PHASENANALYSE VON FUNKTIONEN IN EINEM HIERARCHISCHEN ASYNCHRONEN MULTI-CORE SYSTEM

T. Hanti; A. Frey; M. Ernst, Technische Hochschule Ingolstadt, Esplanade 10, 85049 Ingolstadt, Germany

Zusammenfassung

Die Anzahl und Komplexität an Echtzeitfunktionen in einem Elektrisch/Elektronik (E/E) System, wie in aktuellen Automobil- und Flugzeugsystemen, steigt beständig. Um die nötige Rechenleistung kostengünstig und auf kleinen Raum bereitzustellen, werden immer mehr Multi-Core Prozessoren eingesetzt. Diese Multi-Core Prozessoren werden aber ähnlich wie Single-Core Prozessoren mit einer statischen Software Konfiguration betrieben. Um einen Multi-Core Prozessor effizienter auszunützen, ist der Schritt hin zu einer rekonfigurierbaren Software Konfiguration ausschlaggebend. In unserer Publikation zeigen wir eine Möglichkeit für ein phasengesteuertes rekonfigurierbares Software System im Bereich des Echtzeit Multi-Core Schedulings auf. Hierbei ist vor allem die Phasenanalyse von Bedeutung, bei der wir erläutern, welche Informationen über eine phasenabhängige Echtzeitfunktion und Systemhardware gesammelt werden müssen, um daraus eine Verteilung der Funktionen für diese Phasen im System korrekt vorzunehmen. Als Ergebnis erwarten wir ein mit mehr Funktionen allokiertes System oder einen leistungsärmeren Multi-Core Prozessor im Vergleich zu einer statische Software Konfiguration.

1. EINLEITUNG

In automobilen Elektrisch/Elektronik Systemen gibt es eine Vielzahl an Fahrerassistenzfunktionen, die den Kraftfahrzeugführer während seiner Fahrt in jeder Phase unterstützen können. Diese Funktionen werden abhängig von der jeweiligen Phase aktiviert oder deaktiviert. So werden einige Funktionen nur während einer Autobahnfahrt, andere wiederum nur bei sehr langsamer Fahrt angeboten. Auch im Bereich der Luftfahrt lässt sich diese Eigenschaft zeigen, hier werden bestimmte Funktionen nur in der jeweiligen Missionsphase benötigt, z.B. bei Start/Landung oder Reiseflug. Aus dieser Eigenschaft folgt eine direkte Abhängigkeit der Worst Case Execution Time (WCET), bzw. Rechenzeit, einer Funktion von der Phase. In den heutigen statisch konfigurierten Multi-Core Software Systemen der Automobil- und Luftfahrtindustrie, werden phasenunabhängige maximale WCETs für diese Funktionen reserviert. Um die Rechenkapazität einer Embedded Control Unit (ECU) im E/E System mehr auszuschöpfen, ist der nächste logische Schritt und unser Ziel, die für Single-Core optimierte statische Software System Konfiguration für das E/E System zu einer rekonfigurierbaren Software System Konfiguration weiterzuentwickeln.

2. REKONFIGURIERBARE SOFTWARE SYSTEME

In einem rekonfigurierbaren Software System werden zur Designzeit, Funktionen auf ihre logische Abhängigkeit hin überprüft und in unabhängige, komplementär rechnende, logische Funktionssets zusammengefasst. Zur Laufzeit werden phasenabhängig, die benötigten logischen Funktionssets dynamisch vom Scheduler geladen/entladen und somit das System rekonfiguriert. Durch diese Eigenschaft ist es möglich temporär nicht von Funktionen ausgenutzte Rechenzeit, phasenabhängig anderweitig zu nutzen, um das Ziel der höheren Effizienz zu erreichen. Ein Scheduler der diese Möglichkeit bietet ist der HAMS Scheduler mit seiner *offline* generierten Knowledgebase [1]. Zur Erstellung der in der Knowledgebase abgelegten logischen Funktionssets werden unterschiedlichste phasenabhängige und phasenunabhängige Parameter benötigt. Diese Parameter werden im Folgenden betrachtet.

3. RELATED WORK

Phasenunabhängige Funktionsparameter für Echtzeitscheduling werden benötigt zur Erzeugung eines statischen durchführbaren Plans (Schedule). Bei diesen Funktionsparametern handelt es sich um folgende [2] [3] [4]:

1) Max. Ausführungszeit: C_i
2) Periode: T_i
3) Deadline: D_i

Eine periodische oder aperiodische Funktion τ_i für ein Echtzeitbetriebssystem wird auf Basis dieser Funktionsparameter Models, als 3-Tupel dargestellt $\tau_i = (C_i, T_i, D_i)$. Bei sporadischen Funktionen wird die Periode T_i ersetzt durch den Parameter „minimale Aufrufzeit zweier Berechnungen“ $\{A_i\}$. Durch diese 3 Parameter lässt sich die Auslastung für eine einzelne Funktion berechnen, anhand Gl. (1).

(1) $U_n = C_i/T_i$ bzw. $U_n = C_i/A_i$

Dieses 3-Funktionstupel ist für Echtzeitbetriebssysteme mit zeitlichen Einschränkungen noch um die Parameter [3]:

4) Bereitzeit: r_i
5) Startzeit: s_i
6) Abschlusszeit: e_i

erweitert worden, zu einen 6-Tupel $\tau_i = \{C_i, T_i, D_i, r_i, s_i, e_i\}$, welches einen Systemstart/-Ende mit einbezieht, das unter rein RMS geschedulten Systemen aber nicht betrachtet wird [3]. Für eine statische Software System Konfiguration auf einem Single oder Multi-Core Prozessor sind diese Funktionsparameter ausreichend um mit der nächste Stufe der Schedule Erstellung fortzufahren, bzw. bei statischen Multi-Core Systemen mit einer Zwischenstufe [5] [6] [7].
Bei Multi-Core Prozessoren in einer statische Software System Konfiguration muss zusätzlich die Verteilung der einzelnen Funktionen auf die Cores erfolgen, was als Behälterproblem (Bin Packing) behandelt wird. Ein bekannter statischer offline Algorithmus für dieses Problem ist der „Ratenmonotone First Fit" Algorithmus, welcher nach einem Single-Core Einplanungstest prüft, ob die Summe der allokierten Funktionen pro Core einen bestimmten Scheitelwert der Auslastung nicht überschreitet [9] [10] [11].
Anhand des zugrundeliegenden Scheduling Algorithmus für das System ist der Single-Core Einplanungstest gegeben. Der Einplanungstest sagt aus, ob alle Echtzeitfunktionen ihre Deadline einhalten können oder nicht. Der am häufigsten verwendete hart echtzeitfähige Scheduling Algorithmus ist der „Rate Monotonic Scheduling"(RMS) Algorithmus für eine statische Software System Konfiguration. In RMS lassen sich alle Funktionen ohne weitere Prüfung ohne Deadline Verletzung durchführen wenn die Summe der einzelnen Funktionauslastungen, Gl. (1), pro Core nicht ln(2) (≈ 69%) übersteigen [4] [12].

4. PHASENANALYSE FÜR HAMS

Da aktuelle Echtzeitsysteme eine für Single-Core optimierte, statische Software System Konfiguration inne haben, ist es möglich mehr Funktionen auf einen Multi-Core System mit einer rekonfigurierbaren Software System Konfiguration zu allokieren. Um dieses Ziel der höheren Effizienz zu erreichen, beschreiben wir hier den Ansatz, Funktionen in ihre einzelnen Phasen einzuordnen um alle möglichen Systemkonfigurationen zu ermitteln (Phasenanalyse). Dazu muss das bestehende Funktionsparameter Modell für Multi-Core Prozessoren in den Bereichen Basis-Schedule Funktionsparameter und phasenabhängige Laufzeitparameter erweitert werden. Ebenso muss ein Modell zu Phasenabhängigen logischen Funktionssetdarstellung und zur Multi-Core Systemmodell Darstellung generiert werden.

4.1. Basis-Schedule Funktionsparameter für Echtzeitfunktionen

Zur Beschreibung einer Funktion in einem rekonfigurierbaren System, wie HAMS, wird ein Set von Basis-Schedule Parameter benötigt um die grundlegende Allokierung und Ausführung einer Funktion zu bestimmen. Bei diesen Parametern für eine Funktion τ_i handelt es sich um folgendes 6-Tupel:

(2) $\tau_i = (SC_i , \{NP_a\}_i, (FP_m)_i , CR_i , RO_i)$

Die Bedeutung der einzelnen Parameter ist wie folgt:

- SC_i : Ist die „Scheduling Class" einer Funktion. Um einzuordnen welche Scheduling Bedürfnisse (d.h. harte oder weiche Echtzeit) eine Funktion benötigt, muss die zugrundeliegende Scheduling Klasse angegebenen werden. So ist es möglich diese Funktionen logisch zu trennen aber dennoch auf einem Multi-Core System auszuführen. In HAMS ist RMS für harte Echtzeit und Earliest Deadline First(EDF) für weiche Echtzeit vorhanden [3] [10].
- NP_a : Benötigte eine Funktion spezielle Peripherie, z.B. Ethernet oder RS422 Schnittstellen, wird das in diesem Parameter „Needed Peripherals" angegeben. Da ein Core in einem Multi-Core System nicht immer Zugriff auf jede Peripherie hat, muss zur Basisallokierung der Funktion ein Core ausgewählt werden, welcher Zugriff auf diese Peripherie hat. Dieser Parameter ist eine Aufzählung.
- FP_i : Dieser Parameter „Function Phases" steht für die unterschiedlichen Phasen, die ein Task einnehmen kann und ist ein Tupel in sich. Die Parameter des Sub-Tupels werden in Kapitel 4.2 beschrieben.
- CR_i : Ist das „Criticality Level" einer Funktion. Bei einem temporären Funktionsfehlverhalten wird mit diesem Parameter sichergestellt, dass kritische Funktionen vom Scheduler ausgewählt und zuerst berechnet werden. Ansonsten könnten aufgrund des Scheduling Algorithmus auch unkritische Funktionen vor kritischen gescheduled werden. Dieser Parameter trägt zur ordnungsgemäßen Ausführung des HAMS Schedulers bei, wird aber bei der Baisallokation von Funktionen nicht weiter benötigt.
- RO_i : Bei diesem Parameter handelt es sich um die „Run Once" Eigenschaft einer Funktion. Sollte es aus migrationsbedingt möglich sein einen Parameter innerhalb einer Periode mehrmals zu berechnen, dieses aber aufgrund des funktionsspezifischen Aufbaus nicht erlaubt sein, wie bei Filteralgorithmen, gibt dieser Parameter an, dass die Funktion nur einmal in der Periode laufen darf. Dies ist ein Parameter welcher bei der Funktionsmigration benötigt wird.

Mit diesen Basis-Schedule Parametern können die Funktionen grundlegend eingezuordnen werden. Zur weiteren Phasenanalyse werden aber noch phasenabhängige Laufzeitparameter und eine logische Funktionssetdarstellung benötigt.

4.2. Phasenabhängige Laufzeitparameter einer logischen Funktionsphase

Da in einem rekonfigurierbaren Software System, wie HAMS, das Wissen über die unterschiedlichen Phasen einer Funktion zur Systemrekonfiguration genutzt wird, ist eine spezielle Darstellung für die einzelnen phasenabhängigen Laufzeitparameter einer Funktion nötig. Diese Parameter sind Bestandteil des Sub-Tupels FP_m der Basis-Schedule Parameter und abhängig von der Anzahl der Phasen der Funktion. Die Parameter des Sub-Tupels FP_m für eine periodische harte Echtzeitfunktion lauten wie folgt:

(3) $FP_m = (CP_m , TP_m , DP_m , CMP_m)$

- CP_m : Ist die phasenabhängige WCET für die zu definierende Phase m. Je nach z.B. Fahrsituation wird eine Funktion eine andere Berechnungszeit benötigen. Mit diesem Parameter ist es möglich die WCET für die aktuelle Phase anzugeben. Zugleich ist die WCET auch ein Fehlverhaltensindex der Funktion für den Scheduler. Benötigt ein Task ohne ersichtlichen Grund mehr Zeit zur Ausführung, liegt

mit größter Wahrscheinlichkeit ein Funktionsfehler vor [3].

- TP_m : Ist die Periode für die zu definierende Funktionsphase. Analog zur Änderung der WCET kann sich auch die Periode einer Funktion je nach Phase verlängern oder verkürzen. Mit der Änderung der WCET und der Periode verändert sich somit auch die Auslastung der Funktion, Gl. (1) .
- DP_m : Ist die Deadline für die aktuelle Phase der Funktion. Neben der WCET und der Periode kann sich auch die Deadline einer Funktion phasenabhängig ändern. Bei RMS geschedulten Systemen wird $TP_m = DP_m$ zur Vereinfachung angenommen [3] [4] [9]. In einem HAMS System kann mit Hilfe der offline Funktionsverteilung diese Vorrausetzung aufgehoben werden.
- CMW_m : Ist eine Prozentangabe der Verlängerung der WCET einer Funktion nach der Migration der Funktion auf einen anderen Core. Da sich die Ausführungszeit einer Funktion nach der Migration auf einen anderen Core durch Cache Misses, u.A. bei Multi-Core Systemen ohne gemeinsame L0/L1/L2 Caches, verlängern kann [13], ist es nötig diesen Parameter aufzuführen. So kann der Scheduler überprüfen ob in einem Einplanungstest alle Deadlines eingehalten werden oder ggf. Gegenmaßnahmen einleiten.

Die Parameter für eine sporadische, weiche Echtzeitfunktion, unter EDF, lauten wie folgt:

(4) $TP_m=(CP_m , AP_m , DP_m , CMP_m , CEP_m)$

Im Vergleich zu einer periodischen, hart echtzeitfähigen Funktion, wird der Parameter TP_m durch AP_m ersetzt und der Parameter CEP_m hinzugefügt. Die Bedeutung der geänderten Parameter ist wie folgt:

- AP_m : Ist die minimale Aufrufzeit zweier Berechnungen für die Funktion. Je nach Phase kann sich die Zeit zwischen zwei Aufrufen derselben Funktion verkürzen oder verlängern.
- CEP_m : Ist die zusätzliche Ausführungszeit in Prozent einer Funktion für die zu definierende Phase. Da weiche Echtzeitfunktionen, laut Definition [3] [10], nicht unbedingt ihre Deadlines und WCETs einhalten müssen, kann das in einen rekonfigurierbaren System u.U. zu Problemen führen, nämlich dann wenn diese Funktion den Core nicht mehr frei gibt, für niederpriore Funktionen. Um durch dieses Funktionsverhalten nicht das Gesamtsystem zu gefährden, muss eine maximale Ausführungszeit definiert werden, ab der eine Funktion als fehlerhaft erkannt wird.

4.3. Phasenabhängige logische Funktionssetdarstellung

Zur Darstellung und Schedule Erstellung wird ein Modell benötigt, das logische Funktionsabhängigkeiten aufgrund der unterschiedlichen Funktionsphasen untereinander aufzeigt. Bei dieser Phasenabhängigkeit handelt es sich nicht um eine Abhängigkeit durch Semaphore oder anderen Synchronisationsmechanismen, sondern ausschließlich um eine WCET Abhängigkeit. Aus diesem Grund wird folgendes 3-Tupel LL_b benötigt, welches sich mit den bereits definierte Funktionen τ und deren Phasen bedient.

(5) $LL_b=(LF_b , (LDF_b) , (AFP_b))$

- LF_b : Gibt an welche Funktion τ bestimmend für alle anderen logisch abhängigen Funktionen ist. Das heißt, sollte die Funktion genau diese Funktionsphase erreicht haben und sie ist eine führende Funktion „Leading Function“ so müssen alle anderen logisch abhängigen Funktionen automatisch die entsprechende Funktionsphase einnehmen. Nehmen abhängige Funktionen nicht die entsprechende Funktionsphase ein wird das als Fehlverhalten gewertet.
- LDP_b : Hier werden die logisch abhängige Funktion τ „Logically Depended Functions“ aufgezählt. Z.B. steht bei einem Tempomat hier der Verweis auf die Funktion Einparkhilfe, denn beide Funktionen dürfen nicht gleichzeitig eine aktive Fahrerassistenz durchführen.
- AFP_b : Hier werden für alle in der LF_b vorkommende Phasen die korrekten Phasen der LDF_b verknüpft „Available Function Phases“. Diese ist einer Aufzählung der Art $(\tau_{lf\,x}; \tau_{ldf\,y}, \ldots) \triangleq (FP_{lf\,x}; FP_{ldf\,y.z}, \ldots)$.

4.4. Phasenunabhängiges Systemmodell für Multi-Core Prozessoren

Zur Beschreibung eines rekonfigurieren Software Systems auf Multi-Core Basis wird auch eine Beschreibung des Multi-Core Prozessors (Systemmodell) benötigt, um Funktionen auf den passenden Core zu allokieren oder zu migrieren. Folgende Multi-Core Parameter, dargestellt in einem 6-Tupel MC_a, werden benötigt um einen Schedule zu erstellen:

(6) $MC_a=((SC)_a , (CCF_a) , GCF_a, (AM_a) , (AP_a) , (AI_a))$

- SC_a : Dieser Parameter „Synchron Cores“ ist eine Auflistung und gibt an welche Cores im Gesamtsystem synchron laufen, d.h. ein synchrones Taktsignal bekommen für den Scheduler Tick [14] [15]. Dieser Parameter ist wichtig für die Migration und schließt untereinander nicht synchrone Cores von der Migration in HAMS aus.
- CCF_a : Dieser Parameter „Core Clock Frequencies” gibt an welche Taktfrequenzen ein Multi-Core Prozessor einnehmen kann.
- GCF_a : Da Multi-Core Prozessoren, je nach Hersteller die Core Frequenz einzeln oder nur gesamt ändern können gibt dieser Parameter an ob es eine Prozessor-weite gemeinsame Frequenz gibt „Global Common core Frequency“ oder ob jeder Core seine Frequenz individuell steuern kann.
- AM_a : Dieser Parameter steht für „Accessible Memory”. In einem Multi-Core System ist nicht jeder Speicherbereich für jeden Core zugänglich, vor allem bei sicherheitsrelevanten Multi-Core Systemen [14] [17]. Dies hat Auswirkungen auf die Migration von Funktionen zwischen Cores und muss bei der Schedule Erstellung berücksichtigt werden.
- AP_a : In einem Multi-Core System sind nicht immer alle Peripheriegeräte für jeden Core sichtbar. Darum muss angegeben mit dem Parameter „Accessible Peripherals“ werden, welche Peripherie welchen Core zugeordnet ist.
- AI_a : Auch die Interrupts, „Accessible Interrupts“ sind nicht für alle Cores zugänglich. So können analog zur Peripherie nur bestimmte Interrupts einem Core zugewiesen werden.

5. ALLOKATION VON FUNKTIONEN IN EINEM PHASENABHÄNGIGEN SYSTEM

Um anhand diesem Systemmodell und Funktionsparameter einen gültigen, phasenabhängigen Schedule zu erstellen müssen die vorherrschenden Bin Packing Algorithmen für eine Phasenanalyse angepasst werden. Als Beispiel für einen Bin Packing Algorithmus dient der „First Fit“ Bin Packing Algorithmus.

5.1. Basis Bin Packing Algorithmus

Im First Fit Bin Packing Algorithmus (FF) werden zuerst alle Tasks τ_i ungeordnet in einem Array UQ aufgelistet. Zusätzlich werden die Bins (Behälter) Ψ aufgeführt. Im Falle des FF Algorithmus hat jede Bin δ_a von Ψ die gleiche Größe. Das bedeutet, für den Einplanungstest des zugrundeliegenden Schedulingalgorithmus wird als Bingröße dessen maximaler Schwellwert angenommen. Für RMS mit einem exakten Liu & Layland Einplanungstest liegt dieser Wert bei 69% für die Funktionsauslastung auf einer Bin ohne weitere Einplanungsprüfung [4] [10].

```
program Basis First-Fit Algorithm
UQ:= [τ1 … τi]
Ψ:= [δ1 … δa]
Φ:=[ωδ1 … ωδa]
   while UQi ≠ 0
      while Ψj ≠ 0
      if Uτi + Uδj <= 69%  //Schwellwertcheck
         Uδj = Uδj + Uτi    //Bin Auslastungsakt.
         ωδj → Uτi          // Funktion Bin-Zuweisung
      else if Ψj+1≠ 0
         return (“not allocatable“)
      end if
      end while
   end while
return (“allocatable“)
end program Basis First-Fit Algorithm
```

BILD 1. Allgemeiner FF Algorithmus

Im FF Algorithmus selbst wird beginnend mit der ersten Funktion τ1 deren Auslastung $U_{\tau 1}$, berechnet anhand von Gl. (1). Im Falle des allgemeinen FF Algorithmus (BILD 1) wird diese Funktion der ersten Bin zugeordnet. Überschreitet die Summe an Funktionsauslastung $U_{\delta j}$, der Bin zugewiesenen Funktionen den Wert 69% nicht, wird diese Funktion auf dieser Bin allokiert $U_{\delta j} = U_{\delta j} + U_{\tau i}$. Sollte dieser Schwellwert für die aktuelle Bin doch überschritten werden, wird die nächste Bin überprüft. Der Algorithmus ist beendet wenn alle Funktionen einer Bin zugeordnet sind und zugleich der Schwellwert nicht überschritten wurde. Ist keine passende Bin mehr vorhanden, d.h. der Schwellwert wird in jeder verfügbaren Bin mit Allokation der Funktion überschritten, ist der FF Algorithmus in dieser Form fehlgeschlagen (BILD 1).

5.2. Phasenanalyse mit phasenabhängigen First Fit Bin Packing Algorithmus

Da der Basis FF Algorithmus kein phasenabhängiges System miteinbezieht, müssen Anpassung in der Anordnung der Tasks, d.h. bei der Erstellung der ungeordneten Auflistung der Tasks UQ, der Bin Ψ, sowie in der Allokierung selbst vorgenommen werden.
Die Erstellung der Auflistung UQ kann mit Hilfe der über die Funktionen zu Verfügung stehenden Parameter aus Kapitel 4 vorgenommen werden. Hierbei werden nur die für die Allokierung wichtigen Parameter benötigt, Parameter für Migration und Fehlererkennung werden in dieser Darstellung des FF Algorithmus nicht berücksichtigt, sowie davon ausgegangen wird dass die WCETs der einzelnen Phasen korrekt ermittelt wurden. Basis UQ wird erstellt aus dem Tupel LL, Gl. (7).

(7) $UQ:=[LL_1 , \dots , LL_A]$

Folglich müssen die einzelnen FP-Tupel aufgelöst werden, Gl. (8).

(8) $UQ:=[(LF_1; (LDF)_1)_1 , \dots , (LF_a; LDF_a)_a]$

Zur weiteren Aufteilung, welche Funktionen in welcher Phase aktiv sind, um somit eine Summe der Auslastung der Phase für das System zu bekommen, müssen mit AFP („Available Function Phases“) die FP („Function Phases“) der LF („Leading Function“) und aller LDFs („Logically Dependent Functions“) bestimmt werden. Das für die FF-Algorithmus benötigte logische UQ ist beschreiben in Gl. (9).

(9) $UQ:=[((FP_{LF1.1}; FP_{LDF1.3,\dots})_1 \vee, (FP_{LF1.2}; FP_{LDF1.2,\dots})_1 \wedge , \dots, \wedge (FP_{LFa.x}; FP_{LDFa.y,\dots})]$

In Gl. (9) ist jede mögliche logische Phase im System aufgelistet ausgehend von der Summe aller LFs. Da aber immer nur eine Phase einer LF und deren LDFs laufen kann, müssen diese mit einer „oder“ Verknüpfung dargestellt werden. In Gl. (9) ist dies anhand der Indizes veranschaulicht, wobei die Zahl nach dem Punkt die Phase angibt. So ist die Funktion $FP_{LDF1.3}$ in der Phase 3 wenn die $FP_{LF1.1}$ in der Phase 1 ist. Sollte $FP_{LF1.2}$ in der Phase 2 sein so ist auch $FP_{LDF1.2}$ in der Phase 2. Die LFs untereinander werden mit einer und Verknüpfung verbunden. Zur späteren vereinfachten Allokation wird das Tupel UQ in Θa (Phasen-Funktionstupel) und Θb (Logisches-Funktionstupel) aufgeteilt, Gl. (10). Die Funktionen und Phasen sind ungeordnet in Bezug auf ihre Auslastung, wie im FF-Algorithmus üblich, aber Funktionen mit unterschiedlicher SC (Scheduling Class) werden aussortiert.

(10) $\Theta a:=[((FP_{LF1}; FP_{LDF2.1/1,\dots})_1 \vee \dots \vee (FP_{LF1}; FP_{LDF2.1/d,\dots})_d)]$
(11) $\Theta b:=[\Theta a_1 \wedge \dots \wedge \Theta a_d]$

Auch die Tupel für die Bins muss geändert werden. In einem Multi-Core System, entsprechen die Bins den einzelnen Cores.

(12) $\Psi:= [MC_1 , \dots , MC_A]$

```
program Allocate Limited Functions
while τΘb i ≠ 0
  while Ψj ≠ 0
    if (τNP)Θbi ≙ (MCAP)ψj  && //Peripheriecheck
      (τNI)Θbi ≙ (MCAI)ψj  &&  //Interruptcheck
      MC ψj =0                  //Arraycheck
        Ψj → τΘb i  // Funktion Bin-Zuweisung
    else
      return ("not allocatable")
    end if
  end while
end while
return ("allocatable")
end program Allocate Limited Functions
```

BILD 2. Allokierung limitierter Funktionen

In BILD 2 ist der Algorithmus zur Allokierung von Peripherie/Interrupt limitierter Funktionen als Teil des phasenabhängigen FF Algorithmus dargestellt. Diese Allokierung muss als Erstes durchgeführt werden, denn nur so kann sichergestellt werden, dass Funktionen auch Zugriff auf einen Core mit korrekter Peripherie bekommen. Es wird überprüft, ob auf dem Core alle benötigte Peripherie und Interrupts vorhanden sind. Gibt es nur einen Core mit diesen Eigenschaften wird die Funktion und alle Ihre Phasen auf diesem Core allokiert. Gibt es ein mögliches Subset an Cores auf denen die Funktion allokiert werden kann wird dies für den späteren Systemcheck im eigentlichen FF Algorithmus vermerkt. Ist es möglich die Funktion auf allen Cores des Systems unterzubringen, wird die Funktion nicht weiter betrachtet.
Als zweiter Schritt in dem FF Algorithmus für Phasen (siehe Bild 3) wird in Loop1 die Auslastung der ersten Phase einer Funktion Θa_1, aus dem logischen-Funktionstupel Θb [$U_{\Theta a\ [\Theta b]}$], welches noch nicht allokiert wurde, genommen und dem Schwellwertcheck / Systemcheck unterzogen. Bei diesem Schwellwertcheck wird die maximale Auslastung der aktuellen Bin herangezogen und mit der Funktionsauslastung der zu untersuchenden Phase auf Überschreitung des Schwellwerts überprüft. Dabei ist zu beachten, dass die maximale Auslastung der Bin phasenabhängig ist. Daher werden für Funktionen aus Θb, welche nicht zum aktuellen Logischen-Funktionstupel gehören, die Maximalwerte für die Auslastung genommen. Der Grund dafür ist das nicht logisch verknüpfte Funktionen gleichzeitig mit maximaler WCET laufen können. Wird das nicht beachtet, bekommen die Funktionen zu wenig Rechenzeit, was zu Deadline Misses führt.
Ist diese Prüfung erfolgreich, wird in Loop2 untersucht, ob eine Allokierung dieser Funktion und ihrer Phasen auf dieser Bin(Core) noch zu einem gültigen ausführbaren Schedule im Gesamtsystem führt, wenn das Maximum aller logisch unabhängigen Funktionen zum Schwellwertcheck herangezogen wird. In der „Loop 2" werden nur logisch unabhängigen Funktionen untersucht, welche mit einer „und" Verknüpfung verbunden sind und so unabhängig voneinander mit maximaler WCET rechnen. Daher werden nur die Maxima für diese Funktionen herangezogen. Durch diese Prüfung lässt sich feststellen, ob ein gültiger Schedule entstehen kann wenn ein Logisches-Funktionstupel auf mehrere Cores verteilt ist.
War diese Überprüfung erfolgreich, werden nach dem FF Prinzip alle restlichen Funktionen und deren Phasen auf die Cores verteilt. Ist die Allokierung erfolgreich, wird so ein Schedule Φ erzeugt der für jedes logische Funktionsset und deren abhängigen Phase ausführbar ist.

```
program First-Fit Algorithm with Phases
Θa:=[(LF1; (LDF)1)1 ∨…∨ (LF a; LDF a)a]
Θb:=[Θa1 ∧ … ∧ Θaa)]
Ψ:= [MC1 , … , MCm]
Φ:=[ωMC1 … ωMCm]
Allocate_Limited_Functions()
while Θb ≠ 0
  while Ψj ≠ 0
    while UΘa [Θb] ≠ 0 && UΘb ↛ Φ //Loop 1
      //Schwellwert- /Systemcheck
      if UΘa i + max(UMC j )<=69% && τΘb i = Ψ MC j
        UMC j=UMC j + UUQa i //Bin Ackt.
        while max(UΘb ∉ UΘa i) ≠ 0 //Loop 2
          while UMC g ≠ 0
            if max(UΘb f) +UMC g>=69%
              return ("not allocatable")
            end if
            end while
        end while
        ωMC j → UΘa i  //Funktionsphase Plan-Zuw.
        else
          return ("not allocatable")
        end if
    end while
  end while
end while
return ("allocatable")
end program First-Fit Algorithm with Phases
```

BILD 3. FF Algorithmus für Phasen

6. RESULTAT EINER FUNKTIONSALLOKATION FÜR PHASENABHÄNGIGE SYSTEME

Zur Verdeutlichung der vorher beschriebenen Funktionsparameter und dem phasenabhängigen FF Algorithmus, wird folgendes Beispiel aus dem E/E System eines Automobils herangezogen:
Es existieren 3 Funktionen: der Tempomat (t), die Einparkassistent (p) und der Regensensor (r) mit ihren Funktionsphasen: Diese Funktionen sind mit Ihren Perphierieabhängigkeiten und den jeweiligen Funktionsphasen beschrieben.

- $\tau_t=(0_t , RMS_t , CAN_t , false_t , (FP_{1\text{-}3})_t)$
  - $FP_{1t}=(3_{1t} , 5_{1t} , 5_{1t} , 0_{1t})$
  - $FP_{2t} =(1_{2t} , 2_{2t} , 2_{2t} , 0_{2t})$
  - $FP_{3t}=(1_{3t} , 6_{3t} , 6_{3t} , 0_{3t})$
- $\tau_p=(1_p , RMS_p , (CAN, ETH)_p , false_p , (FP_{1\text{-}3})_p)$
  - $FP_{1p}=(1_{1t} , 6_{1p} , 6_{1p} , 0_{1p})$
  - $FP_{2p} =(1_{2p} , 2_{2p} , 2_{2p} , 0_{2p})$
  - $FP_{3p}=(3_{3p} , 5_{3p} , 5_{3p} , 0_{3p})$
- $\tau_r=(3_r , RMS_r , CAN_r , false_r , (FP_{1\text{-}2})_r)$
  - $FP_{1r}=(1_{1r} , 10_{1r} , 10_{1r} , 0_{1r})$
  - $FP_{2r}=(1_{2r} , 2_{2r} , 2_{2r} , 0_{2r})$

Tempomat und Einparkhilfe sind voneinander logisch abhängige Funktionen. Das Umschaltereignis ist in diesem Beispiel die Fahrzeuggeschwindigkeit (v). Daraus ergeben sich die Phasen: v≤30 km/h und v>30 km/h. Der Regensensor hat keine Abhängigkeiten zu den anderen

Funktionen. Dieser hat 2 Phasen für „Ausgeschaltet“ (FP_{1r}) und „Eingeschaltet“ (FP_{2r}). Die phasenabhängige logische Funktionsdarstellung, wie Gl. (5), ist folgende:

- $LL_{v\leq 30\ km/h} = (\tau_t\ ,\ \tau_p,\ (FP_{3t} \wedge (FP_{2p} \vee FP_{3p}))$
- $LL_{v>30\ km/h} = (\tau_t\ ,\ \tau_p,\ ((FP_{1t} \vee FP_{2t}) \wedge FP_{1p}))$

Der Multi-Core Prozessor ist mit folgendem Tupel spezifiziert und reduziert auf die wichtigsten Parameter zur Schedule Erstellung:

MC_0= ((0,1) , 300Mhz, true, (AM_a) , CAN, ETH, (CAN, ETH))
MC_1= ((0,1) , 300Mhz, true, (AM_a) , CAN, ETH, (CAN, ETH))

Wird der phasenabhängige FF Algorithmus eingesetzt zur Allokierung der Funktionen auf das Multi-Core System werden die Ergebnisse folgende sein:

1) Die Allokierung von limitierten Funktionen wird ergebnislos beendet, da sowohl Core 0 also auch 1 Zugriff auf die benötigten Ressourcen haben

2) Für die Phase v≤30 km/h gilt die Verteilung:

Active:	Core 0	Core 1
FP_{2p}	$FP_{3t} \wedge FP_{2p}$	$FP_{1r \vee 2r}$
FP_{3p}	$FP_{3t} \wedge FP_{1r \vee 2r}$	FP_{3p}

TAB 1. Verteilung v<=30km/h

Entsprechend dem Algorithmus wird die LP (Tempomat) zuerst zugeteilt, hier FP_{3t}. Da sowohl FP_{2p} als auch FP_{3p} aktiv sein können, werden beide Funktionsphasen untersucht. Weil die Auslastung von FP_{3t} + FP_{3p} größer ist als 69% wird FP_{3p} auf Core 1 allokiert wenn FP_{3p} aktiv ist. Der Regensensor wird je nach aktiver Phase Core 0 oder 1 zugewiesen.

3) Für die Phase v>30 km/h gilt die Verteilung:

Active:	Core 0	Core 1
FP_{1t}	FP_{1t}	$FP_{1r \vee 2r} \wedge FP_{1p}$
FP_{2t}	$FP_{2t} \wedge FP_{1p}$	$FP_{1r \vee 2r}$

TAB 2. Verteilung v>30km/h

7. ZUSAMMENFASSUNG UND AUSBLICK

Der Ansatz anstatt einer statischen Systems mit fest vorgegebenen WCETs für Funktionen ein rekonfigurierbares System zu verwenden zeigt vielversprechende Ergebnisse. Im Beispiel von Kapitel 6 wäre es nicht möglich ohne explizite Funktionsphasen ein Dual-Core System zu verwenden, denn es ist laut Einplanungstest nicht zulässig eine Auslastung von 100% auf einen Core zu haben TAB 3

	Statische Software Systeme	**Rekonfigurierbare Software Systeme**
Core	Max. Auslastung	Max. Auslastung
0	*100% !*	66%
1	50%	66%

TAB 3. Maximale Ausführungszeiten von Kapitel 6

Die nächsten logischen Schritte sind den phasenabhängigen FF Algorithmus zu erweitern und auch die Funktionsparameter für Migration und Aufstarten eines Systems mit einzubeziehen und mit anderen Bin Packing Algorithmen, wie z.B. Best Fit oder Worst Fit für phasengesteuerte Systeme, zu vergleichen. Sind diese Algorithmen bekannt und ausgearbeitet müssen sie in der Realität überprüft werden. Dazu dient ein Linux Kernel, welcher mit dem HAMS Scheduler erweitert wurde, und auf einem Multi-Core System lauffähig ist. Mit Hilfe dieses Systems lassen sich die Algorithmen nicht nur in der Theorie sondern auch in der Praxis evaluieren und Rückschlusse auf die Allokation von Funktionen ziehen.

Schrifttum

[1] T. Hanti, M. Ernst und A. Frey. „Higher Utilization of Multi-Core Processors in Dynamic Real-Time Software Systems“. *Int. Journal of Electrical Engergy* .Vol.1 No.4. 2013.

[2] S. Seiden. „New Bouns for variable-sized online Bin Packing“. *29th International Colloquium on Automata, Languages and Programming*. 2002.

[3] D. Zoebel. „Echtzeitsysteme: Grundlagen der Planung“. Springer. Koblenz. 2008.

[4] L. Liu, W. Layland. „Scheduling algorithms for multiprogramming in a hard-real-time environment“. *Journal of the ACM*. 1973.

[5] M. Garey, D. Johnson. „Computers and Intractability: A Guide to the Theory of NP-Completeness“. W. H. Freeman and Company. San Francisco U.S.A. 1979.

[6] A. Fiat, G. Woeginger. „Online Algorithms: The State of the Art“. Springer. Koblenz. 1998.

[7] J. Goossens, S. Baruah, S. Funk. „Real-time Scheduling on Multiprocessors“. *10th International Conference on Real- Time Systems*. 2002.

[8] P. Lehoczky, L. Sha, Y. Ding. „The rate monotonic scheduling algorithm: exact characterization and average case behavior“. *IEEE Real Time Systems Symposium*. 1989.

[9] S. Lauzac, R.Melhem, D.Mosse. „An Efficient RMS Admission Control and its Application to Multiprocessor Scheduling“. *International Parallel Processing Symposium*. 1998.

[10] C.L. Liu, J.W.Layland. “Scheduling Algorithms for Multiprogramming in Hard Real-time Environment“. *Journal of ACM* 20(1). 1973.

[11] J. Lopez, J. Diaz, M. Garcia, D. Garcia. „Worst-Case Utilization Bound for EDF Scheduling on Real-Time Multiprocessor Systems“. *12th Euromicro Workshop on Real-Time Systems*. 2000.

[12] J. Lopez. „Utilization Based Schedulability Analysis of Real-time systems Implemented on Multiprocessors with Partitioning Techniques“. *Ph.D. Thesis*. University of Oviedo. 2001.

[13] H. Negi, T. Mitra, A. Roychoudhury. „Accurate Estimation of CacheRelated Preemption Delay“, international conference on Hardware/Software. 201-206, 2003.

[14] A. Jerraya, W. Wolf. „Multiprocessor Systems-on-Chips (Systems on Silicon)“.Elsevier. San Francisco U.S.A. 2004.

[15] Infineon. „Highly Integrated and Performance Optimized 32-bit Microcontrollers for Automotive“. www.infineon.com/TriCore. 2014.