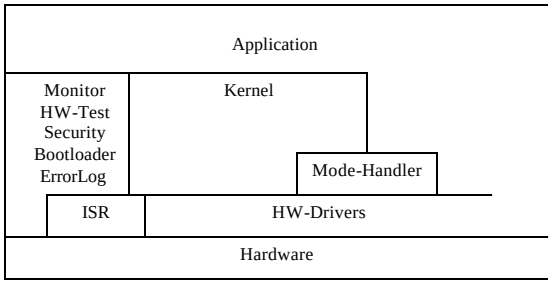
SurefireKernel

Sicherheitskernel

DATASHEET

ÜBERSICHT

SurefireKernel ist ein schlanker skalierbarer nicht preemptiver Echtzeit-Kernel der für den Einsatz auf Kontrollersysteme optimiert ist. Er verfügt über eine Realtime-Überwachung die periodische Task auf ihre korrekte Abarbeitung überprüft.



Der Kernel bildet die Schnittstelle zwischen der Applikation und der Hardware. Durch Anpassung der HW-Treiber lässt sich der in ANSI-C geschriebene Kernel einfach auf eine beliebiges Kontrollersystem portieren.

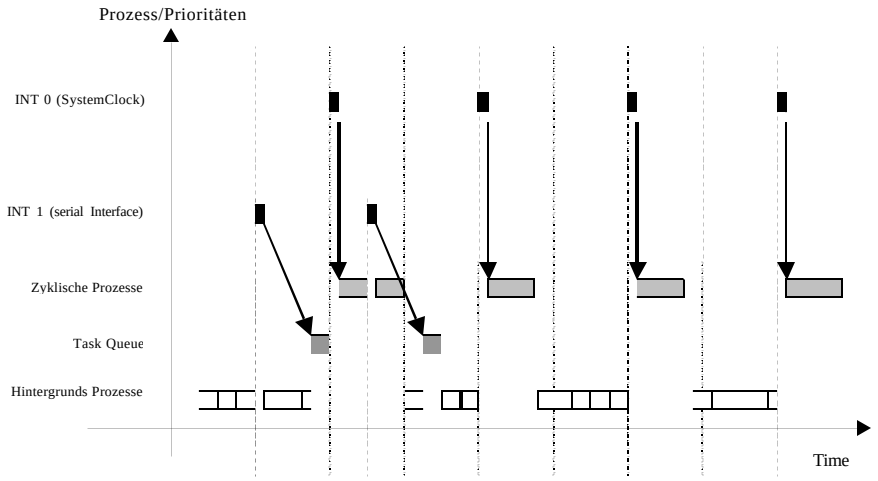
SPEZIFIKATION

Der SurefireKernel ermöglicht dem Anwender auf einfachste Weise die Handhabung hoher zeitlicher Anforderungen. Dabei werden folgende Dienste unterstützt:

1.	Zyklischer Prozess	Zyklische Task sind in einer Liste eingetragen und werden der Reihe nach ausgeführt. Die Abarbeitung der zyklischen Task wird durch einen Timer-Interrupt gestartet. Die zyklischen Task werden jedoch nicht im Interrupt-Kontext ausgeführt und können daher von Interrupts unterbrochen werden.
2.	Hintergrund-Prozess	Die Idle-Zeit des Systems wird für die Abarbeitung der Hintergrund-Task verwendet. Sie können durch Interrupt und zyklische Task unterbrochen werden und die gemeinsam genutzten Resourcen müssen daher geschützt werden.
3.	Task-Queue	Task-Queue enthält Task welche nur einmal ausgeführt werden sollen. Dabei kann es sich um die Abarbeitung einer Funktion handeln, die durch ein Interrupt-Ereignis ausgelöst wurde. Die Task-Queue wird mit hoher Priorität im Hintergrunds-Kontext ausgeführt. Die Task der Task-Queue können von Interrupts oder zyklischen Prozessen unterbrochen werden und die gemeinsam genutzten Resourcen müssen geschützt werden.
4.	System-Timer	Bei dem System-Timer können beliebige Task zur einmaligen oder wiederholten Ausführung registriert werden. Wird die gewünschte Systemzeit erreicht, wird der auszuführende Task in die Task-Queue geschrieben.
5.	Interrupts	Da Interrupts alle Tasks unterbrechen können, müssen die Resourcen, welche innerhalb der Interruptbehandlung benutzt werden in den jeweiligen Task geschützt werden. Generell sollte die Interruptbehandlung möglichst schnell abgeschlossen und aufwendigere Aufgaben an Task (Task-Queue) delegiert werden.

PROZESS-PRIORITÄTEN

In einem Systemdesign ist anzustreben, dass die Latenzzeit möglichst kurz ist. Die Ausführungszeit der Interrupts und der Umfang der zu schützenden Resourcen ist daher entsprechend klein zu halten. Die kurze Kontextswitch-Zeit bleibt unabhängig der Anzahl verwendeter Task.



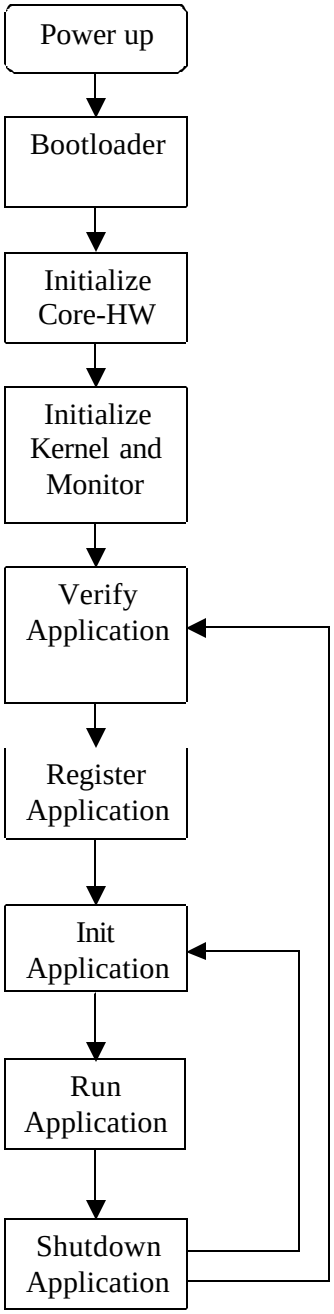
KERNEL-ERWEITERUNGEN

Der Grundkernel kann mit diversen Modulen anwendungsspezifisch erweitert werden.

1.	RAM-Test	Überprüft laufend applikationsunabhängig die Funktionstüchtigkeit des RAM's. Dabei wird eine geringfügige Erhöhung der Latenzzeit in Kauf genommen.
2.	ROM-Test	Überprüft laufend die korrekte Code-Checksumme des aktiven Programms
3.	Monitor	Ermöglicht während dem laufenden Betrieb über eine externe Schnittstelle den beliebigen Zugriff auf den Datenspeicher der Zielhardware.
4.	Security	Speichert und überprüft die Daten von Variablen mit einer Checksumme
5.	Bootloader	Ermöglicht das Laden des Kernels oder der Applikationssoftware auf die Zielhardware
6.	ErrorLog	Protokolliert flüchtig oder nichtflüchtig auftretende Fehler. Diese können mit Hilfe des Monitors ausgelesen werden.

KERNEL-BETRIEBSMODE

Der Kernel läuft als eigenständiges Programm auf der Zielhardware. Mit den notwendigen Kernelerweiterungen besteht die Möglichkeit eine Applikation im Betrieb auf das Zielsystem zu laden und danach zu aktivieren.



Nach einem Powerup wird der Kernel mit seinen Grunddiensten gestartet.

Initialisierung des Kontrollers mit der Kernhardware

Starten des Kernels mit allfälligen optionalen Erweiterungen

Überprüfen ob eine korrekt deklarierte Applikation vorhanden ist.

Registration der Applikation bei dem Kernel.

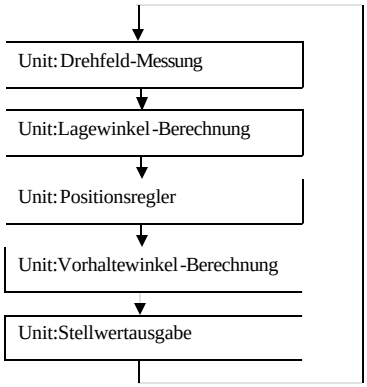
Initialisierung der Applikation

Abschalten der Applikation durch Abarbeitung der Shutdown-Tabelle.

UNITS

In dem Systemdesign wird eine Applikation in logische Funktionsblöcke (Units) gegliedert. Die Komplexität der Anwendung wird heruntergebrochen. Dabei entstehen klare Einheiten die für die Entwicklung exakt spezifiziert werden können. Diese Spezifikation beinhaltet neben der Funktionalität die Echtzeitanforderung und den Bedarf an Systemresourcen.

Zum Beispiel:



Jede Unit ist in einem eigenen C- und H-Datei codiert. Sie verfügt über eine standardisierte Schnittstelle welche folgende drei Funktionengruppen beinhaltet:

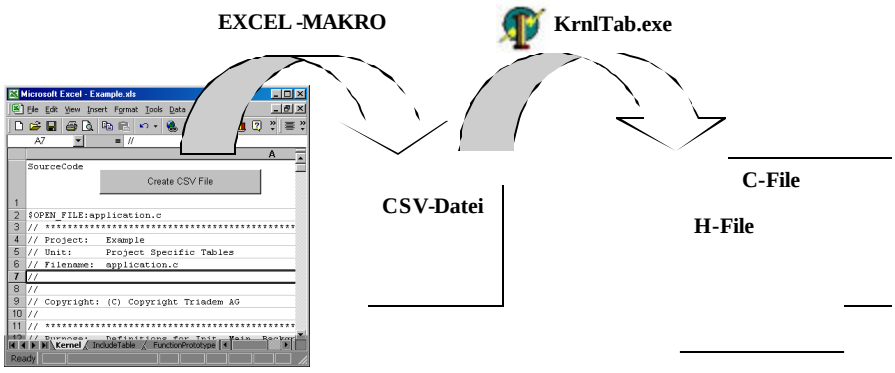
```
// ###########################################################################
// # EXPORTED FUNCTION PROTOTYPES
// ###########################################################################
// Init
tTASKID UnitXyz_Init (tDATA *);
// Execute
tTASKID UnitXyz_Execute1 (tDATA *);
tTASKID UnitXyz_Execute2 (tDATA *);
tTASKID UnitXyz_Execute3 (tDATA *);
// Shutdown
tTASKID UnitXyz_Shutdown (tDATA *);
```

Init-Funktionen werden von dem Kernel in der Initialisierungsphase aufgerufen. Sie ermöglichen die Initialisierung von Variablen und Hardwarebausteinen.
Execute-Funktionen werdem vom Kernel im Betrieb periodisch aufgerufen.
Shutdown-Funktionen verwendet der Kernel für die kontrollierte Abschaltung der Applikation.

Jede Funktion verfügt über einen tTASKID Rückgabewert. Dieser eineindeutige Wert dient zur Überprüfung der korrekten Funktionsausführung. Zudem kann jeder Funktion eine Referenz auf Unitdaten übergeben werden. Dadurch können pro Unit verschiedene Objekte gebildet werden.

ERSTELLUNG EINER APPLIKATION

Zur vereinfachten Erstellung einer Applikation existiert das Tool „KernelTab.exe“. Die zeitlichen Abläufe der Initialisierung-, Betriebs- und Abschaltvorgänge werden in einem Excel-Sheet festgelegt und via Makro in ein CSV-File gewandelt. Diese generierte Datei ist der Input für das KernelTab-Tool, welches daraus den gewünschten C-Code erzeugt.



Initialisierung

Beim Aufstarten der Applikation führt der Kernel deren Initialisierung durch. Dazu werden alle in der InitTable aufgeführte Funktionen sequential abgearbeitet. Die Funktionen verfügen über einen konstanten allenfalls eine eindeutigen statischen Rückgabewert.

```
tTASKID Funktion (tDATA *);
```

Mittels dieser Rückgabewerte berechnet der Kernel eine CRC-Checksumme und überprüft diese mit einem im Programm-Code abgelegten Wert. Dadurch wird die Abarbeitung aller Funktionen in der richtigen Reihenfolge sichergestellt.
Mittels der optionaler Referenz auf eine globale Datenstruktur kann eine Funktion zur Initialisierung verschiedener Datensätze verwendet werden. Diese Auslagerung der Daten ermöglicht die Umsetzung eines Objektorientierten Designs.

Excel-Tabelle zur Beschreibung der Applikations-Initialisierung

	A	B	C	D	E	F
1	**Index**	**Unit**	**Task**	**Data**		
2	1	UnitA	UnitA_Init	NULL		
3	2	UnitB	UnitB_Init	DataUnitB_Obj1		
4	3	UnitB	UnitB_Init	DataUnitB_Obj2		
5	4	UnitC	UnitC_Start	DataUnitC		
6	5	UnitD	UnitD_Init	NULL		
7	6	UnitE	UnitE_Init	DataUnitE_Obj1		
8	7	UnitE	UnitE_Init	DataUnitE_Obj2		
9	8	UnitE	UnitE_Init	DataUnitE_Obj3		
10						

InitTable | MainTable | BackgroundTable | ShutdownTable | Patterns

Betrieb

Zyklische Task werden vom Kernel im Betrieb in einem starren Zeitraster periodisch ausgeführt. Dabei muss vom Entwickler sichergestellt werden, dass die vorhanden Zeitfenster trotz Unterbrechung allfälliger Interrupts (zeitlich sehr kurz) zur vollständigen Abarbeitung ausreichen. Die Zeit welche bei der Verarbeitung von Zyklischen Task verbleibt wird für die Bearbeitung der Hintergrund-Task verwendet.
Während der Abarbeitung der Zyklischen- und Hintergrundstask bildet der Kernel von den Funktionsrückgabewerten CRC-Checksummen. Diese werden mit den im Programm-Code nichtflüchtig abgespeicherten Werten verglichen. D adurch kann der Kernel die sequentielle korrekte Abarbeitung aller Betriebstask gewährleisten.

Excel-Tabellen zur Beschreibung der Betriebsprozesse

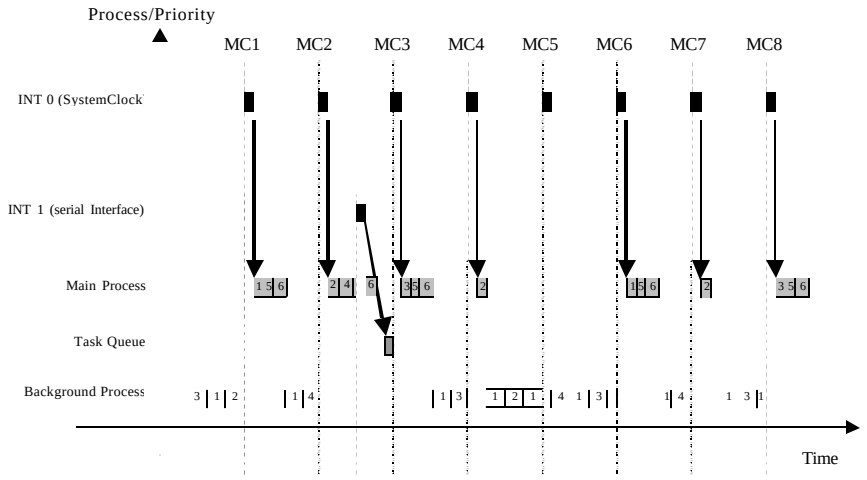
	A	B	C	D	E	F	G	H	I	J	K	L
1	**Index**	**Unit**	**Task**	**Data**	**C8**	**C7**	**C6**	**C5**	**C4**	**C3**	**C2**	**C1**
2	1	UnitA	UnitA_Execute1	NULL	0	0	1	0	0	0	0	1
3	2	UnitA	UnitA_Execute2	NULL	0	1	0	0	1	0	1	0
4	3	UnitA	UnitA_Execute3	NULL	1	0	0	0	0	1	0	0
5	4	UnitB	UnitB_Execute	DataUnitB_Obj1	0	0	0	0	0	0	1	0
6	5	UnitB	UnitB_Execute	DataUnitB_Obj2	1	0	1	0	0	1	0	1
7	6	UnitC	UnitC_Execute	DataUnitC	1	0	1	0	0	1	1	1
8												

MainTable | BackgroundTable | ShutdownTable | Patterns

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Index**	**Unit**	**Task**	**Data**	**C8**	**C7**	**C6**	**C5**	**C4**	**C3**	**C2**	**C1**	**Reload**
2	1	UnitD	UnitD_Init	NULL	1	1	1	1	1	1	1	1	1
3	2	UnitE	UnitE_Init	DataUnitE_Obj1	0	0	1	0	0	1	0	0	1
4	3	UnitE	UnitE_Init	DataUnitE_Obj2	1	0	0	1	0	0	1	0	1
5	4	UnitE	UnitE_Init	DataUnitE_Obj3	0	1	0	0	1	0	0	1	1
6													

MainTable | BackgroundTable | ShutdownTable | Patterns

Zeitverhalten



Abschalten
Eine Applikation kann vom Kernel angehalten und wieder neu gestartet werden. Dies kann zum Beispiel zur Implementierung eines Sleep-Mode verwendet werden, bei dem die Applikation ausgeschaltet und ihre Hardwarekomponenten in einem Powerdown-Mode konfiguriert werden müssen.
Analog der Initialisierung steht auch für das Shutdown eine Tabelle zu Verfügung, die der Kernel mit Überprüfung der CRC-Checksumme abarbeitet.

Excel-Tabelle zur Beschreibung der Applikations-Abschaltung

	A	B	C	D	E	F	G
1	**Index**	**Unit**	**Task**	**Data**			
2	1	UnitA	UnitA_Shutdown	NULL			
3	2	UnitB	UnitB_Shutdown	DataUnitB_Obj1			
4	3	UnitB	UnitB_Shutdown	DataUnitB_Obj2			
5	4	UnitC	UnitC_Shutdown	DataUnitC			
6	5	UnitD	UnitD_Shutdown	NULL			
7	6	UnitE	UnitE_Shutdown	DataUnitE_Obj1			
8	7	UnitE	UnitE_Shutdown	DataUnitE_Obj2			
9	8	UnitE	UnitE_Shutdown	DataUnitE_Obj3			
10							

MainTable | BackgroundTable | **ShutdownTable** | Patterns

GLOSSAR

CRC	Cyclic Redundancy Check
Hintergrunds Prozess	Die Idle-Zeit des Systems wird für die Abarbeitung der Hintergrund-Task verwendet.
Latenzzeit	Verzögerungszeit zwischen dem Eintritt des Ereignis bis zur ausgelösten Reaktion
preemptive	Dynamische Rechenzeitaufteilung durch das Betriebssystem
Queue	Warteschlange
Task	Ausführbarer Programmcode mit zugeordneter Datanstruktur und konstantem Rückgabewert
Unit	Spezifiziertes Programm-Modul
Zyklischer Prozess	Wird vom Kernel im Betrieb in einem starren Zeitraster periodisch ausgeführt.

Triadem Solutions AG
Neuengasse 38
2502 Biel
Switzerland
Tel: +41 32 327 36 36
Fax: +41 32 327 36 37
info@triadem.ch

TRIADEM