Timing-Architektur-Patterns und Anti-Patterns

Wie können Timing-Gaps geschlossen werden? Ein Praxisbericht

Karsten Schmidt, Audi Electronics Venture GmbH
Kai Richter, Symtavision GmbH

Neue Technologien wie Multicore und Ethernet bieten ein Vielfaches der bisher gekannten Rechen- und Kommunikationskapazitäten. Die verfügbaren Ressourcen in den Steuergeräten und im Netzwerk müssen auch auf die Vielzahl der zu integrierenden Teilfunktionen und deren Software aufgeteilt werden. Einen wichtigen Beitrag zur effizienten Umsetzung liefert eine gute Timing-Architektur.

Einleitung

Neue Technologien wie Multicore und Ethernet bieten ein Vielfaches der bisher gekannten Rechen- und Kommunikationskapazitäten. Sie legen damit den Grundstein für die Umsetzung der kundenerfahrbaren Megatrends in nahezu allen Bereichen: Fahrerassistenz, Elektrifizierung, Connectivity, autonomes Fahren. Die neue Herausforderung dabei: Die verfügbaren Ressourcen in den Steuergeräten und im Netzwerk müssen auch auf die Vielzahl der zu integrierenden Teilfunktionen und deren Software aufgeteilt werden. Einen wichtigen Beitrag zur effizienten Umsetzung liefert eine gute Timing-Architektur. Jedoch ist das Finden einer guten Timing-Architektur herausfordernd und häufig mangelt es den Beteiligten an Erfahrung im Umgang mit Echtzeitanforderungen, da diese Aufgabe noch nicht in der Breite etabliert ist.

Im vorliegenden Betrag zeigen wir „Best Practices“ für die Lösung wiederkehrender Entwurfsprobleme im Bereich Timing und erfolgreiche und erprobte Muster für das Finden guter Timing-Architekturen, also „Timing-Architektur-Pattern“. Zudem grenzen wir diese explizit von „Nicht Pattern“ bzw. „worst practices“ oder „not-so-good practices“ ab, die heute in der Praxis (leider) beobachtet werden können. Damit wollen wir Anregungen geben, wie und mit welchen konkreten Schritten das Thema Timing künftig besser gehandhabt werden könnte.

Situation der Automobilindustrie

Für die Realisierung von Fahrzeugfunktionalitäten spielt Software eine zunehmend wichtigere Rolle. Während sie in frühen Systemen lediglich in ausgewählten Komponenten zu finden war, ist sie mittlerweile für fast jede Fahrzeugfunktionalität relevant, wodurch neben der Hardware eine eigene Struktur der Software entsteht. Die Softwareelemente des Fahrzeugs werden innerhalb von Steuergeräten (ECUs) und anderen eingebetteten Komponenten ausgeführt. Traditionell ist neben den nichtfunktionalen Anforderungen dabei der sparsame Umgang mit knappen Betriebsmitteln (Speicher, CPU-Zyklen, Kommunikationsslots) ein wesentlicher Aspekt. Jeder noch so kleine Anstieg im Ressourcenverbrauch wirkt sich als Multiplikator direkt auf die Bauteilkosten aus. Mit den steigenden Anforderungen an Funktionalitäten und neue

Wertschöpfungsketten wächst die Menge von Software mittlerweile so stark, dass Timing-Probleme häufig zu beobachten sind.

Die Entwickler stellen sich den Herausforderungen und haben bereits eine Vielzahl unterschiedlicher Strategien zur Beherrschung von Timing-Problemen entwickelt. Jedoch tauchen durch die Komplexität des gesamten Entwicklungsprozesses immer wieder typische Probleme auf.

Verständnislücke

Bedingt durch die Aufgabenteilung in Systementwicklung, Funktionsentwicklung und der eigentlichen Softwareentwicklung ist häufig ein Kommunikationsproblem zwischen den unterschiedlichen Stakeholdern im Entwicklungsprozess zu beobachten. Jeder beherrscht sein Teilgebiet exzellent, jedoch gehen bei der Kommunikation durchaus relevante Informationen verloren. Erschwert wird dies durch den notwendigen Know-How-Schutz bei der Zusammenarbeit unterschiedlicher Firmen. In [4, 5] ist dieser Fakt ausführlich beschreiben. Verstärkt wird dieser Effekt durch die Integration unterschiedlicher Softwareanteile in ein Steuergerät.

Werkzeuglücke

Sehr eng verwandt mit der Verständnislücke ist die Werkzeuglücke. Jeder Stakeholder benutzt das für seine Zwecke passende Werkzeug, um effizient seine Probleme zu lösen. Der Austausch zwischen den einzelnen Parteien ist häufig durch händische Aktivitäten gekennzeichnet.

Gesamtsystemverständnis

Bedingt durch die Zusammenarbeit über Firmengrenzen hinweg und der Integration von Software unterschiedlichster Parteien ist es für den eigentlichen Integrator schwer, bei Problemen mit einem Gesamtsystemverständnis an die Fehlersuche zu gehen.

Pattern

Grundlegende und wiederkehrende Erfahrungen aus unterschiedlichsten Projekten manifestieren sich in Pattern, die entsprechend angewendet, großen Nutzen spenden. Trotzdem setzt die Anwendung von Patterns bzw. der Vermeidung von Anti-Pattern eine gewisse Erfahrung voraus, da nicht jedes Pattern zwingend immer erfolgreich anwendbar ist.

Vermessung des Systems

Ein Pattern, welches unserer Erfahrung nach immer anwendbar ist, ist die Entwicklung von Messmethoden für Timing-Fragen im zu entwickelnden System, da nur Dinge, die quantitativ vorliegen, auch optimiert werden können [1, 2].

Das grundlegende Vorgehen beim Tracing ist die Aufnahme von Zeitstempeln. In aktuellen Systemen erfolgt das typisch auf zwei Ebenen:

- Schedule: Tasks, Runnables, Start-Stop, Laufzeit, CPU-Last
- Events: Reihenfolgen, Abstände, Versatz, Jitter, Zykluszeit

Dabei geht es grundsätzlich um den ausführbaren Code, der ja auf CPU läuft und somit getraced werden kann. Dabei existieren zwei grundlegende Methoden:

- Code-Instrumentierung
- Hardware-Tracing (Typischerweise durch externe Geräte)

Hierbei ist zu beachten, dass prinzipiell jede Messung das System verfälschen kann. Aus diesem Grund ist es notwendig die Messtechnik möglichst non-intrusive auszulegen. Zukünftig werden deshalb CPU-Hersteller erweiterte on-chip Trace-Möglichkeiten etablieren müssen (z.B. in einer Art Timing-Unit für Programmspeicherzugriffe), um das non-intrusive Tracing aus den Laboren in die Fahrzeuge zu bringen.

Systemmodellierung

Habe ein Modell deines Systems in der richtigen Granularität! Eine wichtige Grundlage dafür ist ein systematisches Erfassen der Anforderungen. Hier empfiehlt sich folgende Vorgehensweise:

- Auf Code-Ebene sollten mindestens Laufzeiten, Zykluszeiten und die CPU-Last erfasst werden.
- Auf Schedule-Ebene müssen die Deadlines erfasst werden, sonst existieren keine Kriterien für Schedule-Verifikation beziehungsweise Optimierung.
- Auf der Ebene der End-to-End Kommunikation sollten mindesten Anforderungen an die Reihenfolge der Runnables erfasst werden.
- Eine praktische Empfehlung besteht in der Annotierung der Timing-relevanten Daten an die erzeugten Artefakte.

Vergleiche System und Messung

Dieses Pattern benutzt und kombiniert die beiden vorhergehenden Patterns. Wichtig ist, dass prüfbare Anforderungen existieren, und Timing-Anforderungen mit größerer Komplexität auf einfache und prüfbare Timing-Anforderungen zurückgeführt werden können.

Nutzung einer Timing-Simulation zur Prognose

Simulationen, insbesondere im Bereich Scheduling, welches die originäre Timing-Ebene darstellt, können zur Vorabschätzung verschiedener Fragestellung verwendet werden. Dazu zählen insbesondere Fragestellungen des Typs „Was wäre wenn …?“, wie z.B.

- Weitere Softwareanteile müssen/sollen integriert werden
- Veränderungen im Mapping (z.B. Verschieben von Softwareteilen in andere Zeitraster, insbesondere zur Optimierung
- Robustheit gegen Laufzeitüberschreitungen einzelner Software-Teile
- Grobabschätzung der benötigten Steigerung der Rechenleistung

Interessant bei diesem Pattern ist, dass bei systematischer Modell-Erfassung dieser Schritt leicht umsetzbar ist.

Problemverständnis

Verstehe das Problem und stelle die richtigen Fragen in der richtigen Granularität. Hierzu sind die involvierten Stakeholder im Projekt durch passende Weiterbildungen zu schulen. Bei den Schulungen ist speziell darauf hinzuweisen, dass es unterschiedliche Sichten auf das Timing gibt (funktionale Sicht, Software, Architektur und Schedule-Sicht) mit teilweise unterschiedlicher Terminologie [3].

Weiterhin ist es zwingend notwendig, Timing-Experten im Unternehmen zu etablieren, die in komplizierten Projekten die Rolle eines „Dolmetschers“ zwischen den real unterschiedlichen Sichten wahrnehmen.

Schulungen

Ein wichtiger Pattern-Baustein stellen entsprechende Schulungen da. Sie vermitteln das notwendige Wissen, die notwendigen Fähigkeiten sowie das Handwerkzeug um die genannten erfolgreich Pattern anzuwenden. Timing-Schulungen sollten folgende Fragen behandeln:

- Was ist die „richtige“ Granularität der Modellierung?
- Wie kann ich mein System vermessen?
- Worauf muss ich als Dolmetscher achten?
- Unterschied zwischen Konzepte versus Mechanismen!
- Timing als querschneidender Aspekt und die Notwendigkeit mit unterschiedlichen Sichten zu arbeiten!

Keep it simple

Das ist ein häufig unterschätztes Pattern. Als Daumenregel gilt: „Alles, was man nicht wirklich schnell erklären kann, ist nicht einfach.“ In unserem Kontext bedeutet „wirklich, dass sichergestellt ist, dass eine Erklärung auch vollständig verstanden wird, also was es ist und was es nicht ist. Den Autoren ist klar, dass die Fahrzeugsysteme komplex sind. Trotzdem muss beachtet werden, dass Komplexität ein Problem insgesamt ist und dass es sich auch lohnt, einfachere Systeme zu bauen.

Wichtig ist auch, dass was möglicherweise für einen Stakeholder eine einfache Lösung darstellt, für andere Stakeholder zu extrem komplizierten Lösungen führt.

Workflow und Tools

Ein sinnvoller Workflow ist unabdingbar. Um die Timing-Eigenschaften zu erfüllen, sollte jeder Workflow mindestens die folgenden Schritte in der passenden Granularität umfassen

- Tracing,
- Soll-Ist-Vergleich
- und der dafür notwendigen Anforderungserhebung mindestens auf Schedule-Ebene.

Die folgende Abbildung zeigt diese Schritte exemplarisch an einem beispielhaften Produkt-Entstehungs-Prozesses unter Beachtung der Timing Eigenschaften.

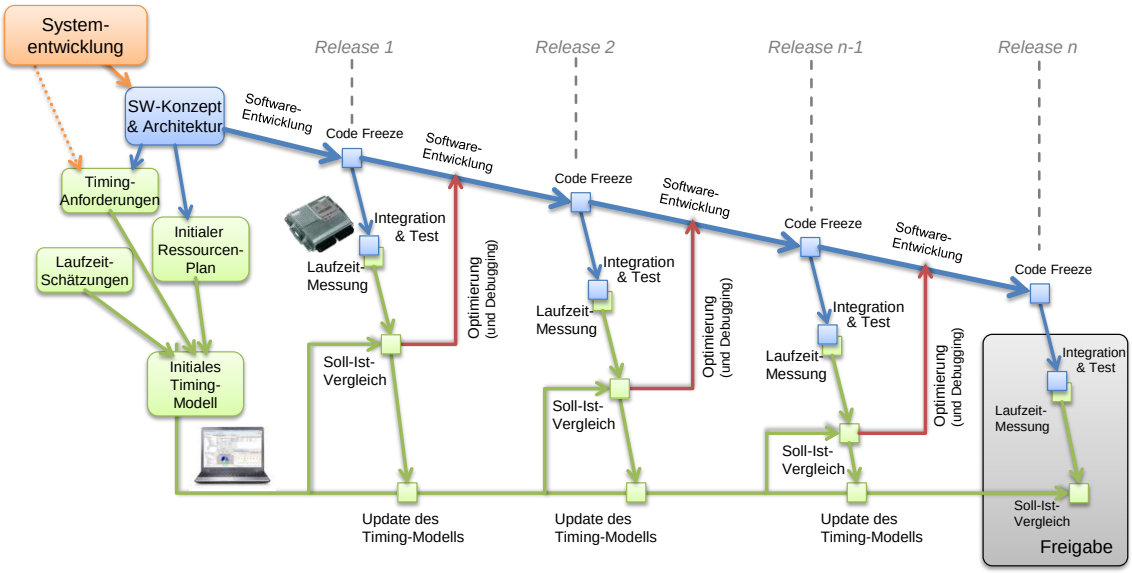

Bild 1 Schematische Darstellung des angereicherten Produkt-Entstehungs-Prozesses

Anti-Pattern:

Auch wenn es zunächst widersinnig erscheint, Anti-Pattern zu dokumentieren, haben sie doch einen großen Nutzen.

Noch mehr Prozess

Prozesse sind nicht grundsätzlich schlecht. Aber Prozesse brauchen Inhalte und unterliegen Randbedingungen, die sich unterscheiden können in:

- Hardware von 8-Bit-uC über embedded-32-bit bis zu Workstations mit Video-Beschleunigern.
- Funktions-Sprachen (DSL)
- Lieferantenketten

Ein Prozess um des Prozesses willen sind akademische Übungen, aber niemals echte Lösungen. Nach Meinung der Autoren müssen die einzelnen existierenden Prozessschritte optimiert werden, durch gezielten Einsatz von neuen und aktuell verfügbaren Technologien. Man erhält dadurch akzeptierte Standardbausteine mit konkretem Nutzen, die je nach Projekt entsprechend kombiniert werden können. Zu strikte Prozessvorgaben schränken ein, wodurch die Akzeptanz sinken wird. Auf der anderen Seite bieten zu generische Prozessbeschreibungen keinen Mehrwert.

Zusätzlich müssen die Prozessbausteine mit entsprechen Schulungsmaßnahmen unterstützt werden, damit die Anwender die Prozessschritte an die konkrete Situation anpassen können.

Monster-Tools

Der wichtigste Aspekt dieses Pattern ist es zu verstehen, dass Tools keine Magie vollbringen können. Wenn das Problem verstanden ist, dann können Tools, die im wesentlichen Maschinen sind, die Dinge automatisieren, sehr gewinnbringend ange-

wendet werden. Weiterhin gilt zu beachten, Tools haben dem Prozess zu folgen, nicht umgedreht, dass ein Tool einen bestimmten Workflow bestimmt.

Zusammenfassung

Die Komplexität im Entwicklungsprozess hochintegrierter Steuergeräte erfordert eine Berücksichtigung von querschneidenden Entwurfsaspekten, wobei wir uns im vorliegenden Beitrag auf Echtzeitfähigkeit und Ressourcenverbrauch fokussiert haben.

Es ist wichtig zu verstehen, dass dies keine leichte Aufgabe, sondern vielmehr ein Paradigmenwechsel ist. Kein einzelnes Werkzeug und kein einzelnes Austauschformat vermögen die Prozesslücke zwischen Systementwicklung, Funktionsentwicklung und Softwareentwicklung zu schließen. Auf der Basis eigener, gelebter Erfahrung haben wir elementare, aufeinander aufbauende Muster vorgestellt:

- Etablierung einer systematischen Ressourcen-Überwachung in der Softwareentwicklung als elementarer Startpunkt.
- Einsatz von modell-basierten Timing-Werkzeugen zur Bedarfs-Prognose und Potenzialanalyse, dadurch Abbau der Lücke „von unten“.
- Experten-Moderation bei der Anforderungsanalyse und
- Schulungen für Mitarbeiter, die Timing und Ressourcen als querschneidenden Aspekt begreifen und das Thema gerade für die unterschiedlichen Sichtweisen aufbereiten, inkl. der Betrachtung von Schnittstellen und Übergängen.

Einfache Standardlösungen existieren zum jetzigen Zeitpunkt nur für wenige Spezialfälle, die von Experten erkannt und genutzt werden können. Diese müssen wir konsequent ausbauen, indem wir die genannten Maßnahmen ergreifen. Systematisch verfolgt, werden dadurch die Prozesslücken sukzessive minimiert.

Literaturverzeichnis

[1] K. Schmidt, J. Harnisch, D. Marx, A. Mayer, „Timing Analysis and Tracing Concepts for ECU Development“, SAE Technical Paper 2014-01-0190, 2014, doi:10.4271/2014-01-0190.

[2] M. Jersak, K. Richter. H. Sarnowski, P. Gliwa, „The Right Timing Analysis Tools Increase Safety and Productivity”, ATZ Elektronik, 01/2009

[3] A. Hamann, D. Ziegenbein, S. Lampke, S. Schliecker, “Resource-Aware Control - Model-Based Co-Engineering of Control Algorithms and Real-Time Systems”, SAE World Congress, April 2015, Detroit, USA

[4] K. Schmidt, D. Marx, K. Richter, K. Reif, A. Schulze, T. Flämig, „On Timing Requirements and a Critical Gap between Function Development and ECU Integration“, SAE World Congress, April 2015, Detroit, USA

[5] K. Reif, K. Schmidt, F. Gesele, S. Reichelt, M. Saeger, N. Seidler, „Networked control systems in motor vehicles“ in ATZelektronik worldwide, 04/2008 Pages 18-23, Springer Fachmedien Wiesbaden GmbH (2008)

Autoren

Dr. Karsten Schmidt studierte Elektrotechnik an der TU Dresden und arbeitet als Entwicklungsingenieur bei der Audi Electronics Venture GmbH. Er ist verantwortlich für neue Architekturkonzepte und den Einfluss der Ethernet-Vernetzung auf zukünftige Steuergerätegenerationen.

Zu seinen weiteren Interessen gehören Entwurfskonzepte für Multicore-Systeme, Zusammenarbeitsmodelle zwischen Software und Funktionsentwicklung sowie Konzepte für sichere (safe und secure) Software.

Kontakt
E-Mail: karsten.schmidt@audi.de
Twitter: @dg1vs

Dr. Kai Richter hat Elektrotechnik an der Technischen Universität Braunschweig studiert (1998) und promoviert (2004). Im Jahr 2005 gründete er die Symtavision GmbH, deren Geschäftsführer und CTO er ist. Kai Richter ist weltweit beratend für Symtavision tätig, kann auf Erfahrungen aus unzähligen Kunden-Projekten zugreifen und zählt zu den Top-Experten für Timing- und Echtzeitfragen.

Zu seinen weiteren Interessen gehören Software- und System-Architekturen, neue Technologien (z.B. Multicore und Ethernet) sowie Zusammenarbeitsmodelle zwischen OEMs und Lieferanten.

Kontakt
Email: richter@symtavision.com