRS\text{-}FS1} = E_K + 2*E_A + 3*E_R$$

Im dritten Modus mit nur einem aktiven DSP, der alle drei Tasks ausführt, würde sich die Energie wie folgt berechnen:

$$E_{SRS\text{-}FS3} = E_K + E_A + E_D + E_R$$

und ist somit nur kleiner, wenn

$$E_{SRS\text{-}FS3} < E_{SRS\text{-}FS1}$$
$$E_K + E_A + E_D + E_R < E_K + 2*E_A + 3*E_R$$
$$E_D < E_A + 2*E_R$$

Dies gilt jedoch nur unter der Annahme, dass E_K beim Vergleich redundanter Werte und beim Akzeptanztest annähernd gleich bleibt. Für die *fail operational* Implementierung ist die Berechnung unter den gleichen Annahmen analog. Im ersten Modus erhalten wir:

$$E_{SRS\text{-}FO1} = E_K + 3*E_A + 4*E_R$$

Für den dritten Modus mit zwei aktiven DSP, einem Task **K**, einem Task **A** und zwei Task **D**:

$$E_{SRS\text{-}FS3} = E_K + E_A + 2*E_D + 2*E_R$$

Woraus folgt:

$$E_{SRS\text{-}FO3} < E_{SRS\text{-}FO1}$$
$$E_K + E_A + 2*E_D + 2*E_R < E_K + 3*E_A + 4*E_R$$
$$E_D < E_A + E_R$$

Was sogar noch ein wenig ungünstiger ist, als im *fail safe* Fall.

ZUSAMMENFASSUNG UND DISKUSSION

Der vorgeschlagene Ansatz, durch den Austausch von Messwerten Kosten bei Strom und Hardware einzusparen, ohne dabei Einbußen in der Fehlererkennungs- und -korrekturrate in Kauf zu nehmen, erscheint bei ersten Betrachtungen durchaus sinnvoll. Jedoch ist an einigen Stellen noch viel Arbeit nötig, um dies eindeutig zu klären. Dazu gehört als erstes die Fertigstellung der Tasks, um Machbarkeit, Stromverbrauch und Timing zu überprüfen. Weiterhin müssen die bisher nur am Rande betrachteten Sonderfälle wie (absichtlich) falsche Werte und fehlende Verkehrsteilnehmer eingehender untersucht werden. Als drittes werden echte Sensormessungen im Feld benötigt, um beispielsweise Einflüsse wie den Signal-Rausch-Abstand oder nicht-punktförmige reflektierende Objekte mit in die Betrachtungen und speziell den Akzeptanztest einfließen zu lassen.

Trotz dieser Lücken halten wir den Ansatz für wert, weiter verfolgt zu werden. Wir konnten bei unseren Literaturrecherchen keine ähnlichen Ansätze finden, bei denen Referenzwerte auf so einer niedrigen Systemebene ausgetauscht werden können wie hier. Und selbst wenn damit keine Einsparungen in Bezug auf Energie oder Hardware möglich sind, so stellt es doch ein zusätzliches Mittel dar, um die Robustheit des Systems deutlich zu erhöhen. Dies liegt im Wesentlichen an drei Punkten:

- Wir erhalten externe Referenzwerte, bei deren Erzeugung mit hoher Wahrscheinlichkeit nicht die gleichen Fehler passiert sein können, wie im eigenen Fahrzeug.
- Dadurch, dass sich die Software von Task **A** und **D** unterscheidet, aber gleiche Ergebnisse entstehen, erhalten wir eine Art N-Version-Programming, was die Maskierung von Fehlern deutlich erschwert.
- Dieser Unterschied in der Software wird höchstwahrscheinlich auch unterschiedliche Komponenten der Hardware ansprechen und dadurch eventuelle Fehler in diesen Teilen aktivieren, die sonst unentdeckt blieben. Damit erhöht sich die Beobachtbarkeit des Systems, wodurch Fehler schneller gefunden und entsprechende Gegenmaßnahmen, wie etwa Selbstreparatur, eingeleitet werden können.

DANKSAGUNG

Das diesem Bericht zugrunde liegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 16EMO0177K gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

LITERATUR

[1] S. Inc., “Dual-core lockstep processors,” 03 2017. [Online]. Available: http://articles.sae.org/15319/

[2] L. D. Daniel Wanner, Annika Stensson Trigell and J. Jerrelind, “Survey on fault-tolerant vehicle design,” International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium (EVS26), 05 2012.

[3] IHP, “Emphase project page,” 01 2017. [Online]. Available: http://www.emphase-projekt.de/

[4] Maciej Kucharski, Dietmar Kissinger and Herman Jalli Ng, “A Monostatic E-Band Radar Transceiver With a Tunable TX-to-RX Leakage Canceler for Automotive Applications” in Proc. 2018 IEEE MTT-S International Microwave Symposium (IMS), 2018

[5] Herman Jalli Ng, Maciej Kucharski, Wael Ahmad, and Dietmar Kissinger, “Multi-Purpose Fully Differential 61- and 122-GHz Radar Transceivers for Scalable MIMO Sensor Platforms,” IEEE J. Solid-State Circuits, vol. 52, no. 9, pp. 2242–2255, Sept 2017.

[6] B. Parhami, “Design of reliable software via general combination of n-version programming and acceptance testing,” in Software Reliability Engineering, 1996. Proceedings., Seventh International Symposium on, Oct 1996, pp. 104–109.

[7] Christopher Temple and Antonio Vilela, „Fehlertolerante Systeme im Fahrzeug – von fail-safe zu fail-operational". 07 2014. [Online]. Available: http://www.elektroniknet.de/elektronik-automotive/assistenzsysteme/fehlertolerante-systeme-im-fahrzeug-von-fail-safe-zu-fail-operational-110612-Seite-2.html

[8] Israel Koren and C. Mani Krishna, CHAPTER 5 - Software Fault Tolerance, In Fault-Tolerant Systems, Morgan Kaufmann, Burlington, 2007, Pages 147-191, ISBN 9780120885251