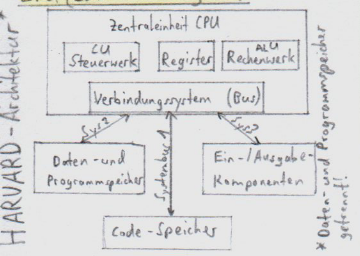
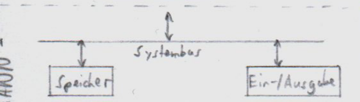
Zusammenfassung EmbSys 2

CPU (Central Processing Unit)





Systembus
dient als Verbindungssystem zwischen Komponenten µC-Systems
↳ Master-Slave-Prinzip (CPU) (Speicher und Peripherie)
↳ Adressierung Teilnehmer und Speicherbereiche, Datenübertragung, Steuerung Datenübertragung und Richtung, Signalisierung Ereignisse
↳ Interrupt-Konzept
- Erweiterung Master-Slave-Prinzip
  ↳ periphere Slave-Einheiten können bei CPU Interrupt-Requests (IRQ) anmelden
- CPU entscheidet über Bearbeitung IRQ und über Zeitpunkt Bearbeitung

Teilbusse im Systembus

Bus	Bestimmt...
Adressbus	... Grösse des adressierbaren Bereichs
Datenbus	... Datengrösse, welche transferiert werden kann
Steuerbus	... Koordination

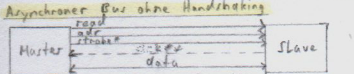
⚠ µC normalerweise Master von Systembus und koordiniert geregelten Ablauf ⚠

⚠ Während DMA (Direct Memory Access) übernimmt zwischenzeitlich ein anderer Bus-Teilnehmer die Funktion des Bus-Masters ⚠

Funktionsweisen
- Synchroner Systembus
  Synchronisation Busteilnehmer über Bustakt
- Asynchroner Systembus
  Synchronisation der Busteilnehmer über andere Steuersignale

zusätzl. Charakterisierungen:
- ohne Handshaking
  zeitlicher Ablauf ist vorausbestimmt und immer gleich
- mit Handshaking
  der zeitliche Ablauf wird über Quittierungssignale gesteuert
- Wartezyklen (wait states)
  erlauben korrektes zeitliches Abstimmen auf langsamsten Busteilnehmer in Datentransfer

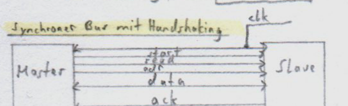
Asynchroner Bus ohne Handshaking



* strobe erklärt Adressen als gültig

Asynchroner Bus mit Handshaking
** ack → acknowledge kommt noch hinzu!
↳ quittiert gegenüber Master strobe-Signal
↳ erlaubt schnellere Übertragung, da keine fixen Zeiten abzuwarten sind
↳ aber Achtung, auch Wartezeiten

Synchroner Bus mit Handshaking



↳ Master legt Signale zwischen den steigenden Flanken auf den Bus

Schieberegisterspeicher
↳ Umwandlung von seriell zu parallel und umgekehrt
↳ Peripherieschnittstellen
- A/D-Wandler
- serielle Schnittstelle (RS232, SPI...)
- LAN (z.B. Ethernet)
- Peripheriebusse (z.B. USB)

↳ Links-/rechts-schieben

FIFO
↳ Ein-/Ausgabepuffer
- Tastatur, Drucker, serielle Schnittstelle, Festplattenkontroller, LAN-Schnittstelle
- Ringbuffer

Stapelspeicher LIFO
↳ für Stack

Hardware-Aufbau Speichersystem
- System Modul
  erzeugt Bus-Takt, Reset-Signal und allfällige weitere Signale
- Prozessor
  bestimmt Organisation des lokalen Busses
- Speicher- und Peripheriebausteine
- Adressdekoder und Handshake-Logik
  ↳ erzeugt Select-Signale für die Speicherbausteine entsprechend gewählter Adresse
  ↳ handhabt zeitlich richtige Abfolge Handshake-Signale

Adressdecodierung
Beispiel:

Bank-Nr.	Chip-Nr.	Interne Adr. auf Chip
28	27 26	22 21 ... 0

a) Wie gross Speicherkapazität jedes Chips: 2^{22} = 4MB
b) betriebene Speicherchips pro Bank: $2^5 = 32$
c) Anzahl mögl. Speicherbänke: $2^2 = 4$
d) max. Speicherkapazität einer Bank 2^{27} = 128MB
e) max. gesamte Speicherkap.: $2^{29} = 512$MB
f) Welchen Adr.-Bereich belegt Speicherbank 2?
0x1000'0000 ... 0x17FF'FFFF

memory mapped IO
↳ memory und Daten im gleichen Adressraum

Bus-Interface
↳ bildet Schnittstelle zum Prozessor hin

Hauptspeicher (Central Memory)
↳ dient der Ablage und Bereitstellung von Programm- und Dateninformationen

Little Endian (Intel)
↳ Least Significant ⚠Byte⚠ liegt an der adressierten Stelle, die weiteren folgen

n+3	n+2	n+1	n
...	76543210	76543210	76543210

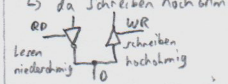
Big Endian (Motorola)
↳ Most significant ⚠Byte⚠ liegt an der adressierten Stelle, die weiteren folgen

n	n+1	n+2	n+3
76543210	...	76543210	76543210

Bits innerhalb Byte bleiben jedoch gleich!

Direkt adressierter Speicher
↳ angesprochene Speicherstellen sind oft 1Byte gross, aber auch mehrere Bytes möglich
↳ zwei Hauptgruppen
- RAM
- ROM

Prüfungsfrage:
Weshalb wird bei einem Bit-organisierten Speicher bei gleichzeitigem Lesen (OE oder RD) und Schreiben (WR) aktiv, schreiben priorisiert?
↳ da Schreiben hochohmig bedeutet.



Mehrport Speicher
↳ Schreib-/Lese-Speicher mit mehreren Zugriffspfaden (Bus-Ports)
↳ benötigt Bus-Arbitrierung
↳ je nach Anwendung uni-/bidirektional
- Dualport-RAM (DPRAM)
  zur Kopplung von mehreren Prozessoren oder Peripherien
- Dualport-Speichersystem
  mittels Steuerlogik erweitertes Speichersystem
- Video-RAM (VRAM)
  unidirektionaler Dualportspeicher im Grafiksystem; Speicherung von Bildschirm-Inh.
- Multiport-Bauelemente
  mehr als zwei Zugriffspfade
- Datenkonsistenz sicherstellen
  HW-Semaphoren, betätigt durch SW

Speichertypen
- Zentraler Speicher
  - sind direkt am Bussystem des µC angeschlossen
- Peripherie Speicher
  - werden über Ein-/Ausgabe-Schnittstellen an µC angeschlossen
- Halbleiter-Speicher Familien
  - Mask ROM MROM
  - Programmable ROM PROM
  - EPROM
  - EEPROM
  - Flash
  - Static RAM SRAM
  - Dynamic RAM DRAM
  - Ferroelectric RAM
  - Magnetoresistive RAM MRAM

Assoziativ-Speicher (CAM)
CAM: Content Addressable Memory
↳ inhaltadressierter Speicher mit einer Teil-Information eines Eintrages kann der gesamte Informationseintrag gefunden werden.
↳ Für Zugriff ist keine Adresse notwendig, hingegen kann beim Speichern von Informationen Auswahl der Speicherstelle über Adressen geschehen.
↳ Anwendungen:
- Software-Lösung bei Datenbankabfragen
- als Teil einer Cache-Hardware (Associative Cache)
- als Teil der Adresstransformation (Translation Lookaside Buffer, TLB)

↳ Nachteile:
- Mehrfachtreffer sind möglich (wg. Maske)
- Problematische Verwaltung von freien Speicherstellen
- aufwendige Hardware bei Direkt-Zugriffs-CAM

Muster: 00101101
Maske: 00100011

Valid Bits	CAM-Inhalt	Trefferbits
1	11010010	0
1	11110001	1
0	11101101	1
1	01010011	0
1	10100010	0
1	01111001	1
1	00010100	0
0	11011110	0
1	00000110	0
1	11010101	0

Resultate: 11110001, 01111001

CAM → ist wie eine Datenbank

Valid-Bit:
Blendet die Zeilen einfach ein oder aus

Muster:
Dies sind die Bits, nach denen gesucht wird.

Maske:
Mit der wird ausgewählt, welche der Musterbits überhaupt angeschaut wurden.

Resultate:
Wenn Valid gesetzt und alle von Maske ausgewählte Bits übereinstimmen ist Resultat

CAM Aufgabe
Ein CAM-Speicher enthalte 8 Worte zu je 16 Bit. Beim Zugriff mit dem Muster 0x10C4 ergibt sich der Treffer 0x3824.
↳ Gesucht ist die verwendete Maske mit der maximalen Anzahl binärer Nullen (0).

⚠ Muster: 0001 0000 1100 0100
Treffer: 0011 1000 0011 0100
Maske: xx0x 0xxx 0000 xxxx
↳ max. Anzahl Einer (1).
Maske: 1101 0111 0000 1111

⚠ Bei Umwandlung Hex zu Binär immer sehr sorgfältig und gut kontrollieren!

Fehlersicherung im Speicher
- Fehlerklassen
  - Fehler beim Schreiben (Write Soft Error)
  - Momentaner Fehler beim Lesen, bei Wiederholung kein Fehler mehr (Read Soft Error)
  - Defekter Speicherbaustein, liefert konstant falsche Information (Hard Error)
  - Fehler bei der Verarbeitung in der CPU
- Hamming-Codierung
  - min. 2 Fehler erkennbar
  - 1 Fehler korrigierbar

Anzahl Datenbits	Anzahl Prüfbits (Redundanz)
16	6
32	7
48	8
64	8

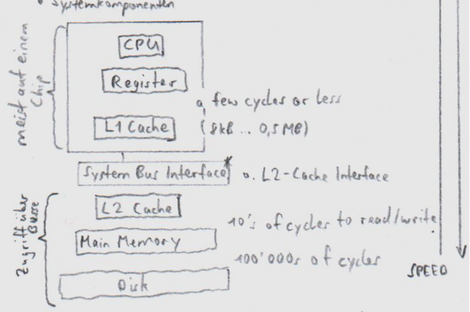
FRAM
core → ferroelectric crystals, the central atom moves in the direction of the field
→ non-volatile and fast

1MByte/s = 10^6 Byte/s
⚠ nicht 2^{20} Byte/s ⚠

Betriebssystem Kern hat Kernfunktionen wie...
- Prozessverwaltung
- Prozesssynchronisation
- Interprozess-Kommunik.
- Exception- und Interrupt-Handling
→ oft als Mikrokern (micro kernel) realisiert

Michael Schwenter FS16

Speicher-Hierarchie
- Anforderungen an Systemarchitektur
  - hierarchische Organisation der Bussysteme und Systemspeicher (Speicher-Hierarchie)
  - Caches
  - Adresstransformationen
  - Multiprozessor-Strukturen
  - Systemkomponenten



* Interface zum Hauptspeicher und zu externen Systemkomponenten
  - kommuniziert auch mit L2 und In- und Outputs
  - Busbreite häufig das doppelte von Prozessor-Wortbreite

Speicherbandbreite
↳ Anzahl Bit/Bytes, die in einer Zeiteinheit transferiert werden können

$$S_p = L_p \cdot SB_p$$

S_p Speicherbandbreite [byte/s]
L_p Leistung eines Prozessors P in MIPS (Millionen Instruktionen pro Sekunde)
SB_p Speicherbedarf (in Byte) pro Durchschnittsbefehl des Prozessors P

Info: durch intensive Nutzung von Load/Store-Befehlen, wie dies bei RISC-Architekturen typisch ist, erhöht sich der SB_p deutlich

Prozessor-Leistung

$$L_p = \frac{1}{CPI_p \cdot T_p}$$

L_p Leistung in Instruktion pro s [MIPS]
CPI_p Zahl der Taktzyklen pro Durchschnittsbefehl des Prozessors P (Cycles per Instruction)
T_p Taktzyklus des Prozessors P in µs

Info: CPI_p kann statisch oder abhängig von best. Programmen ermittelt werden.

Betriebssysteme und RTOS

Betriebssystem (Operating System)
Stellt das Bindeglied zwischen der Hardware eines Computers einerseits und dem Anwender bzw. seinen Programmen andrerseits dar.
Es umfasst Programme, die zusammen mit den Eigenschaften des Computers die Grundlage der möglichen Betriebsarten dieses Systems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.

Virtuelle Maschine → Formen und Ausprägungen
- Hardware + Laufzeitumgebung des Sprachsystems
  - geeignet für Kleinsysteme mit einfacher Peripherie.
  - bereitgestellt werden Dienste zur Programmausführung, eventuell angereichert für Parallelverarbeitung durch Laufzeitbibliothek (z.B. C++ oder Java).
- Hardware + Betriebssystem
  - abgestuft im Funktionsumfang für Systeme jeder Grössenordnung geeignet
  - Bereitgestellt werden Ein-/Ausgabedienste und Mittel zur Parallelverarbeitung (z.B. Linux, Windows), einfache Betriebssysteme für µC's)

Betriebssystem im Schichtenmodell
↳ bietet Anwendung API-Schnittstellen zur realen Maschine
↳ macht Anwendung damit weitgehend unabhängig von realer Maschine (CPU, Hardware, Peripherien)

		Beschreibung	Beschreibungs-/
	Anwendung	Methoden, HLL	Hochsprachen
OS-System	Virtuelle Maschine	Betriebssystem-Dienste	⇒ API: Application programming interface
		Betriebssystem-Kern	⇒ SPI system progr. interface
	Reale Maschine	Logische Organisation	CPU/CU/ALU Register, Memory,
		Physische Organisation	I/O/Peripherien, Clock

konkrete Teilaufgaben (Kernfunktionen)
- parallele Prozesse
- Interprozess-Kommunikation
- Peripherie- und Gerätetreiber
- Speichermanagement
- Hardware Abstraktion
- Ressourcenverwaltung
- Dateisysteme (↳ Auftragsverwaltung)
- Utility Funktionen (Benutzer/Betriebsmittelverwaltung)
- Dateiverw.

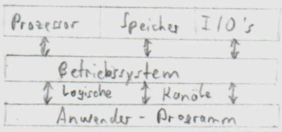
!!Parallelität!!
↳ Prozessverwaltung

Zusammenfassung EmbSys 2

Anwenderprogramm ohne/mit Betriebssystem

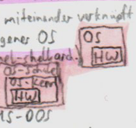
ohne:
- Anwenderprogramm greift direkt über physikalische Kanäle auf Reale Maschine zu
- Anwenderprogramm ist eng mit realer Maschine gekoppelt

mit:
- OS bildet klare Abstraktionsschicht für Zugriff auf reale Maschine
- entkoppelt reale Maschine von Anwenderprogramm
- bietet Anwender-Programm über Logische Kanäle zusätzl. Funktionen, Möglichkeiten und Diensten an



OS Architekturen

- Monolitische Architektur (monolitic architecture)
  - ↳ OS und HW sehr eng miteinander verknüpft ⇒ jede HW braucht eigenes OS
- Kern-Schale-Archit. (kernel-shell arch.)
  - ↳ Free RTOS
- Hierarchische Schichten (hierarchical Layers) ⇒ MS-DOS
  - ↳ HW ↔ BIOS ↔ DOS ↔ Command.com
- Mikrokern (micro-kernel)
  - ↳ Client-Server Modell (Windows)
- Virtuelle Maschine (virtual machine)
  - ↳ siehe S. 1 Grafik unten rechts



OS Betriebsarten (operation mode)

- Stapelverarbeitung (batch processing)
  - Bearbeitung Folge von Stapelaufträgen
  - Jobs werden immer ohne Interaktion von User abgearbeitet
- Dialogbetrieb (interactive processing)
  - ständiger Wechsel zwischen Aktionen des Users und des Systems
  - User kann Arbeitsablauf im Dialog jederzeit beeinflussen
- Echtzeitverarbeitung (real time processing)
  - Einsatz eines Computersystems zur Steuerung und Überwachung von techn. Prozessen
  - Einhaltung von Zeitbedingungen muss gesichert sein
- Verteilte Verarbeitung (distributed pr.)
  - System besteht aus mehreren gekoppelten PC's
  - OS dient vorrangig Verteilung von Daten, Ressourcen, Arbeitslast

Entwurfskriterien OS

- Modularität und Orthogonalität
- inkrementelle Erweiterbarkeit und Konsistenz
- Skalierbarkeit
- Zuverlässigkeit und Fehlertoleranz
- Portierbarkeit
- Transparenz und Virtualisierung

Multitasking Task = Prozess

- nebenläufige, gleichzeitige Prozesse (concurrent processes)
  - ↳ werden benutzt um quasi-parallele Ausführ. auf einem Einprozessor-System zu erreichen

Regeln für erfolgreiches Multitasking

- von allen Prozessen gemeinsam benötigte Rechenzeit muss ≤ sein als die vom Prozessor verfügbare Rechenzeit
- Prozesse müssen zwischenzeitl. auf Ereign. warten können
- Aktives Warten (busy waiting) nicht erlaubt!

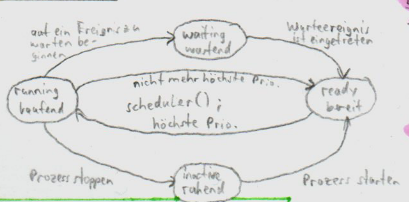
Definition Prozess

Ein (sequentieller) Prozess (task, process) ist eine dynamische Folge von Aktionen, die durch Ausführung eines Programms auf einem Prozessor zustande kommt.
Ein Prozess ist insbesondere durch seinen zeitlichen unveränderlichen Zustand gekennzeichnet. Er wird im OS infolge eines Auftrags erzeugt.

Dämonen (demon processes)

↳ dauernd im Hintergrund existierende OS-Prozesse

Prozesszustände



FreeRTOS verwendet dies ⇒ *

Prozess-Beschreibung

Kontext (contex)

- zur Verwaltung → jeder Prozess ein Kontext
- ↳ Speicher-Kontext
  Speicherabbild des Prozesses (Programm-Code, Stack, Daten)

  Hardware-Kontext
  Registerabbild (Programcounter, Stack-pointer, Registerinhalte usw.)

  System-Kontext TCB
  Betriebssysteminterne Verwaltungsdaten zum Prozess (auch als TCB)

Adressraum: PC → Task-Code; SP → Task-Daten; Flags → TCB

Task Control Block TCB

- Datenstruktur, in welcher Kontext des Prozesses abgelegt werden kann
- Informationen in einem TCB
  - Prozess-Identifikation (Name, Nummer)
  - Prozesszustand
  - Priorität
  - belegte und angeforderte Betriebsmittel
  - Verwandtschaftsbeziehungen

Prozesswechsel (Context Switch) *

1. Sicherung gesamten Kontextes des unterbrochenen Prozesses ①
2. Änderung Zustand in "Ready" oder "Waiting" je nach Grund Unterbrechung ①
3. Auswahl des zu aktivierenden Prozesses ② durch Scheduler
4. Änderung des Zustandes des Prozesses ② von "Ready" in "Running"
5. Wiederherstellung des (gesicherten) Kontextes dieses Prozesses ②. Durch Laden der Prozessorregister wird dieser Prozess fortgesetzt.

* → bedeutet "Ressource Prozessor" neu zuteilen

Idle Task (Dummy-Prozess, "Leerprozess")

- ↳ besteht meist nur aus einer Warteschleife und kann jederzeit verdrängt werden
- ↳ nur aktiv, wenn bei Scheduling kein anderer Prozess laufbereit!
- ↳ ohne Idle Task müsste innerhalb OS-Kern gewartet werden ("busy waiting")

Kernsperren

↳ verhindert ungewollte Unterbrechungen während den Abläufen innerhalb OS-Kern

Scheduling- / Abfertigungsstrategien

- Hohe CPU Auslastung (ideal: 100%, normal 40-90%)
- Hoher Durchsatz (Zahl der Tasks pro Zeiteinheit)
- Faire Behandlung Tasks (im Mittel gleicher CPU-Zeitanteil für jeden Benutzer)
- Ausführungszeit: Zeitspanne von Jobbeginn bis Jobende
- Kurze Wartezeit: Der Scheduler kann nur Wartezeiten in Waiting- und Ready-Listen beeinflussen
- Kurze Antwortzeit (von Jobstart bis erste Reaktion an Bediener erfolgen kann)

Unterschiedliche Verfahren

- Kooperative Zuteilung (cooperative scheduling) *
  Jeder Prozess läuft so lange, bis er CPU selber freigibt.
  - ↳ minimaler Verwaltungsaufwand
  - ↳ funktioniert nur so lange, bis ein Prozess CPU NICHT mehr frei gibt ⇒ deadlock
  - ↳ Reaktionszeiten direkt von Prozesslaufzeiten abhängig
- Zeitscheibenverfahren (round-robin scheduling) * (optional Time slicing)
  Betriebssystem bildet gleichmässige Zeitschlitze (Quantum) mit Hilfe eines HW-Timers (Ticks). Nach Ablauf einer Zeitscheibe (Tick) wird alter Prozess wieder zuhinterst eingereiht in Ready-List.
  - ↳ FIFO
  - ↳ Alle Prozesse gleich wichtig wenn keine Prios!
  - ↳ bescheidener Verwaltungsaufwand
- Priorisierte Zuteilung (mit Verdrängung) *
  (priority based, preemptive scheduling)
  Jedem Prozess wird von Anwender eine Prio zugewiesen. Prozess mit höchster Prio erhält CPU zugeteilt.
  (Wird ein Prozess mit höherer Prio lauffähig, wird sofort ein Re-Scheduling veranlasst und der laufende Prozess verdrängt.
  ↳ anspruchsvolle Echtzeitsysteme)
- First Come First Served (FCFS)
  - ↳ non-preemtive
- Shortest Job First (SJF), Shortest Processing Time (SPT)
  - ↳ non-preemtive
- Feste Prios (Fixed External Priorities, FEP)
- Dynamische Prios
- Terminabhängige Zuteilung
  - ↳ dynamische Prio. Ermittlung anhand von Zeitkriterien
  - → Earliest Deadline First (EDF)

Listen:

- Ready-List
  ↳ enthält die lauffähigen Prozesse
  ↳ mehrere Listen können vorhanden sein → z.B. eine pro Prio. Stufe
- Waiting-Listen

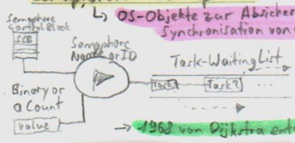
Prozesswechsel:
- aktiver Prozess gibt Prozessor entweder freiwillig ab oder er bekommt ihn entzogen
- Auswahl welcher Prozess Prozessor als nächster zugeteilt wird erfolgt durch Scheduler nach vordef. Scheduling-Strategie
- Prozesswechsel wird von Dispatcher gesteuert

Koordinierung paralleler Prozesse

- Synchronisation
- Interprozess-Kommunikation (Interprocess Communication, IPC)
- Wechselseitiger Ausschluss (Mutual Exclusion)
- Kritischer Bereich (Critical Section/Region)

Semaphoren — Semaphores

↳ OS-Objekte zur Absicherung und Synchronisation von Prozessen



→ 1968 von Dijkstra entwickelt

P-Operation (proberen, prüfen)
V-Operation (verhogen, erhöhen)

P — zähler > 0

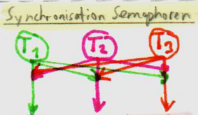
then	else
zähler = zähler − 1 - 1 Marke vergeben - Prozess darf weiterarbeiten	Prozess aus dem Zustand laufend entfernen und an Semaphore-Warteschlange anfügen. schedule (CPU neu zuteilen)

V — semaphore-Warteschlange leer

then	else
zähler = zähler + 1 - 1 Marke zurücknehmen	ältesten Prozess aus Sema.-Warteschlange entfernen und in Bereit-Liste einfügen schedule (CPU neu zuteilen)

Semaphore Typen
- Binary Semaphores
  - ↳ only values 0 or 1
- Counting Semaphores
  - ↳ value 0 ... max
- Mutex Semaphore
  - ↳ special Binary Semaphore that supports...
    - Ownership: cannot be unlocked by another task
    - Recursive Locking: counts number of p and v-operations by the task
    - Task Deletion Safety
    - Prioritätenvererbung (priority inheritance)
      - ↳ dient zur Vermeidung Prio.-Umkehr (priority inversion)
      - ↳ besitzt ein Task niedriger Prio. (T1) eine Semaphore kann es Problem geben, dass Task hoher Prio (T2) nicht weiterlaufen kann, da gleiche Semaphore belegen möchte
    - Lösung: Prio von T1 wird temporär auf Prio T2 erhöht ↳ Prio-vererbung!

Synchronisation Semaphoren

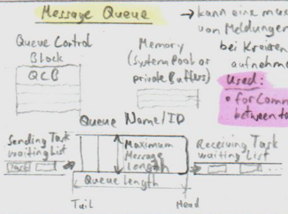


Counting-Semaphoren für Synchronisation 3er Tasks

Message Queue

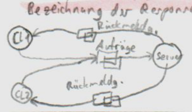
→ kann eine maximale Anzahl von Meldungen (bestimmt bei Kreieren der Queue) aufnehmen

Used:
- for Communication between tasks / ISR



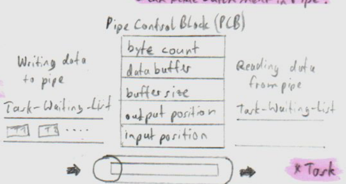
Client / Server Konzept

- Client beansprucht Dienstleistung des Servers
- Server erbringt die geforderte Dienstleistung
  - ↳ die Aufträge (Messages) müssen die Absenderadresse des zugehörigen Clients enthalten, bzw. die Bezeichnung der Response-Message Queue.



Pipes (Pipelines)

- Pipes können dynamisch created oder destroyed werden.
- verbindet zwei oder mehrere Prozesse / Tasks / ISRs
- Synchronisation ⇒ Lesender Prozess wartet automatisch, wenn keine Daten mehr in Pipe.



- Select Operation erlaubt * sich zu blockieren und auf ein spezifiziertes Ereignis zu warten an einer oder mehreren Pipes

* Task

Event Registers, Event Groups

- Event Register ist ein Bestandteil des TSB eines Tasks.
- An external source ⇒ Task/ISR, can set bits in event register to inform task, that a particular event has occured.
- Event Register size: 8-, 16-, 32-, bits or wider
- Each bit in event reg. is treated like a binary (event) flag.
  - ↳ can be set or cleared
- Typical Event Register operations:
  - get, set, clear Event Bits
  - wait for Event Bits / sync Event Bits
  - AND / OR operation on Event Bits

Event Reg. Control Block TCB
- wanted Events
- received Events
- timeout value
- notification conditions

Michael Schwenter FS16

Signals

- treten asynchron auf!!
- Signale können nach Benutzerdefinierten Aktionen behandelt werden
  - ↳ ASR = Asynchronous Signal Routine
- Signals can be ignored, made pending, processed (handled) or blocked
- Signals sind sogenannte SW-Interrupts → viel langsamer als Interrupts, da noch viel SW im Hintergrund abläuft!
- Signal Control Block (SCB)

Signal Control Block	Handler
wanted signals →	signal handler
ignored signals	signal handler
pending signals	⋮
blocked signals	

Deadlock

Ein Deadlock ist eine Situation, in welcher mehrere Threads eines Systems permanet geblockt sind, weil Ressourenanforderungen nicht erfüllt werden können.

Bedingungen für Auftreten von Deadlocks:

- Mutual Exclusion
  Auf eine Ressource kann nur von jeweils einem Task zugegriffen werden.
- No Preemtion
  Eine nicht preemptive Ressource kann nicht dem Task entzogen werden, welcher momentan auf sie zugreift. Sie ist nur verfügbar wenn der zugreifende Task sie wieder freigibt.
- Hold and Wait
  Task besitzt bereits gewisse Ressourcen, benötigt aber noch weitere um fortzufahren.
- Circular Wait
  Eine Kette von zwei oder mehr Task existiert, wenn jeder Task eine Ressource besitzt, welche vom nächsten Task der Kette angefordert wird.

⚠ Das zeitliche Auftreten von Deadlocks ist nicht vorhersagbar! ⚠
↳ Race Condition
Deadlocks sind klare Design-Fehler!!

Deadlock Prevention

- Eliminieren des Hold and Wait
  Ein Task fordert zu einem Zeitpunkt alle Ressourcen an, welche dieser benötigt und kann nur beginnen, wenn alle Ressourcen erhalten hat.
- Eliminieren des No Preemption
  Ein Task muss bereits erhaltene Ressourcen wieder freigeben, wenn benötigte Ressource nicht erhalten wurde. Task muss dann wiederum alle Ressourcen anfordern.
- Eliminieren des Circular Wait
  Alle Tasks müssen die Ressourcen in selber Reihenfolge anfordern.