Zusammenfassung Vorlesung MPS 2 26.09.2003

Mikrocontroller-Kern auf Chip (ARM7-Core). Peripherie Komponenten sind auf Chip.

Warum?: maximale Integration bei geringen Systemkosten

Unterscheidung:

Standard: universell
Kundenorientierte: zugeschnitten auf bestimmtes Marktsegment

Aufbau: Core (on ChipCPU + onChipPeripherie) -> digital

A/D -> Einlesen
D/A -> Steuerung
WDT -> WatchDogTimer

1) ASB = Advanced System Bus = Kommunikation mit Speicher(LDR/STR)
2-Schreib und 2 Lesezyklen notwendig.

Der Timercounter hat 3 Kanäle.

Parallele kann auch Empfangsschnittstelle für Serielle sein. Da sie Doppeltbelegt sind!

Timerprogrammierung: Memorymapped

Default: Alle aus bei Reset!. Mit Takt versorgen.

Powermanagement: 0xFFFF4000
+ 10

0xFFFF4010

Peripherie Clock einschalten.

TC2-Einschalten:

9. Bit = 1
0xFFFF4010
Enablen mit: 0x100

Auf C-Ebene

Termin2:

PMC_PCER_ Clock einschalten für Absatz.

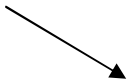
int main(void) 32 Bit Peripheral clock enable Register
{
volatile unsigned int * pm_pcer
Andere dürfen Register beschreiben. Nicht optimierbar!!! Zeiger a. unsig. int.

Int x = 1; Optimierung 1 bei arm-elf-gcc wird keine Anweisung ausgefüjrt. Wird nie benutzt. Deswegen wegoptimiert.

x=3; -> Bei Optimierung würde wegfallen. Deswegen volatile.

x=5;

pmc_pcer = (unsigned int *) 0xFFFF4010;



Adresse zuweisen
dereferenzieren:
* pmc_pcer = 0x100;
.
.
.
*(pmc_pcer +4) = 0x100; -> Disablen
}

übersetzt:

mvn r1,#48896 (0xBF00) Achtung Negation !!!
mov r3,#0xFFFF4010 <- geht nicht, da max 12 Bit!!

r3: FFFF40FF

sub r3,r3,#0xEF
mov r2,#0x100
str r2,[r3] r2 an Adresse r3 speichern -> Pointer!

Ansteuern

Über Portpins (PIO).

Portpins werden zu Ports zusammengefaßt.

- **è** über Register ansteuern.

Steuerregister.

S.20

Wichtig: Signalpfade lernen!!

1. Schritt

PIO_PSR wenn 1, dann durchschleifen. Ähnlich schalter oder Multiplexer.
PAD = z.B. LED

Statusregister = wg. Doppelbelegung abfragen.

Portbelegung PIO(B).

S.23

0	Portname	Signal	Direction	Reset	Pin	Tasten
1						
2						
3	SW1					PB3
4	SW2					PB4
5	SW3					PB5
6	SW4					PA9
7						
8						
9						
10						
11						
12						
13						
14						
15						

DS1 = PB8
DS2 = PB9
.
.
.
DS8 = PB15

Zusammenfassung Vorlesung MPS 2 10.10.2003

Memorymapped I/O.

Volatile unsigned int * pmc_pcer = (unsigned int *).

0xFFFF4010; -> Pointer auf per. Reg. Zeigen.

Dereferenzieren: * pmc:pcer = 0x4 (Hex 0x100 bei US 0)
.
.
.
wiedereinschalten mit 0xFFFF4014 !!

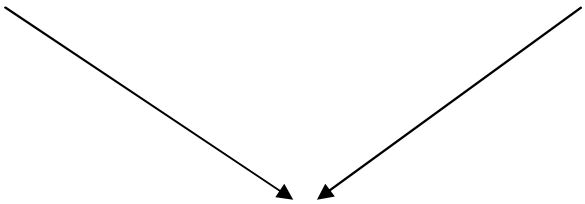
2 Typ-Definitionen in Headerfile (elegantere Lösung).

```
typedef volatile unsigned int at91reg;
typed strct
{
        at91reg PMC_SCER;
        .
        .
        .
        at91reg PMC_PCSR;

} struct PMC;
```

Basisadresse definieren

#define PMC_BASE ((Struct PMC *) 0xFFFF4000)



in PMC.h schreiben

in C-Datei

#include “PMC.h“

int main (void)
{
struct PMC * pmcbase = PMC_BASE

Zugreifen

pmcbase->PMC_PCER = 0x4; (Einschalten)
pmcbase->PMC_PCDR = 0x4;

über PAD’s werden externe Signale angesteuert. S19.
Es existieren 32 Ebenen von diesen PAD’s.
2 PIO’s = 64 Leitungen nach außen.
Portdatenregister
Richtungs(daten) bidirektional (raus = led ansteuern)
Alternativfkt. (rein = Taste eingangssignal)

immer: status, Ein, Aus. (ein/aus im Debugger nicht sichtbar, nur stat.)

PIO(B) arbeiten!

Taste1: SW1 auf Pin3 (PB3) – 3 Bit seten = 2^3 = 8 reinschreiben.
Taste2: SW2 auf Pin4 (PB4)
Taste3: SW3 auf Pin5(PB5)

Hier: Seite 23 drucken, als Referenz!

LED1: DS1 auf Pin8 (PB8)
LED8: DS8 auf Pin15(PB15).

PSR: PioStatusRegister: 1 oder 0.

Kontrolle: 1 = Pio und 0 = Alternativfkt.

Am Anfang Pio_Enable beschreiben.

Taste 1 an Pin 3
also mitteilen durch header-Datei. (bauen einer Parallel I/P Datei!!!!!)

piob_base-> PIO_PER = 0x8
..... 0 | 1000
1 ist PB3.

beide Tasten
SW1 und SW2 = 0x18 → Nur Kontrolle.
3.Bit und 4.Bit

Output Status Register

OSR: PioOutputStatusRegister (Richtung des Datenverkehrs)

Wenn 1 -> Ausgangssignal bsp. LED
Wenn 0 -> Eingangssignal z.B. Tasstenabfrage.

Taste 1 und 2 = 1 und 8 von den Bits her
Bittabelle:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Bsp:

DS1 ist PB8 und belegt die Bits 8 und 9 also schreibt man 1 und 2 = 1100 das ist 3 in Hex.
PB9 ist das Bit 2.

DS8 ist PB15.

Taste 1 und 2 haben die Bits 3 und 4 also 8 und 1 in Hex.

piob_base->PIO_ODR = 0x18 (Taste 1 und 2).
= 0x318(Taste 1 u. 2 & Led 1 und 2)
Richtung mitteilen.

Was bedeutet dies für die Datenverarbeitung?

piob_base->PIO_ODR = 0x18;
piob_base->PIO_ODR = 0x300;

Enable= Outputsignale
Disable= Inputsignale

ODSR = OutputDatenStatusRegister

ODSR (Led zum Leuchten bringen) Man kann es nur setzen (set).

ODSR: PioOutputDataStatusRegister
welches Signal liegt an 0 / 1 ?
SetOutPutDataReg -> 1
ClearOutPutDataReg-> 0

Bsp. S. 19

An PAD ist LED1 verdrahtet. Ein u. Ausschalten.

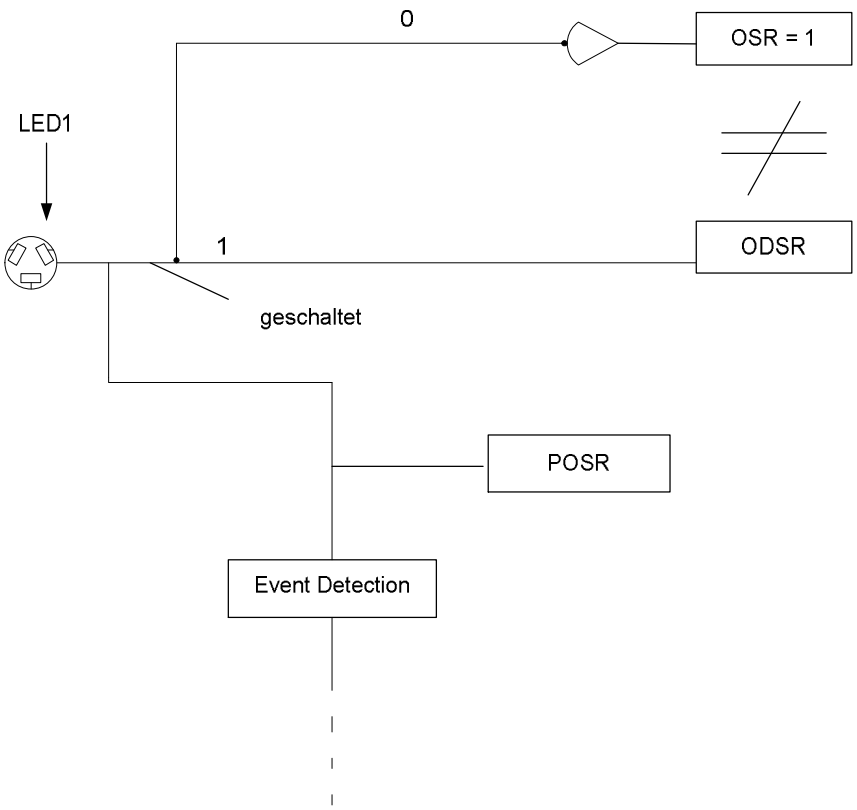
Init-Phase

*piob_per = 0x100; ⟶ INIT
*piob_oer = 0x100; ⟶

(Schleife)
*piob_codr = 0x100; ⟶ Einschalten
*piob_sodr=0x100; ⟶ Ausschalten

Negativlogik. Clear 0 dann im Status
0 = Einschalten !!!

Set = Ausschalten !!

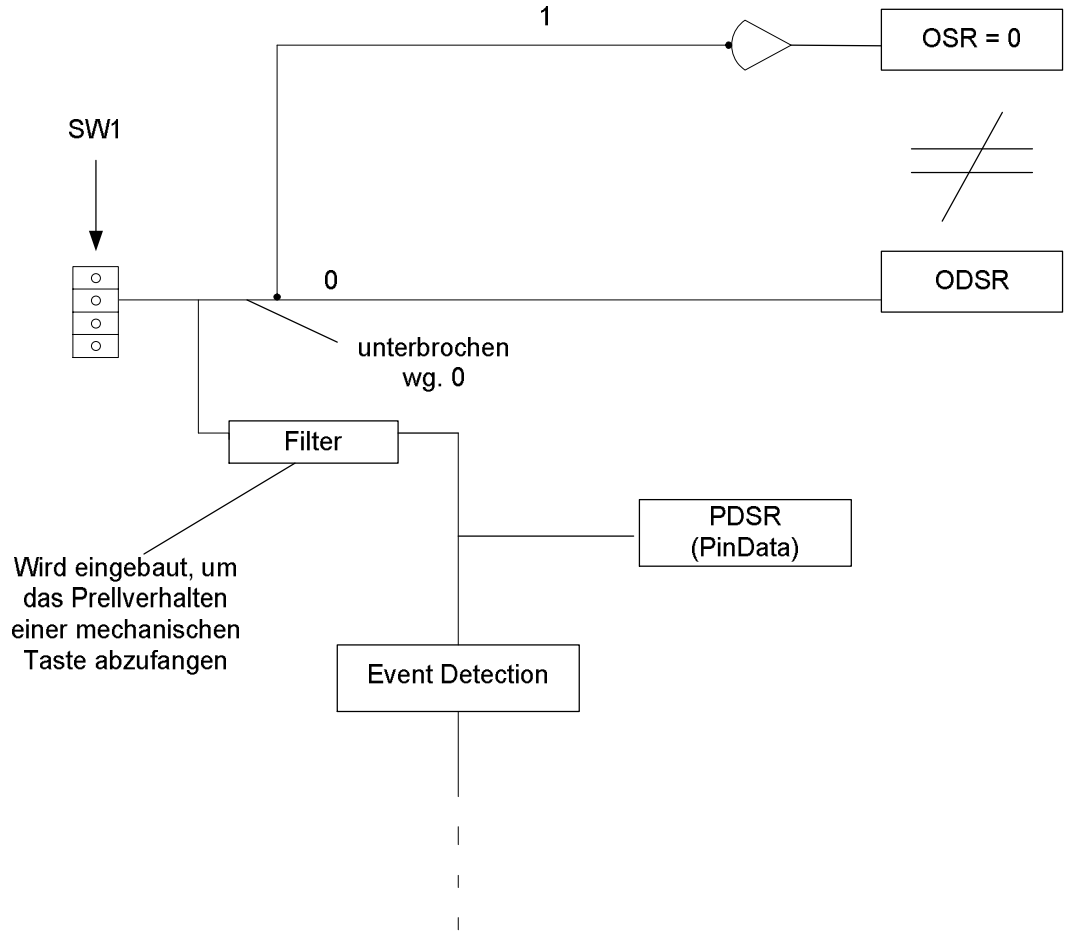


2 Bsp:

Taste 1

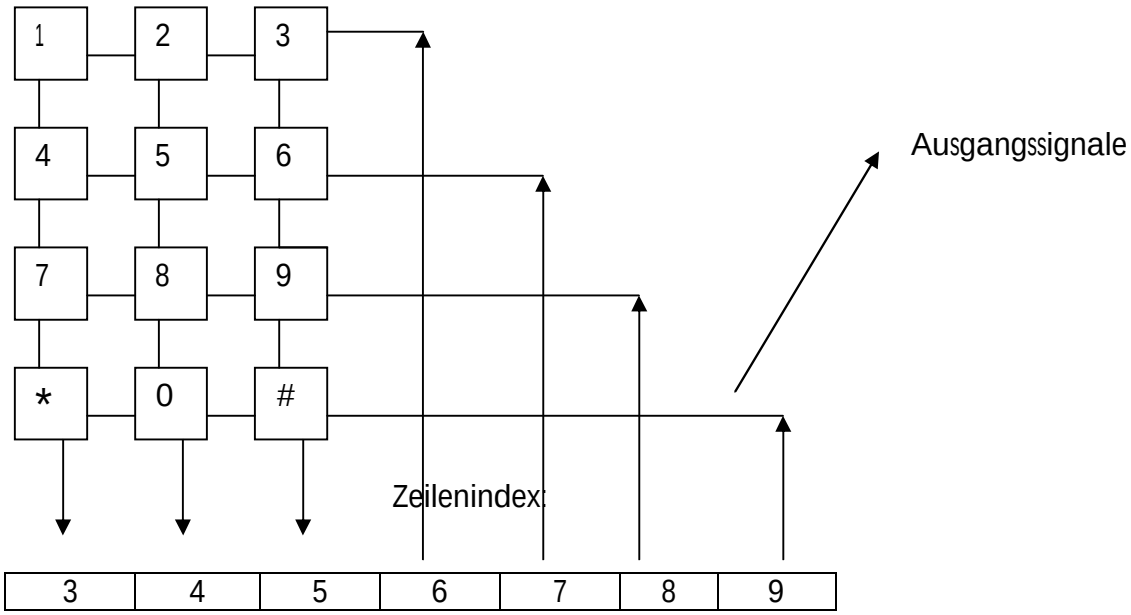
*piob_per = 0x8;
*piob_odr = 0x8; → Init !!

*piob_ifer = 0x8;



PDSR = 0x0 -> offene Taste
PDSR = 0x8 -> geschlossene Taste(gedrückt)

Beispiel an einer Tastaturmatrix



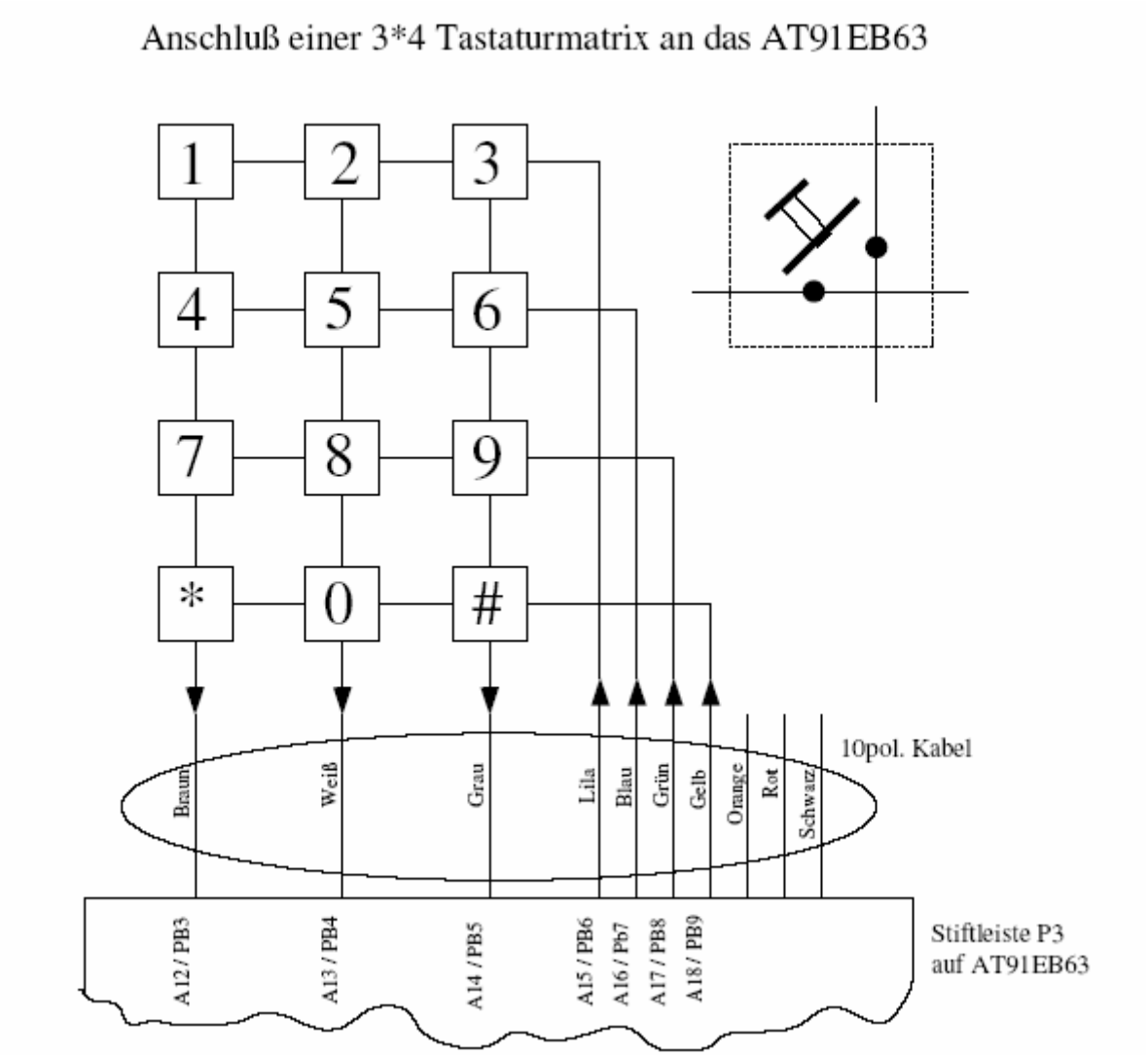

```
#define Spalten (0x38)
#define Zeilen (0x3C0)
oder
#define Spalte 1 0x8
#define Spalte 2 0x10
#define Spalte 3 0x20

Auswahl durch oder !: #define Spalten { Spalte1 | Spalte 2 | Spalte 3}
int main (void)
{
struct PIO * piobaseB = PIOB_BASE;
struct PMC * pmcbase = PMC_BASE;

pmcbase-> PMC_PCER = ...;
piobaseB-> PIO_PER = Zeile | Spalten;
pioBaseB->Pio_OER = Zeilen;
piobaseB->PIO_ODR = Spalten; (Eingangssignale)

while (Taste !='#') {
//Feststellen ob Taste gedrückt ist
//Feststellen welche Taste gedrückt wurde
}

pmcbase_>PMC_PDER = ...;
} ende;
```



In Taste:=0 -> angespeichert.

OER = Zeilen (Ausgang)
ODR = Spalten(Eingang)

PDSR = ob Taste gedrückt ?

Busy-Waiting -> wenn Prozessor besetzt!

Bits 3 – 5 = Eingang der Spalten
Bits 6 – 9 = Ausgang der Zeilen

Lowactive Signale = wenn Taste drücken dann liegt immer 0 an.
>> = Schiebeprozess nach rechts in C++.

(0 0110

0x040 = Zeile 1
0x080 = Zeile 2 1 nach links → Prüfen = Polling

Interrupt-Handling

s. 30
Peripherie-Baustein z.B. PIO dann Interrupt-Controller AIC dann CPU

AIC_IVR -> Adresse
Interrupt Vector Register

Wenn in Hex, dann 4'er Blöcke.

*AIC_EOICR = * PIOB_ISR -> Komplette Register schreiben(Beide gleichzeitig).

Zusammenfassung Vorlesung MPS 2 24.10.2003

Mechanismen zur Ereignissteuerung

Busywaiting (Prozessor ist gesperrt – Dauerbelastet mit Anfrage)
Polling (Prio gesteuert mit continue)
Interrupt (Peripherie Controller, Interrupt Controller, CPU)

Merke: Taste SW gibt 0 bei Drücken aus !! Low-Active !!
Merke: Immer 4'er Blöcke = 0. z.B. 0x4000 = 3 4'er Blöcke sind 0!
Merke: Für jede Interruptquelle eine Startadresse.

<u>AIC-Befehle</u>

Im AIC_IVR -> Sprungadresse zu der CPU verzweigt. SVR SCHREIBT IN IVR!

Ende: AIC_EOICR -> End of Interrupt Command Register. Mit irgendeinem Wert beschreiben.

Urzustand: 1. AIC_IDCR = 0x4000 -> Interrupt Disablen Command Register
i. AIC_ICCR = 0x4000 -> Zustand um Interrupt abzuschalten(empfangsbereit). Initialisierung. Interrupt Clear Command Register. Interrupte zurücksetzen.

AIC_SVR[PIOB_ID] -> Adresse der Interrupt-Service-Routine -> taste_irg_handler. Korrespondiert zu entsprechenden Interrupt.

AIC_SMR -> Source Mode Register. Für Pios 1-7. Verhalten vom Interrupt beim angegebenen Signal. z.B. Mode 0x7. 8 Prio Levels. Bits 5-6 für Sourcetype.

AIC_IECR = 0x4000 ->Enable Command Register. Erst dann Interrupt zulässig
-> Einschalten für PIO_B erst dann Interrupt annehmbar.

Ende:

AIC_IDCR = 0x4000 -> Disablen
AIC_ICCR = 0x4000
pmc-pcdr = PMC_BASE -> PIO abschalten

Interrupt Service Routine

__attribute__ -> teilt dem Compiler mit, dass eine ISR anfängt(standard)

Abschluß für ISR: aiobase->AIC_EIOCR = piobase->PIO_ISR
Was passiert: 1. EOICR wird beschrieben
PIO_ISR wird gelesen !! mit einem Befehl !!!!

Merke: ~ immer 1'er Komplement in C also Negation.

Merke: Wenn CPU im Interruptmode dann ist er im priv. Modus

Was ist zu tun ?

Retten der Statusregister passiert im SPSR. Save Program Status Register.

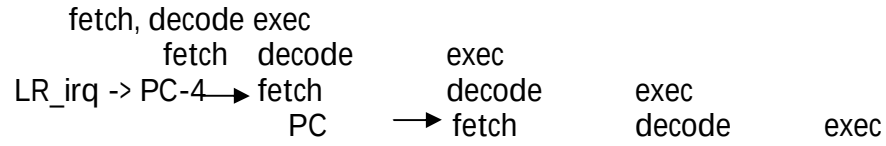
CPSR -> SPSR_irq
Bits 0 - 7

Eintriit in ISR (User-IRQ).

7 -> Interrupt Disable Bit. Wenn 1 sperrt CPU Interrupt. Prüft zuerst Bit. Default gesperrt!

Mode-Bits(4 Stück).. -> CPU sorgt dafür.

1. Gespeichert CPSR
2. Mode Bits gesetzt und Interrupt Disable Bit Nr. 7 im CPSR wird also überschrieben. Wenn Breakpoint 0x92.
3. Mapping des Register LP,SP
4. Speichern der Return-Adresse (PC-4) im LR_irq.
5. Setzen des PC auf Beginn des ISR. Info steht in AIC_IVR.



PC-4 korrigiert PC und das LR, wg. Pipelining.

---

ldmed fp, {....}^ -> das Dach leistet, das CPSR wiederhergestellt wird.

Beendingung eines Interrupt, was passiert:

* Wiederherstellen des CPSR aus SPSR_irq
* Wiederherstellen des PC an LR_irq

Prios bei S. 36 Interrupts

Timer S. 84

Timerbaustein ist getaktet über eine Uhr(Clock).

25MHz läuft ARM-CPU wird durch MCLK gesteuert in folgenden Abständen /2, /8, /32, /128, /1024 um z.B. auf eine Sekunde umzurechnen.

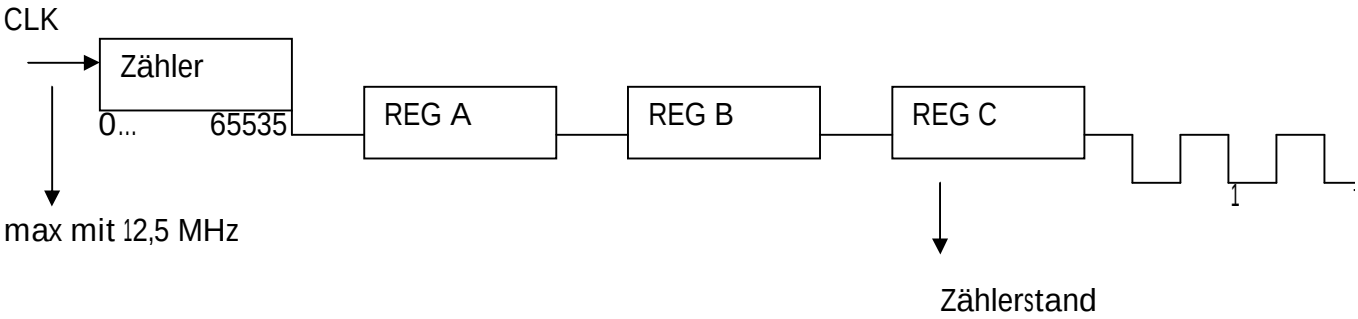
Zusammenfassung Vorlesung MPS 2 31.10.2003

Timer:

2 Seiten: 16Khz

Verhältnis beider Frequenzen lässt rückschlüsse auf Gewicht zu.

Timer: 16 Bit Baustein



12,5 MHz entspricht MCLK / 2 (Machineclock)

= MCLK / 1024 -> 2,7 sek -> Zählerstand erreicht

Im Sekundentakt Signal erzeugen:

IM C-Reg: 24000 eintragen.

MCLK / 1024 -> 1 Sek.

2. Parameter: Steigung(8/32/128) von Zähler über Clockfrequenz

50Hz = 1 /50 = 0,02 sek.

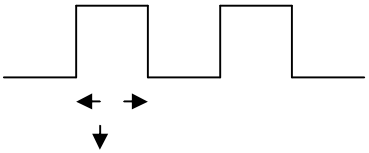
-> 2 weitere Zählerstände um Frequenz zu messen.

Läuft im Comparemodus.
Messen läuft im Capturemodus -> Eingangssignal

Was passiert im Comparemodus ?

1 Interrupt ausgelöst
2. Zähler geht auf 0 zurück

Pulsbreiten Modulation



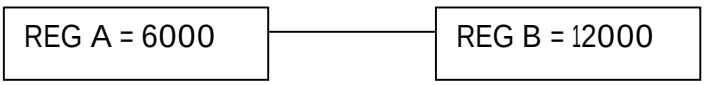
Pulsbreite ist ein Verhältnis von High zu Low.

-> Lichtorgel.

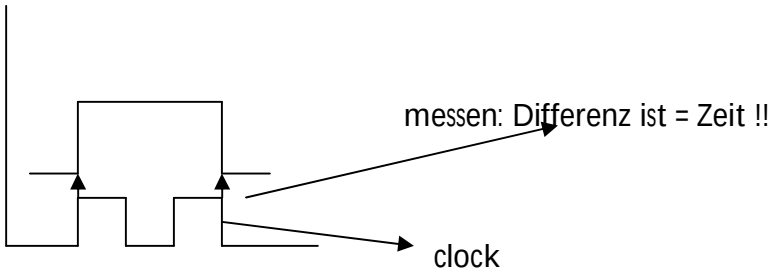
A und B Register beschreiben

Periode = 1 sek. ca Wert: 6000 in Register A

S. 89



CLK



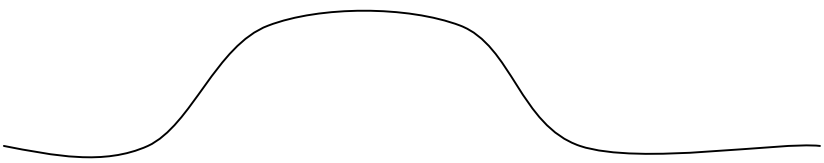
Flanken des ZS messen.

1 sek,. aktivieren.

Die steigenden Flanken kann man über den Kehrwert der Periode ausrechnen.

Analog

Tiefpassfilter: Tiefe Frequenzen rausfiltern.



1. Baustein mit Takt versorgen
2. Clockselet CLKS. Eingangstakt festlegen.

CLKI -> CLKInvert
bei positiven Flanken wird Zähler inkrementiert

oder

Negierer zu ZP der fallen Flanken

BURST
immer 1

UND-Verknüpfung übt eine Torfunktion aus

XC0 -> externes Gerät -> z.B. Schalter

Auf Microcontroller -2 Timerblöcke mit je 3 Kanälen

24000 = 1 sek. !

24000 in Kanal 0 -> TIOA ...TIOA2. CX unabhängig von Prozessor Takt. Takt für 1 Kanal wg. Minute -> Takt für 3 Kanal wg. Stunden

---

S. 94

TC - TCLKS

Waveformmode = Comparemode

ACPC = wenn Compare Ok = set
itogglen: 1 auf 0 und 0 auf 1

CPCTRG = Zähler zurücksetzen. 2 Reaktionsmöglichkeit.

Wave = 0. -> Compare

CLK -> REG A = 12000 REG B

ACPA clear nach 1 Sekunde
ACPA Set nach ½ sekunde

<u>2. Lösungsmöglichkeit</u>
REG C = 12000
dann togglen
ACPA immer togglen.

LED an Pumpe ist Highactive !!!!!!!!!!!!!!!!!

TIOA3 -> 50Hz Timer schon fertig in .c Datei.

TIOA3 -> PA1
PIOB_PER = (1<< PIO_TIOA3) ->(Define für Pin 1. 1 um 1 nach links)
↓
Weiche scharfschalten

PIOB_CODR = (1 << PIOB_TIOA3) -> Kontrolle hat PIO. Nicht Timersignal.
↓
zurücksetzen
↓
Wer hat Kontrolle ?
→ PIO
PIOB_PDR (1 << PIOB_TIOA3)
↓
Pin jetzt freigeben für Timerbaustein

while (! Key 3 pressed)
{
Taste 1 einschalten
Taste 2 ausschalten
}
ausserhalb PIOB_CODR definieren !!!!!! wg. Dauerhighsignal !!

SWTRG -> setzt Timer zurück und startet ihn.
PMC anpassen und 2x Zeiten einfügen für Praktikumsdatei.
-> timer3_init(); -> Zähler gestartet
-> PDR=(1<<...) -> Kontrolle geben

1 und 2 Ein und ausschalten. 3 Tasten Programmabbruch.
PDR oder PER.

Taktberechnung

25MHz = 25000000 / 1024 = 24414,0625 -> / 50Hz = 488 einschalten
die andere hälfte ist auscchalten also 244. 50/s ist die Impulsbreite.
Zusammenfassung Vorlesung MPS 2 07.11.2003

50Hz = 0,02 Sekunden = 1/50 Sek.

PDR -> aktiv schalten

Im Statusregister das BIT abfragen.

Capture Mode

-> festhalten. Erzeugen eines Eingassignals für Timer-> ca. 16Khz / 32 dividiert wg. Anpassungsplatine.

REG C behält Compareeigenschaft.

A und B haben die Eigenschaft den Zählwert festzuhalten.

Messen: wg. Differenz von A und B.

Wenn REG B captured bleibt Zähler stehen.

500Hz = 0,002 Sekunden.

Serielle Schnittstelle

Synchrone Datenübertragung
-> gemeinsamer Zeittakt zwischen Sender und Empfänger

Asynchrone Datenübertragung
-> kein gemeinsamer Takt

Übertragungsarten

Volldu, Halbdu und Simpl-ex

Leitungscodes

Binäre Leitungscodes.

NRZ (None Return to Zero) -> hat sich durchgesetzt!!

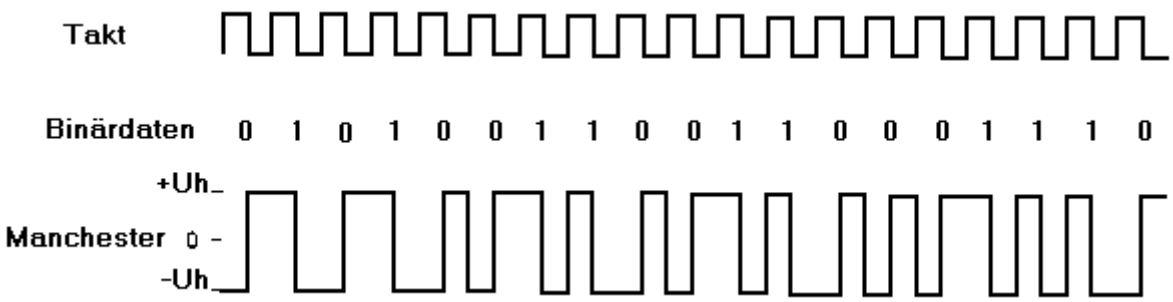
Wenn löngere 1-Folge nicht zurück zu 0.

RZ (Return to Zero)

Halbe Schrittweite bei 1.

NRZI (None Return to Zero Interchange(wechseln))

0: Pegel wechselt
1: bleibt gleich



Manchester Code

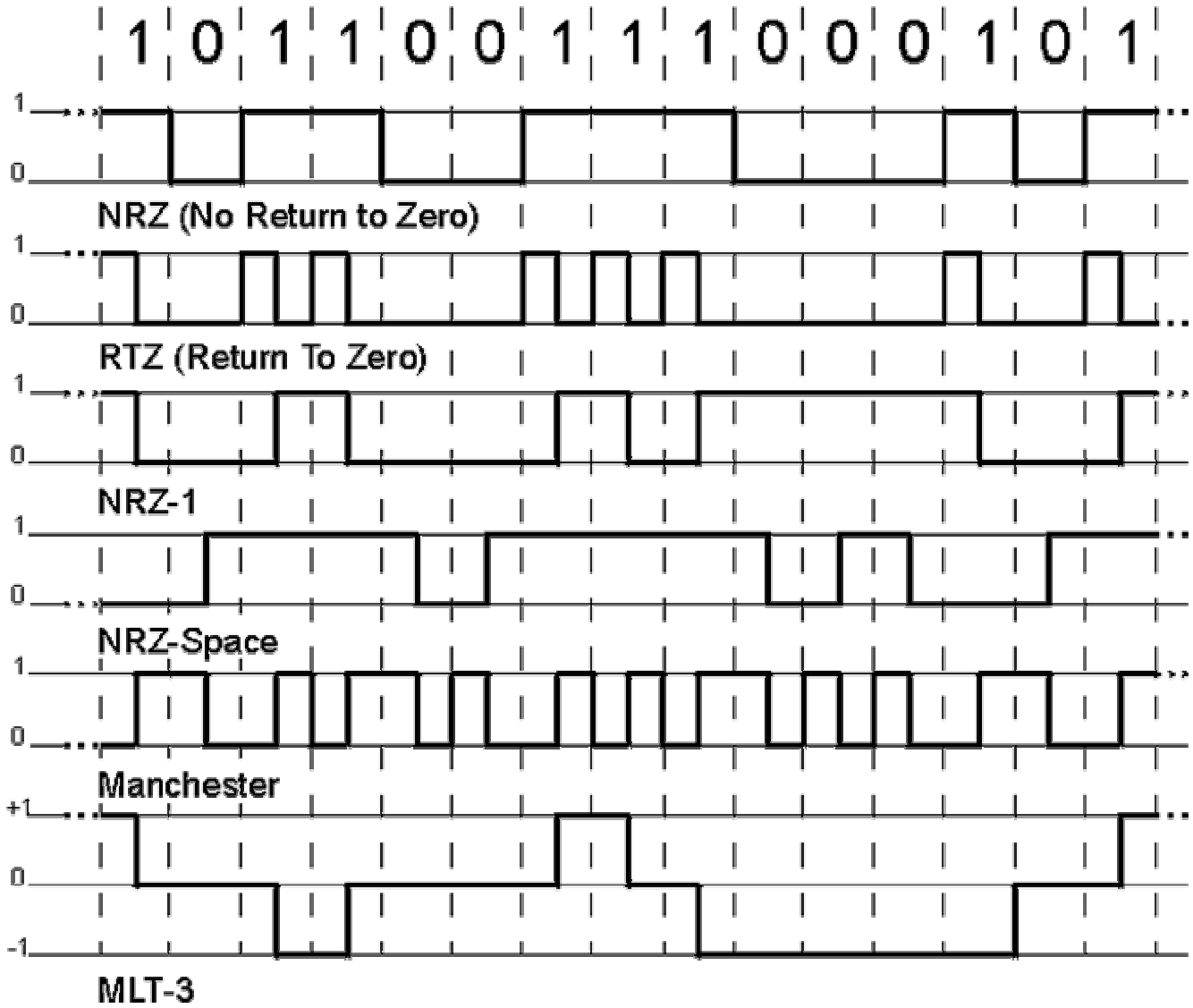
Beim Ethernet eingesetzt. EOR_ logisch verknpüft mit NRZ.

Bei synchroner Kommunikation genutzt.

(x EOR y)

EOR X = Y
Takt- und Nutzinformationen unterscheiden!

Schaubild aller Codes



Zusammenfassung Vorlesung MPS 2 12.11.2003

<u>Async-Mode</u>

Start+Stop-Bit | EOR 0

Parity-Bit -> 0 1 0 0 1 1 1 0 1 ⟶ Ergebnis bei gerade Parität
-> 1 1 0 1 0 0 1 1

gerade 5 = 1 Bit
unterade = 0 Bit
EOR-Verknüpfung leistet 1 Erzeugung!

Sync-Mode

Manchester-Code

Grundaufgabe

Serialisieren von Byte in Bitströmen

-> 3 USART - Bausteine bzw. Kanäle(async).

THR + TSR - Register im Sendeteil -> Parity generieren.

RHR - RSR - Register im Empfangsteil -> Parity prüfen,

Im SR-Register des Steuerwerks Status abfragbar.

Serial_init(); Bits

5 4 3 2 1 0
.....0 0 0 0 0 1

Shiften um 4 (1<<5)
1. T,R ausschalten (CR)
2. 8N1 (MR)

ODD = ungerade Parität

BRGR

S. 124 Fehler 0 und 1 tauschen

Im Praktikum 38400 Baud -> Baudrate erzeugen (CD)
welcher Wert für Klausur berechnen können !

Baudrate = Selected Clock / 16 * CDRegister

CD = SelectClock / 16 * Baudrate S127 - 129

1. T,R ausschalten (CR)
2. 8N1 gesetzt (MR)
Baudrate eingestellt (BRGR)
3. T,R einschalten

LDMIA ro!{r2-r3} lädt alles was in L1 steht und arbeitet ab

getch()

tst setzt Condition codes

LDRNE -> 32 Bit. Haben kein LDRB
andne r0,r1, #255 maskieren der letzten 8 Bits, um Byteinformation zu erzeugen. LDRB!

=FF oder 8 mal die 1

int getch_nonblock(char *p) -> Prototyp in C.

Zusammenfassung Vorlesung MPS 2 26.11.2003

Softwareinterrupt

Im Usermode nur Zugriff auf Flags NZCV nur obersten 4 Bits (31-28) änderbar.
Rest ist geschützt und nur im SV-Mode möglich.

S. 134 im Skript

CPSR kopiert in SPSR_SVC

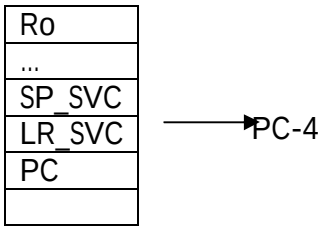
S. 136

CPSR (SVC, Int. Disable Bit gesetzt)

Mapping von SP,LR

Re-Entering Fähigkeit

Mapping von SP,LR



PC-4 wird in LR gespeichert.

Ausgeführte Adresse + 8 -> wg. Pipelining des PC

Bei PC-4 in externer IRQ ist PC-8 gemeint!

IRQ: -> externer Interrupt
SUBS: Softwareinterrupt pc,lr,#4

SVC:
movs pc,lr

Boot.s -> Vektortabelle

IRQ->0x18
LDR pc, [pc,-#F20]

Wert von P0x20(dez. 32)

AIC_IVR: 0xFFFFF100
PC! -> 0 = 0x18, 4 = 0x18, 8 = 0x18 -> Pipelining

LDR PC, [pc,#0x18]
pc = 0x28 -> dez. 40 geht an .word bei swi_handler in boot.s

PC ist immer eins zurück. Deswegen alle 0x18 in boot.s.

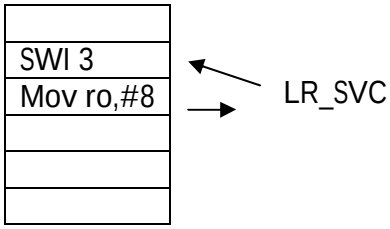
CPSR: 0x93 für disable SWI.

S. 142

Beim Austritt aus SWI muss Statusregister zurückgesichert werden.

SPSR_SVC -> CPSR
LR_SVC->PC

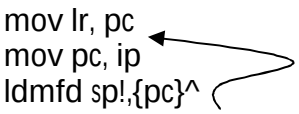
LR-4 = PC-8



S. 146 lernen!

SWI 0,1,2 springt Adresse.

mov lr,pc: PC wird gespeichert wg. kein Branch-Befehl. Rpcksprungadresse wird gesichert.
-> ldmfd wieder laden auf Sprungadresse.



<u>Status Register nach Register Transfer</u>

mrs Status nach Register
-> komplementär
msr Register nach Statusregister

Setzen der Flags im Usermode N, C, Z, V gezielt setzen.

MSR CPSR_f, #0xF0000000 (alle CPSR Bits setzen)

<u>Setzen des C-Flags (Carry)</u>

1. Holen der aktuellen Statusflags wies aussieht:

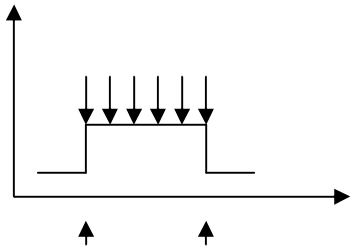
MRS r0, CPSR -> rausholen
ORR r0,r0,0x20000000 -> einzelnes Carrybit setzen
MSR CPSR_f , r0 -> Status zurückschreiben

Minicom heißt die Konsole unter Linux, mit der wir was ausgeben!

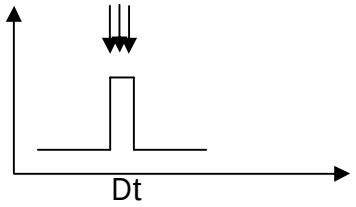
Zusammenfassung Vorlesung MPS 2 17.12.2003

SMR: Source Mode Register

-edge triggered (Flanke)
-level sensitive

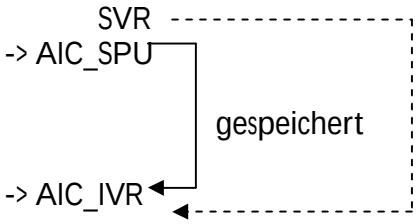


Interrupt schon gelesen. Bis noch gar nicht aktiv.



-spurious Interrupt

AIC erkennt Leseanforderung des IVR.



ISR_SPURIOUS Code Einfügen! S. 57

MSR / MRS

Bei oder Verh. bleiben Bits unverändert (andere).

-> Re-Entering Fähigkeit
Interrupts disablen andere Interrupts durch Prios.

Im Usermode nur die obersten 4 Bits (CPSR) änderbar!

S. 49

MSR CPSR_f -> setzen-> NZCV des CPSR

1)

1	1	1	1

2)

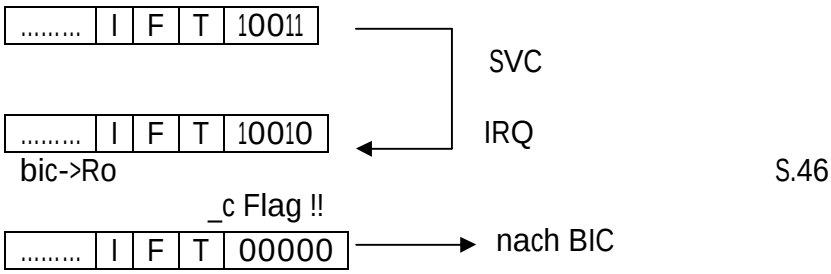
1	0	1	1

3)

MRS oder Verh.

Ro 1 0 0 1
oder 0 0 1 0 ox2
Ergeb. 1 0 1 1

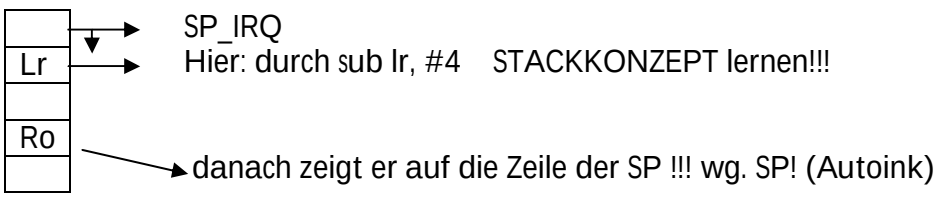
Vom Supervisor in Interruptmode



Re-Entrant S. 59

define / redefine

automap: LR und SP des Interrupt !



MRS LR, SPSR -> Speichern des Status

In Sys-Mode wechseln

Manipulation des CPSR
BIC gibt Interrupt frei. In C nicht möglich weil man von dort nicht in den SV-Mode kann.
I_Bit wird gelöscht und Int. Disable Bit freigegeben
!

Jetzt Exit aus I-Mode. S. 60

S. 65 Fragen bei Vektortabelle -> immer an 0 beginnt. DHS-Sections !!