Fragenausarbeitung Embedded Systems Engineering VO WS 2010/11

Franz Preyser

9. März 2011

Inhaltsverzeichnis

Vorwort

Diese Fragenausarbeitung habe ich für mich, zum Zweck des Lernens für die Prüfung verfasst. Die Fragen stammen aus dem VO - Wiki, wurden also bereits von anderen Studierenden zusammengetragen. Die Informationen mittels derer ich die Fragen beantwortet habe, stammen zum einen aus den Vorlesungsfolien und dem Buch [HK] und zum anderen aus Wikipedia.org ergänzt durch Wissen, welches ich glaube aus den VOs und vorhergehenden LVAs mitgenommen zu haben. Aus diesem Grund geben ich keine Garantie auf Vollständigkeit oder Richtigkeit der Inhalte. Also alle Angaben ohne Gewähr. Ich stelle zusätzlich zu diesem pdf auch noch die .tex-files online, falls jemand die Ausarbeitung erweitern, vervollständigen oder Fehler ausbessern möchte.

1 MFM-Codierung

Modified Frequency Modulation ... eine Leitungscodierungsart
Es wird zwischen Takt- und Daten-Zeitpunkten unterschieden. Ein Signalwechsel zu einem Daten-Zeitpunkt bedeutet eine logische 1. Kein Signalwechsel zu einem

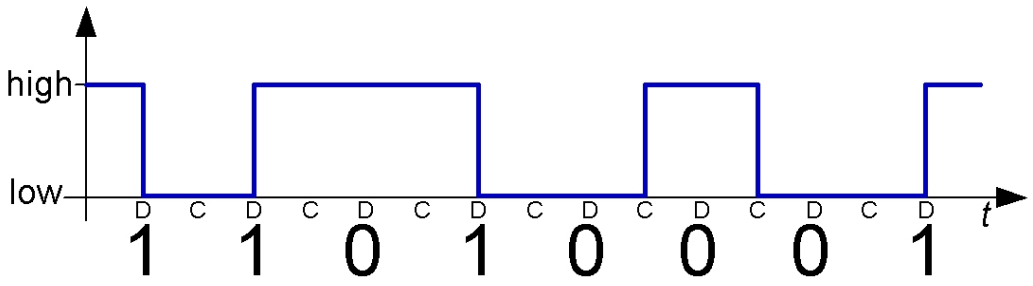

Abbildung 1: Modified Frequency Modulation

Daten-Zeitpunkt bedeutet eine logische 0. Treten zwei logische 0er hintereinander auf, so wird zur Erleichterung der Taktrückgewinnung, ein Signalwechsel bei dem Takt-Zeitpunkt zwischen den beiden 0en eingefügt. Symbolrate entspricht der Bitrate. Verwendung: Leitungscodierung, Datencodierung auf magnetischen Datenspeichern(Magnetband, Diskette)

2 Manchester-Codierung

... eine Leitungscodierungsart.
0 wird als fallende Flanke, 1 als steigende Flanke codiert. Diese Flanke müssen

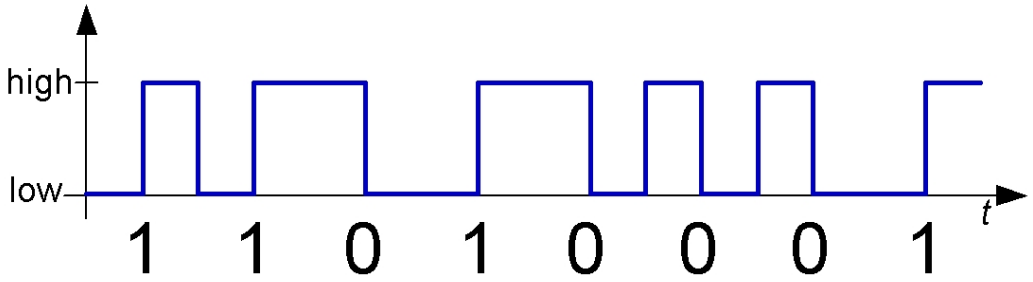

Abbildung 2: Manchester-codierter Datenstrom

zu einem bestimmten Taktzeitpunkt passieren. Folgen zwei logisch gleiche Werte aufeinander, so wird dazwischen eine weiter Flanke eingefügt. Z.B.: wenn zwei 0er aufeinander folgen, muss dazwischen eine steigende Flanke eingefügt werden, damit nach der ersten 0(fallende Flanke) eine weiter fallende Flanke auftreten kann. Diese eingefügten Signalflanken unterscheiden sich von den Datenflanke im Taktzeitpunkt, zu dem sie auftreten. Ein Nachteil ist, dass die Symbolrate die doppelte Bitrate(z.B. bei einer 0-Folge) annehmen kann. Ein Vorteil ist die

Gleichstromfreiheit, wodurch eine Taktrückgewinnung möglich ist. Anwendung: 10 MBit/s - Ethernet, AS(Aktuator-Sensor) - Interface

3 Ersatzschaltbild für einen Conductor Path

Leiterbahn-Ersatzschaltbild: R, L, C Alle drei parasitären Größen R, L und C

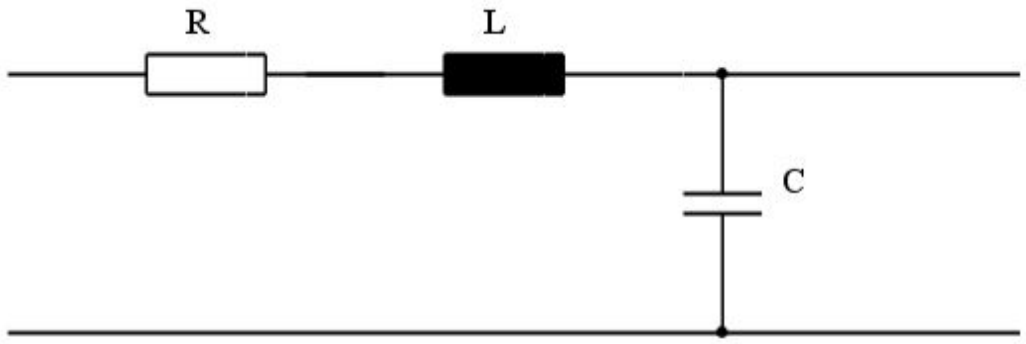

Abbildung 3: Ersatzschaltbild für ein Stück Leiterbahn.

hängen von der Leiterbahnlänge ab. Zusätzlich hängt der ohmsche Widerstand R von der Leiterbahnquerschnittsfläche ab(Kupferwiderstand), die Induktivität L von der Leiterbahngeometrie(Schleifen) und die Kapazität C von der Nähe der Leiterbahn zu benachbarten(parallel verlaufenden) Leiterbahnen.

4 Sampling/Nyquist-Shannon-Abtasttheorem

Das Abtasttheorem von Nyquist-Shannon besagt, dass ein mit f_{max} band-begrenztes Signal mit mindestens der doppelten Frequenz $f_S \geqslant 2 \cdot f_{max}$ abgetastet werden muss um theoretisch das ursprüngliche Signal aus den Abtastwerten fehlerfrei/-verlustfrei wiedergewinnen zu können. Praktisch bräuchte es dazu ein perfektes Tiefpassfilter.

5 Minislotting/Arinc629

Minislotting bezeichnet eine zeitgesteuerte “Medium Access” - Strategie. Dabei wird die Zeit in Minislots eingeteilt, die länger als die längste Signallaufzeit sein müssen. Bevor ein Knoten senden darf, muss für mindestens ein bestimmte Anzahl an Minislots “Ruhe” am Bus geherrscht haben. Diese Anzahl an Minislots ist für jeden Knoten verschieden, wodurch eine Prioritätsrelation auf der Menge

der Knoten eingeführt wird. Ein Beispiel, bei dem Minislotting angewandt wird, ist das Aring629 - Protokoll, benannt nach der US-amerikanischen Firma Aeronautical Radio Inc.. Es handelt sich hierbei außerdem um ein "Waiting Room - Protocol" Prinzipiell gibt es drei verschiedene Zeitintervalle:

- **Transmitting Interval TI:** Zeit, die jeder Knoten maximal zum Versenden seiner Daten brauchen darf. Für alle Knoten gleich.
- **Synchronisation Gap SG:** Zeitdauer, nach jeder Sende-Runde, in der Knoten den "Warteraum" betreten dürfen und kein Knoten senden darf. Wird nach letzter Aktivität am Bus von allen Knoten gewartet, bevor die nächste Sende-Runde beginnt und jeder seinen TG-Timer wieder startet. Für alle Knoten gleich.
- **Terminal Gap TG:** Zeitdauer für die der Bus nach dem SG ruhig sein muss, damit der Knoten zu senden beginnt. Der entsprechende Timer wird bei vorzeitiger Bus-Aktivität wieder neu gestartet. Für jeden Knoten unterschiedlich.

... wobei folgende Größenrelationen gelten:

$$TI > SG > \max_i TG_i \qquad \forall \text{ Knoten } i$$

Anwendung: Flugzeug-Industrie: Boing 777

Abbildung 4: Das Arinc629 Protokoll.

6 Offener/geschlossener Regelkreis

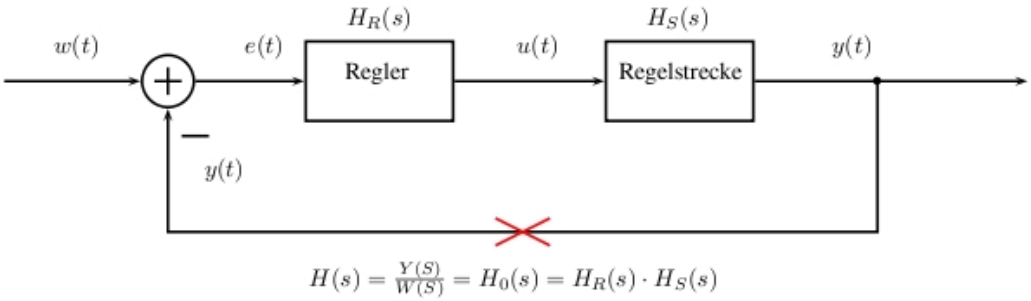

Abbildung 5: Der offene Regelkreis und die Definition von $H_0(s)$.

Den offenen Regelkreis erhält man durch "kappen" der Rückführung. Die Übertragungsfunktion $H_0(s)$ des offenen Regelkreises spielt bei Stabilitätsuntersuchungen des Regelkreises eine wichtige Rolle.

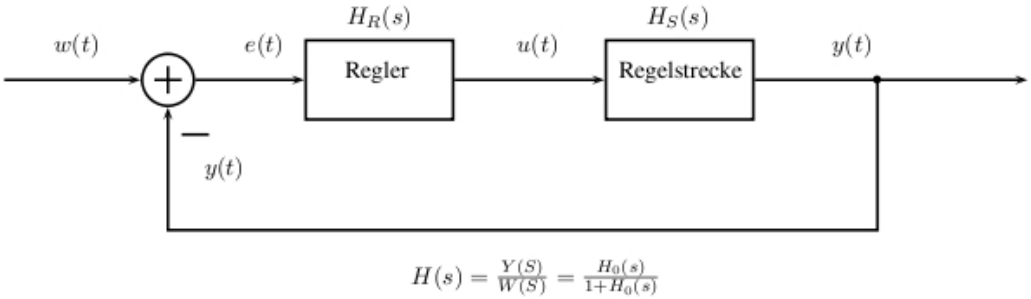

Abbildung 6: Der geschlossene Regelkreis und die Definition der Übertragungsfunktiondes Regelkreises $H(s)$.

7 Interne/externe Clocksynchronisation (Central Master Algorithm)

interne Clocksynchronisation Jeder Knoten hat eine eigene Uhr(Oszillator). Durch Versenden von Uhrzeiten werden die einzelnen Uhren aufeinander abgeglichen. Ziel ist es, dass alle Uhren in einem Cluster bis auf eine vorgegebene Präzision Π synchron laufen. Der Jitter bei der Übertragung spielt hierbei eine wesentliche Rolle für die maximal erreichbare Präzision. Es werden zwei Verfahren unterschieden:

- **Central Master Algorithm** Die einzelnen lokalen Uhren der Knoten werden auf die Uhrzeit eines Master-Knotens synchronisiert. Dieser Master-Knoten verschickt periodisch seine Uhrzeit. Nachteil: nicht fehlertolerant(Ausfall des Master-Knotens)
- **Distributed Synchronisation Algorithm** Alle Knoten versenden ihre Uhrzeit an alle anderen Knoten. Jeder Knoten berechnet aus den erhaltenen Werten mittels einer Konvergenzfunktion die neue Uhrzeit.

externe Clocksynchronisaton Es gibt ein sogenanntes "time-gateway" zu einem "time-server"(z.B. GPS). Von dort wird regelmäßig die aktuelle Uhrzeit bezogen und die lokalen Uhren werden daran angeglichen.

8 Arten von Messfehlern

Systematic Error Treten bei jeder Messung auf und heben sich daher bei mehreren Messungen im Mittel nicht auf. Beispiele sind Verstärkungsfehler, Offset-Fehler oder Fehler durch Hystereseförmige Übertragungsfunktion des Sensors. Diese Art von Fehlern können mittels Kalibrierung beseitigt werden.

Conditional Error Sporadisch zufällig auftretende Fehler z.B. durch elektromagnetische Interferenz durch andere elektronische Geräte oder elektrostatische Entladungen.

Stochastic Error Überlagerung von dauerhaft vorhandenen stochastischen Störgrößen wie z.B. Rauschsignale.

9 Sensor Fusion

Bezeichnet das Messen einer physikalischen Größe über mehrere Sensoren. Man unterscheidet:

- **Competitive Fusion:** Mehrere Sensoren messen die gleiche physikalische Größe um fehlertoleranter zu sein(Voting) oder um die Messintervalle zu verkleinern(set-up time eines Sensors).
- **Complementary Fusion:** Mehrere Sensoren messen Verschiedenes.
- **Cooperative Fusion:** Mehrere Sensoren messen dasselbe aus verschiedenen "Blickwinkeln", woraus dann die gesuchte Größe ermittelt werden kann. z.B.: Triangulation

10 WCET

WCET ... Worst Case Execution Time
Maximal mögliche Dauer für das Ausführen eines Tasks/Algorithmuses. In einem Echtzeitsystem ist das Wissen der WCET aller Tasks notwendig, um garantieren zu könne, dass bei einem bestimmten Schedule alle Deadlines eingehalten werden können. Die WCET hängt nicht nur vom Source-Code ab, sondern auch sehr stark von der Hardware, auf der der Algorithmus ausgeführt wird. Um die WCET abschätzen zu können, wird meistens "non-blocking" und "non-preemption" vorausgesetzt. Das Messen der WCET durch Ausführen des Tasks mit verschiedenen Eingaben ist nur bedingt möglich(es müssten alle möglichen Kombinationen von Eingangsgrößen getestet werden) und wird immer schwieriger, je komplexer der Task wird. Insbesondere wenn der Task innere Zustände besitzt, wird diese Methode unbrauchbar. Sie liefert jedoch eine gute erste Einschätzung. Die Alternative ist "static analysis", das Analysieren des gesamten Codes und aller möglicher Ausführungspfade. Dabei müssen die Ausführungszeiten aller einzelnen Code-Blöcke abgeschätzt werden. Je komplexer die Hardware(Pipelining, Caching), desto schwieriger wird diese Abschätzung bzw. desto weniger wird die Worst-Case Abschätzung mit dem Average-Case übereinstimmen. Bereits beim Programmieren kann darauf geachtet werden die WCET-Berechnung möglichst einfach zu halten, indem man Optimierungen, welche die durchschnittliche Ausführungszeit senken bleiben lässt und den Ausführungspfad möglichst unabhängig von den Eingabewerten macht.

11 Synchrone Programmiersprachen (SCADE - Granularität, Block Diagramme, (Safe) State Machines)

Synchrone Programmiersprachen basieren grundsätzlich auf der synchronen Hypothese, die aus folgenden Annahmen besteht:

- **multi-form time:** Die Zeit wird durch Eingaben definiert, d.h. jedes mal wenn eine bestimmte Eingabe(Takt) geschieht, schreitet die Zeit um einen definierten Wert voran.
- **zero-delay model of circuits:** Der Programmablauf kann in einzelne Zyklen zerlegt werden. Neue Programmeingaben kommen immer nur am Zyklenbeginn und die dazu berechneten neuen Programmausgaben stehen noch im selben Zyklus zur Verfügung(zero-delay). Dazu darf die längste Berechnung nicht länger dauern als ein Zyklus.

- **perfectly synchronous model:** Alle Berechnungen in einem Zyklus laufen synchron ab.

SCADE ist ein Softwaretool zum designen von synchronen Systemen. Als Eingabemethoden stehen die Eingabe als Blockdiagramm und die Eingabe als Zustandsautomat zur Verfügung. Beide können auch kombiniert werden bzw. ist hierarchisches Design möglich(ein Block besteht wieder aus einem komplexeren Scade-Model). Das eingegebene Modell kann mittels eines Model-Checker formal verifiziert(d.h auf Widerspruchsfreiheit geprüft) werden. Weiters stellt SCADE einen Simulator und einen Code- und einen Dokumentations-Generator, zur Verfügung.

12 Requirements Analyse

Die Requirements sind die Anforderungen die der Kunde/Nutzer des zu erzeugenden Systems an dieses System stellt. D.h. was soll das System können? Vorgabe vom Kunden - vertraglich festsetzen! Man unterscheidet

- **functional Requirements:** Welche Funktionen soll das System bieten.
- **non-functional Requirements:** Welche Bedingungen soll das System erfüllen, z.B. Der Sourcecode soll zertifiziert sein

13 Unterschied TTP/A : TTP/C

TTPA - zentrale Uhrensynchronisation(Master gibt Clock vor) TTPC - verteilte Uhrensynchronisation TTP/A hat (wie schon erwähnt wurde) eine zentrale uhrensync, waehrend die von TTP/C verteilt und fehlertolerant ist. fehlererkennung im TTP/A basiert auf parity, im TTP/C auf 16/24 bit CRC. das membership service ist im TTP/A simpel gehalten, ausserdem unterstuetzt TTP/A keine duplex nodes, - channels, kein redundancy management und hat keine shadow nodes (warm standby)

14 Wie funktioniert TTP/A?

Timer Triggered, Master-Slave Principle, Master gibt Clock vor(Fireworks-Byte) Kommunikation läuft in vordefinierten Runden ab: RODL-File(jeder Knoten) Fireworks-Byte gibt folgende Runde an. Im ROSE-File(nur Master Knoten) ist die Abfolge der verschiedenen Runden definiert. Man unterscheidet grob die drei Runden: multi-partner round, master-slave round(bestehend aus Master Slave Adress Round(MSA), Master Slave Data Round(MSD)), broadcast round

Eine Runde besteht aus bis zu 16 - Zeitslots. In einem Zeitslot kann ein Knoten einen Wert aus dem IFS auf den Bus schreiben, alle anderen können diesen Wert ins IFS einlesen oder einen Task ausführen. Dies ist im RODL-File definiert. Die einzelnen RODL-Files müssen also konsistent sein. IFS ... Interface File System
Darunter liegendes Bus-Protokoll: UART

15 Welche UML-Modelle gibt es?

UML ... Unified Modeling Language

- **functional model:** Beschreibt die Funktionalität des Systems von außerhalb(Anwendersicht) betrachtet. z.B.: Use Case Diagram
- **object model:** Beschreibt den strukturelen Aufbau des Systems. z.B.: Class Diagram
- **dynamic model:** Beschreibt das interne Verhalten des Systems. z.B.: Sequence Diagram, Activity Diagram, State Machine Diagram

16 Smart Transducer

Smart Transducer, zu Deutsch: intelligenter Sensor, bezeichnet einen Sensor, dem gleich ein Controller mit Kommunikationsschnittstelle beigefügt ist. D.h. die Signalaufbereitung und Codierung für die entsprechende Kommunikationsschnittstelle passiert direkt beim Sensor und der gemessene Wert wird dann über diese Schnittstelle anderen (übergeordneten) Knoten im verteilten System zur Verfügung gestellt. Prinzipiell unterscheidet man zwischen 4 verschiedenen Schnittstellen eines Knoten in einem verteilten Echtzeit-System. Diese sind in Abbildung 7 dargestellt. Diese 4 vier Schnittstellen müssen aber nicht zwangsmäßig 4 physikalisch verschiedene Schnittstellen sein.

- **Real Time Service Interface:** Hier wird der Echtzeitservice, also der codierte Sensorwert, zur Verfügung gestellt. Diese Schnittstelle muss den entsprechenden Echtzeitanforderungen(bez. Timing) genügen.
- **Diagnostic and Management Interface:** Bietet die Möglichkeit Diagnosen durchzuführen oder die Funktion des Systems zu überwachen(Fehlersuche).
- **Configuration and Planning Interface:** Hierüber wird der Controller configuriert. z.B.: Programmieren des Microcontrollers, Updates und Patches

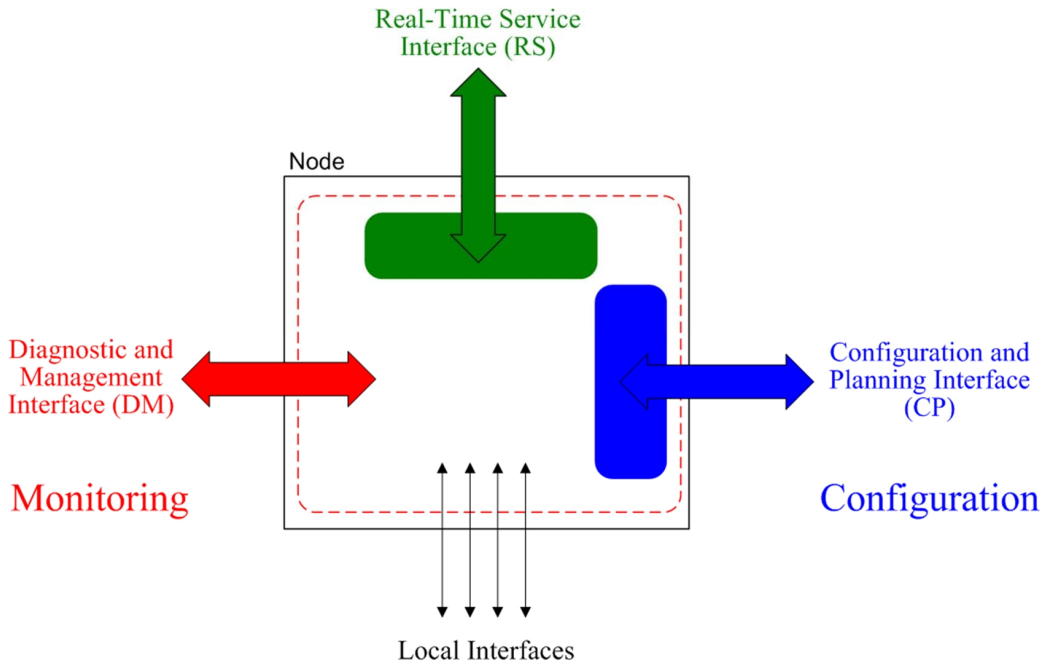

Abbildung 7: Die verschiedenen Schnittstellen eines Smart Transducers(intelligenter Sensor).

- **Local Interfaces:** Hiermit sind die “dummen Sensoren” mit dem Controller verbunden.

Die Motivation für Smart Transducers besteht darin, dass der System-Designer sich nicht mehr mit Sensor-spezifischen Problemstellungen auseinandersetzen muss, sondern den ST als Black Box betrachten kann, der, sind alle Voraussetzungen erfüllt, den gewünschten Service liefert. Überdies wurden in den letzten Jahrzenten Microcontroller immer kleiner und günstiger.

17 Welche Arten von Rauschen gibt es?

Das Leistungsdichtespektrum des grauen Rauschens orientiert sich am menschlichen Gehör. D.h. gibt man graues Rauschen über einen Lautsprecher wieder, so empfindet der Mensch jede Frequenz als gleich laut enthalten.

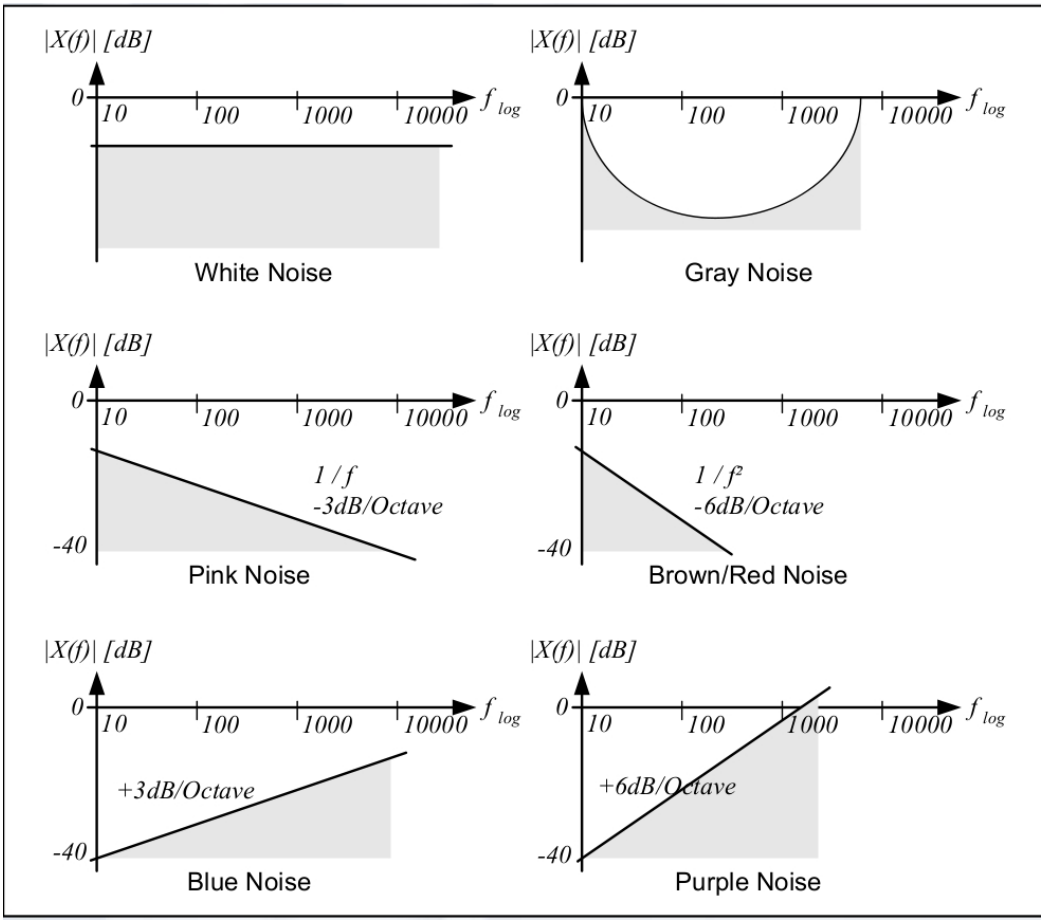

Abbildung 8: Die verschiedenen Arten von Rauschen mit ihrem jeweiligen Leistungsdiche-Spektrum.

18 CAN (Arbitrierung)

Event-Triggered, Master-Slave Bus-Protokoll(Bus-Topologie). Wird in der Auto-Industrie angewendet.
MAC-Verfahren: CSMA/CA; Leitungscodierung: NRZ mit Bit-Stuffing
Collision Avoidance durch Busarbitrierung mittels rezessivem und dominantem Buszustand. Jede Nachricht beginnt mit einem Objektidentifier, der Sender und Art der Nachricht identifiziert. Zusätzlich bestimmt der Identifier die Priorität der Nachricht. Je weiter hinten im Identifier das erste rezessive Bit vorkommt, desto höher ist die Priorität. Beginnen zwei Teilnehmer gleichzeitig zu senden, so bemerkt dies der Knoten, bei dem das erste rezessive Bit früher vorkommt und bricht daraufhin seinen Sendeversuch ab. Die Bitdauer muss daher mindestens so lang wie 2 mal die größte Signallaufzeit sein, damit jeder Knoten noch während

des aktuellen Bits erkennen kann, ob der von ihm gesetzte Wert wirklich am Bus liegt.
Gibt es mehrere Master, so werden diese in einem Token-Ring Netz betrieben.

19 Wozu Distributed Systems?

DS ermöglichen das Einführen von Redundanzen und erhöhen somit die Fehlertoleranz und Ausfallsicherheit eines Systems.
Parallelität - mehrere Rechner können gleichzeitig arbeiten.
Lokalität - Funktion wird dort berechnet, wo sie gebraucht wird.
Sehr komplexe Problemstellungen können partitioniert und die einzelnen einfacheren Teilprobleme von separaten Rechnern gelöst werden. Die separaten Rechner werden dann nach dem Composeability-Prinzip auf höherer Abstraktionsebene wieder miteinander vernetzt und bilden wieder ein System, welches das komplexe Problem löst.

20 Wasserfallmodell

Ein lineares(nicht iteratives) Vorgehensmodell in der Softwareentwicklung. Das

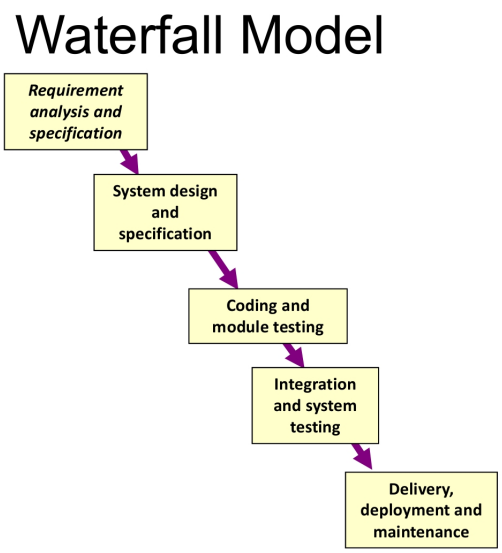

Abbildung 9: Das Wasserfallmodell im Softwareentwicklungsprozess.

Wasserfallmodell sieht eine scharfe Trennung zwischen Spezifikations- und Design-Phase vor und berücksichtigt keine möglichen Iterationsschritte. Dieses Modell stammt aus den 1970er-Jahren und ist mittlerweile überholt.

21 White-Box-Testing

Im Gegensatz zum Black-Box-Testing werden die Testdaten beim White-Box-Testing genau auf den inneren Programmaufbau der Software angepasst. D.h. es wird z.B. versucht die Daten so zu wählen, dass jedes Stück Code im Programm mindestens einmal ausgeführt wird(Code-Coverage) und dass jeder Entscheidungs-Zweig mindestens einmal durchlaufen wird(Branch-Coverage). Bei der Branch-Coverage unterscheidet man noch zusätzlich zwischen Decision-Coverage(DC) und Modified-Condition-Decision-Coverage(MCDC).

- **DC:** Jede Entscheidung die im Programmfluss zu einer Verzweigung führt soll mindestens je einmal zugunsten jedes Zweiges getroffen werden.
- **MCDC:** Jede Entscheidung die im Programmfluss zu einer Verzweigung führt soll zugunsten jedes Zweiges auf alle möglichen Arten getroffen werden.

White-Box-Testing kann parallel zur Entwicklung durchgeführt werden.

22 Was sind Bit-Rate bzw. Baud-Rate?

Die BAUD-Rate ist ein Synonym für Symbolrate. Die Symbolrate ist proportional zur Bitrate. Je nach Leitunscodierung kann durch ein Symbol eine gewisse definierte Anzahl von Bit dargestellt werden. Beispiele:

- **QPSK** es gibt 4 verschiedene Symbole $\Rightarrow$ ein Symbol beschreibt 2 Bit $\Rightarrow$ $Bitrate = 2 \cdot Symbolrate$
- **NRZ** es gibt 2 verschieden Symbole(LOW und HIGH) $\Rightarrow$ Jedes Symbol entspricht einem Bit $\Rightarrow$ $Bitrate = Symbolrate$

23 Interrupts/Polling

Prinzipiell unterscheidet man 3 Arten:

- **Sampling:** Die zu messende RT-Entity(z.B. Sensor-Wert) wird periodisch abgefragt. Handelt es sich bei der zu messenden Größe um eine digitale Größe, so kann es passieren, dass ein Ereignis, falls es kürzer dauert als die Sampling-Period lang ist, übersehen wird(siehe Abb. 10). Abhilfe kann hierbei durch Verwendung eines Speicherelements direkt am Sensor geschafft werden. Jedoch kann es selbst mit so einem Speicherelement noch immer passieren, dass ein Ereignis übersehen wird(siehe Abb. 11).

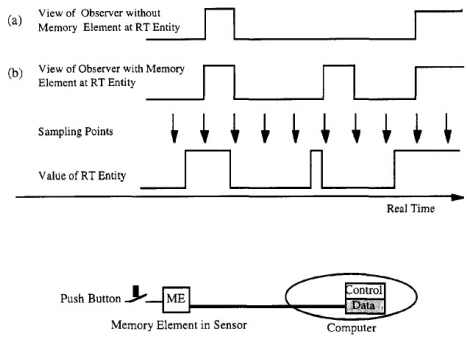

Abbildung 10: Sampling: Das Speicherelement sitzt direkt am Sensor.(Bild aus [HK])

- **Polling:** Das Polling entspricht im Wesentlich dem Sampling mit Speicherlement, mit dem einzigen Unterschied, dass das Speicherelement direkt bei Microcontroller sitzt und somit in dessen Einflussbereich liegt. Ein Nachteil dabei ist, dass auch alle Störungen, die auf der Leitung zum Sensor eingefangen werden, im Speicherelement sind(falls sie groß genug sind um einen digitalen Signalwechsel zu verursachen).(siehe Abb. 11)

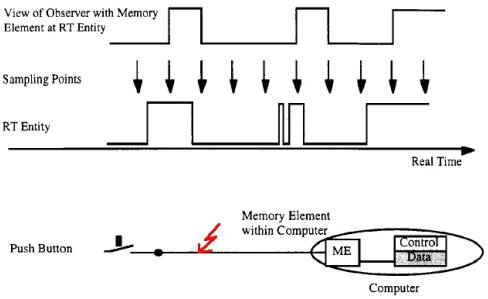

Abbildung 11: Polling: Das Speicherelement sitzt direkt am Controller.(Bild aus [HK])

- **Interrupts:** Interrupts machen nur Sinn, wenn wir die zu beobachtende Größe als digital betrachten(z.B.: $x \geqslant x_{ref} \rightarrow HIGH$; $x < x_{ref} \rightarrow LOW$). Erfüllt die beobachtete Größe eine bestimmte Bedingung wird im Microcontroller ein Interrupt ausgeführt, d.h. der laufende Prozess wird unterbrochen und eine Interrupt-Service-Routine wird ausgeführt, in welcher der ausgelöste Interrupt behandelt wird. Dadurch erreicht man eine

schnellstmögliche Reaktion auf ein Ereignis. Der große Nachteil dabei ist, dass dadurch die Analyse der WCET der verschiedenen Prozesse und Code-Stücke unheimlich erschwert wird, da ja ein auftretender Interrupt jederzeit die Laufzeit eines Code-Stückes erhöhen kann. Zusätzlich kann es passieren, dass durch einen Fehler z.B. in der Leitung, der Interrupt ständig getriggert wird. In Embedded Real Time Systems sollte man daher Interrupts nur dann verwenden, wenn die Anforderungen an die Reaktionszeit durch Sampling nicht erreicht werden können.

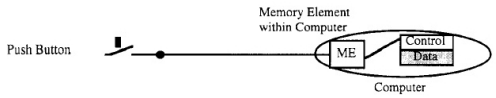

Abbildung 12: Interrupts: Signaländerungen greifen direkt in den Kontrollfluss des Microcontrollers ein.(Bild aus [HK])

24 Safety/Security

Safety bezeichnet die Sicherheit eines Systems im dem Sinne, dass bei einem Ausfall oder im Fehlerfall Niemand verletzt wird.
Security bezeichnet die Sicherheit eines Systems gegenüber böswilligen Angriffen von Außen. In Embedded Systems können Security-Risiken schnell auch zu Safety-Risiken werden. Dem Begriff Security schreibt man folgende primäre Attribute zu:

- **Confidentiality:** übersetzt bedeutet der Begriff: Diskretion, Geheimhaltung, Vertraulichkeit; Also wer darf was wissen/verändern
- **Integrity:** Integrität bedeutet Unversehrtheit, Intaktheit: d.h. Daten wurden nicht unauthorisiert verändert.
- **Availability:** Verfügbarkeit, Betriebsbereitschaft: Für authorisierte Anwender stehen die entsprechenden Services zur Verfügung.

Weiters bezeichnet man folgende Begriffe als sekundäre Attribute:

- **Accountability:** übersetzt bedeutet der Begriff: Haftung, Verantwortung; D.h. Personen die Änderungen an dem System vornehmen können ausgeforscht und haftbar gemacht werden(z.B. über User-Account-Informationen).
- **Authenticity:** übersetzt: Authentizität, Echtheit, Glaubwürdigkeit; Betrifft Echtheit einer Nachricht(Inhalt und Absender wurden am Weg nicht verändert).

- **Non-repudiability:** übersetzt: Nichtabstreitbarkeit; d.h.: Ein Autor einer Nachricht kann nicht abstreiten, dass die Nachricht von ihm ist bzw. ein User der eine Veränderung am System vorgenommen hat, kann nicht abstreiten, dass er es war.

25 Klassifikation von Embedded System Attacks

Man unterscheidet einerseits nach funktionalen Kriterien:

- **Integrity-Attacks:** z.B.: Abfangen einer Nachricht - Ändern des Inhaltes . weitersenden der Nachricht.
- **Privacy-Attacks:** z.B.: Lesen von privaten Daten, auf die man keine Zugriffsrechte hat.
- **Availability-Attacks:** z.B.: Einen Server lahm legen durch eine DDOS-Attacke.

und andererseits nach den Kriterien wie ein System angegriffen wird(von welcher "Richtung"):

- **Physical-Attacks:** Spezialfall für Embedded Systems, da man das Gerät zur Verfügung hat(Handy, Bankomat-Karte). z.B.: Abhören, Backward-Engineering
- **Side-Channel-Attacks:** Angriff über einen Kanal, den man als Entwickler des Systems nicht bedacht hat. z.B.: Stromverbrauch analysieren
- **Software-Attacks:** Trojaner, Virus

26 Eigenschaften eines RTOS

Garantierte maximale Ausführungszeit von System-Calls. Garantierte maximale Antwortzeit des OS auf externe Ereignisse. Garantierte maximale Ausführungszeiten von OS-Funktionen(ISRs, context switch, Driver). Deterministisch.

27 Verification/Validation

- **Verification:** "are we building the system right?", D.h. Überprüfen, ob das entworfene System der Spezifikation entspricht.
- **Validation:** "are we building the right system?", D.h. Überprüfen, ob das, der Spezifikation entsprechende System, dem System entspricht, das wir eigentlich bauen wollten.

28 Einzeichnen der Wendetangente in einer Sprungantwort (Bestimmung der Größen Ks, Tu, Tg)

29 Was ist beim Programmieren von Embedded Systems zu beachten?

30 SPI Blockdiagramm aufzeichnen (Leitungen)

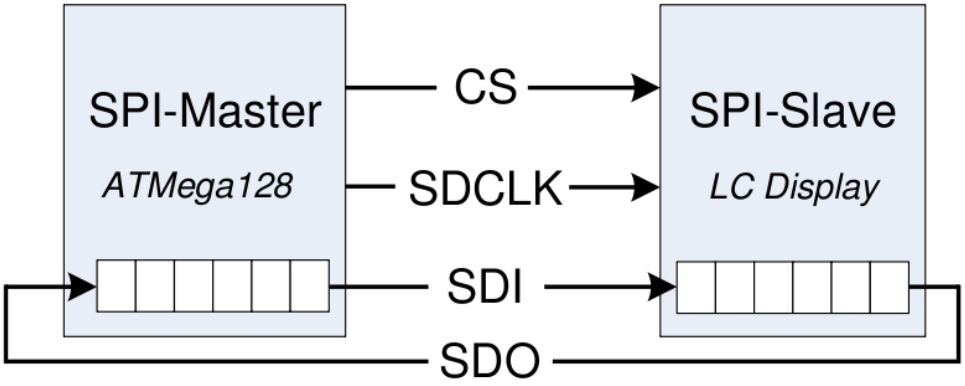

Abbildung 13: Blockdiagramm zum SPI-Protokoll.

- **CS / SS** . . . Chip Select / Slave Select
- **SDCLK / SCK** . . . Serial Data Clock
- **SDI / MOSI** . . . Serial Data In / Master Out Slave In
- **SDO / MISO** . . . Serial Data Out / Master In Slave Out

31 PCB Design Designflow

PCB ... Printed Circuit Board

32 einiges zu Modulation

Amplitude, Frequenz, Phase, Code

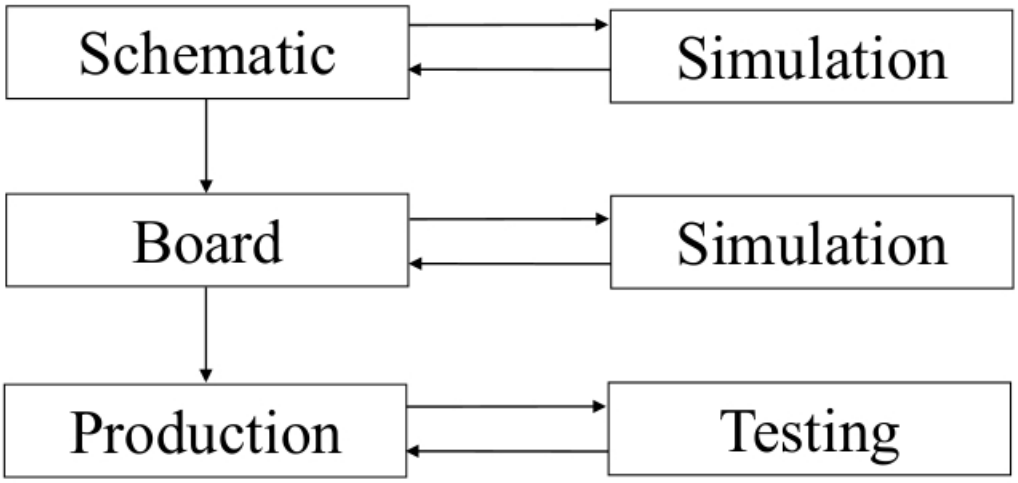

Abbildung 14: Grober Designflow eines PCB.

33 Anforderungen an Embedded Systems

Real-Time Kommunikationssystem. Clock-Synchronisation.

34 RMS Scheduling

Rate Monotonic Scheduling ist ein Static-Priority-Scheduling Algorithmus. Das bedeutet, dass jeder Task statisch eine Priorität zugeschrieben bekommt. Jeder Task i besitzt eine WCET C_i und eine Periode T_i mit der der Task ausgeführt werden muss. Je kleiner die Periode T_i desto größer ist die Priorität des Tasks. Die Bedingung dafür, dass es einen Schedule gibt, in dem alle Deadlines erfüllt werden(d.h. der Schedule ist feasible) lautet:

$$U \leq n \cdot \left(2^{\frac{1}{n}} - 1\right) \qquad \text{wobei } n \ldots \text{ Anzahl der Tasks}$$

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

35 Model based design

36 regelkreis aufzeichnen und erklären

37 Berechnung der precision PI des Central Master Algorithmus

Beim Central-Master Algorithmus synchronisieren sich alle Knoten auf die Uhrzeit eines Master-Knotens auf. Dabei spielt einerseits der Jitter ε in der Kommunikation eine Rolle(Maximale Laufzeit einer Nachricht minus Minimaler Laufzeit einer Nachricht) und andererseits der Drift-Offset Γ, der Angibt um wieviel die lokale Uhr in einer Synchronisationsperiode von der Central-Master-Uhr abtrifftet.

$$
\begin{aligned}
\Pi_{Central} &= \varepsilon + \Gamma \qquad \text{wobei } \varepsilon \ldots \text{ Jitter, } \Gamma \ldots \text{ Drift-Offset} \\
\Gamma &= 2 \cdot \rho \cdot R_{int} \qquad \text{wobei } \rho \ldots \text{ Driftrate, } R_{int} \ldots \text{ Synchronisationsintervall} \\
\rho &= \left| \frac{T_{osc} - T_{osc\_master}}{T_{osc\_master}} \right| \qquad \text{wobei } T_{osc}, T_{osc\_master} \ldots \text{ Periode lokaler/Master- Uhr}
\end{aligned}
$$

38 History Dependent Error (hysteresis)

39 digitalen regelkreis aufzeichnen

40 Welche möglichkeiten gibt es, ein Board zu schützen

41 Funktionale/Nichtfunktionale Requirenments von RTS

42 Use Case Diagramm + Beispiel dafür

43 Vor- und Nachteile der ”Controltheory”

44 Synchrone Programmiersprachen: Grundkonzept von SCADE, wie wird Zeit bzw Gleichzeitigkeit dargestellt, wer löst in SCADE die Gleichzeitigkeit auf?

(Antw: Compiler)

45 Central Master Clocksynchronization, wodurch wird die Präzission bestimmt.

Durch den Jitter ε und den Drift-Offset Γ. Wird eine Nachricht von einem Knoten zu einem anderen Knoten gesendet, so kann es unterschiedlich lange dauer, bis die Nachricht beim Empfangsknoten ankommt. Der Unterschied zwischen längster und kürzester Dauer eines Nachrichtenversandts wird als Jitter bezeichnet.

46 Was ist beim Erstellen von Embedded Systems Programmen zu beachten

47 Dual-Kernel Approach

48 wirtschaftliche Aspekte bei Security

(AW: ROI und CTB, und ein wenig erklaeren)

49 welche Anforderungen stellt ein Embedded System an die verwendete Programmiersprache

50 SCM (dezentrale, zentrale mit Beispielen - Problem von zentralen SCM usw.)

zentrales Source Code Management: Concurrent Versions System(CVS), dessen Nachfolger Subversion (SVN) dezentrales Source Code Management: git

51 praktische Beispiele

- Manchester-Codierung einer Bitfolge
- NRZ-Codierung einer Bitfolge
- Danach bekam er 4 Timings für ARINC 629 Minislotting und musste abschätzen, welche Timings zu einem gültigen Minislottingprotokoll führen (SG, TG, TI waren gegeben)
- ein Rechenbsp zu Central Master Clocksynchronization: Gegeben war Jitter 15[us] (eps), Driftrate $10^{-}4[\frac{s}{s}]$ (roh) und Resync-Intervall 200[ms] (Rint) (Ich durfte die Rechnung wegen Rechenfehler wiederholen)
- Minislotting Beispiel. Als Angabe verschiedene Timeout Kombinationen mit der frage welche gültig sind.
- TDMA (mehrere Nodes, in richtigen Slot einzeichnen was er machen soll)

Literatur

[HK] Titel: Real-Time Systems; Design Principles for Distributed Embedded Applications; Autor: Hermann Kopetz; 2002 Kluwer Academie Publisher

[Wiki] Wikipedia - die freie Enzyklopädie; URL: http://de.wikipedia.org