microcontroller basics
a description based on TI’s MSP430

author and speaker
Prof. Dr. Matthias Sturm

based on TI’s design seminar MSP430
topics

- architectural overview
- memory configuration
- instruction set and addressing modes
- software development ★
- stack and subroutines
- interrupt process
- system clock generator
- periphery
  - parallel ports ★
  - basic timer 1 ★
  - LCD driver module ★
  - ADC ★
  - 8-bit interval timer / counter
  - timer / port module ★
  - watchdog timer module
<table>
<thead>
<tr>
<th>Handheld Measurement</th>
<th>Utility Metering</th>
<th>Home environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air Flow measurement</td>
<td>Gas Meter</td>
<td>Air conditioning</td>
</tr>
<tr>
<td>Alcohol meter</td>
<td>Water Meter</td>
<td>Control unit</td>
</tr>
<tr>
<td>Barometer</td>
<td>Heat Volume Counter</td>
<td>Thermostat</td>
</tr>
<tr>
<td>Data loggers</td>
<td>Heat Cost Allocation</td>
<td>Boiler control</td>
</tr>
<tr>
<td>Emission/Gas analyser</td>
<td>Electricity Meter</td>
<td>Shutter control</td>
</tr>
<tr>
<td>Temperature measurement</td>
<td></td>
<td>White goods</td>
</tr>
<tr>
<td>Weight scales</td>
<td></td>
<td>(Washing machine,..)</td>
</tr>
<tr>
<td>Medical Instruments</td>
<td>Sports equipment</td>
<td>Misc.</td>
</tr>
<tr>
<td>Blood pressure meter</td>
<td>Bike computer</td>
<td>Smart card reader</td>
</tr>
<tr>
<td>Blood sugar meter</td>
<td>Diving watches</td>
<td>Taxi meter</td>
</tr>
<tr>
<td>Breath measurement</td>
<td></td>
<td>Smart Batteries</td>
</tr>
<tr>
<td>EKG system</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Glass break sensors</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Door control</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Smoke/fire/gas detectors</td>
<td></td>
</tr>
</tbody>
</table>
architectural overview: configuration ‘320

- CPU
- JTAG
- Power
- FLL
- ROM
- RAM
- Watch
- ADC 12+2 bit
- Bus conv.
- MDB
- MAB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port

microcontroller basics
Prof. Dr. Matthias Sturm HTWK Leipzig
CPU register

<table>
<thead>
<tr>
<th>special register</th>
<th>universal register</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 programm counter PC</td>
<td>R4 universal register</td>
</tr>
<tr>
<td>R1 stack pointer SP</td>
<td>R5 universal register</td>
</tr>
<tr>
<td>R2 status register SR</td>
<td>R6 universal register</td>
</tr>
<tr>
<td>R3 const. generator CG2</td>
<td>R7 universal register</td>
</tr>
<tr>
<td></td>
<td>R8 universal register</td>
</tr>
<tr>
<td></td>
<td>R9 universal register</td>
</tr>
<tr>
<td></td>
<td>R10 universal register</td>
</tr>
<tr>
<td></td>
<td>R11 universal register</td>
</tr>
<tr>
<td></td>
<td>R12 universal register</td>
</tr>
<tr>
<td></td>
<td>R13 universal register</td>
</tr>
<tr>
<td></td>
<td>R14 universal register</td>
</tr>
<tr>
<td></td>
<td>R15 universal register</td>
</tr>
</tbody>
</table>

16 bit

microcontroller basics
## CPU register

<table>
<thead>
<tr>
<th>special register</th>
<th>universal register</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 programm counter PC</td>
<td>R4 universal register</td>
</tr>
<tr>
<td>R1 stack pointer SP</td>
<td>R5 universal register</td>
</tr>
</tbody>
</table>

### R0 programm counter

```
15  1  0
0    0
```

### Universal registers

- R11 universal register
- R12 universal register
- R13 universal register
- R14 universal register
- R15 universal register

---

**microcontroller basics**

Prof. Dr. Matthias Sturm HTWK Leipzig
### CPU register

**special register**
- R0 programm counter PC
- R1 stack pointer SP

**universal register**
- R4 universal register
- R5 universal register

#### R1 stack pointer

```
   15  1  0
    0
```

#### R11 universal register
- R12 universal register
- R13 universal register
- R14 universal register
- R15 universal register

16 bit
CPU register

**special register**
- R0 programm counter (PC)
- R1 stack pointer (SP)

**universal register**
- R4 universal register
- R5 universal register
- R6 universal register
- R7 universal register
- R8 universal register
- R9 universal register
- R10 universal register
- R11 universal register
- R12 universal register
- R13 universal register
- R14 universal register
- R15 universal register

R2 *status register*

```
  V | SCG1 | SCG0 | OSC_off | CPU_off | GIE | N | Z | C
```

- **V**: Valid
- **SCG1** and **SCG0**: System Clock Group 1 and 0
- **OSC_off**: Oscillator Off
- **CPU_off**: CPU Off
- **GIE**: Global Interrupt Enable
- **N**: Negative
- **Z**: Zero
- **C**: Carry

16 bit
formats of numbers

unsigned

0x0000h
0x0001h
0x7FFFh
0x8000h
0x4000h
0x32768
0x49152
0x65535
0x32767
0x16384
0x7FFFh
formats of numbers

signed

-1FFFFh 0000h 0001h 1

-16384 C000h ± 16384

-32768 8000h 32767

-32768 8000h 7FFFh 32767

0
**Flags**

**Flags** are set or cleared in dependence of logic or arithmetic instructions. **Flags** are used to control the program flow.

**Z-Flag** *(zero)*
set, if the result of a logic or arithmetic instruction is zero, otherwise cleared.

**N-Flag** *(negative)*
set, if the result of a logic or arithmetic instruction is negative, otherwise cleared.

The N-Flag is a copy of the most significant bit (MSB)
Flags

**C-Flag (carry)**
set, if the result of a logic or arithmetic instruction produced a carry, otherwise cleared.

**V-Flag (overflow)**
set, if the result of an arithmetic instruction overflows the signed variable range, otherwise cleared.

- positiv + positive = negative
- negativ + negative = positive
- positive - negative = negative
- negative - positive = positive
memory configuration MSP430P325

Prof. Dr. Matthias Sturm HTWK Leipzig

microcontroller basics
memory model MSP430P325

- **FFFFh**: interrupt vectors
- **FFE0h**: 16 kB OTP
- **FFDFh**: unused
- **C000h**: unused
- **03FFh**: 512 Byte RAM
- **0200h**: 16 bit periphery
- **01FFh**: 8 bit periphery
- **0100h**: special function register
- **00FFh**: unused
- **0010h**: unused
- **000Fh**: unused
- **0000h**: unused

Word access: OTP EPROM
Byte access: RAM
MSP430P325 starter kit

The board
### memory model MSP430P325

**pre-programmed in starter kit**

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFh</td>
<td>interrupt vectors</td>
</tr>
<tr>
<td>FFE0h</td>
<td>monitor</td>
</tr>
<tr>
<td>FFDFh</td>
<td></td>
</tr>
<tr>
<td>EA00h</td>
<td></td>
</tr>
<tr>
<td>C000h</td>
<td>16 kB OTP</td>
</tr>
<tr>
<td></td>
<td>unused</td>
</tr>
<tr>
<td>03FFh</td>
<td>512 Byte RAM</td>
</tr>
<tr>
<td>0200h</td>
<td></td>
</tr>
<tr>
<td>01FFh</td>
<td>16 bit periphery</td>
</tr>
<tr>
<td>0100h</td>
<td></td>
</tr>
<tr>
<td>00FFh</td>
<td>8 bit periphery</td>
</tr>
<tr>
<td>0010h</td>
<td></td>
</tr>
<tr>
<td>000Fh</td>
<td>special function register</td>
</tr>
<tr>
<td>0000h</td>
<td></td>
</tr>
</tbody>
</table>

**OTP EPROM**

**RAM**
### Memory Model MSP430P325

#### RAM-area in starter kit

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3FEh</td>
<td>interrupt vectors</td>
</tr>
<tr>
<td>3E0h</td>
<td>identification bit pattern</td>
</tr>
<tr>
<td>3DFh</td>
<td>stack used by application</td>
</tr>
<tr>
<td>3DCh</td>
<td>stack needed from monitor, 50 Bytes</td>
</tr>
<tr>
<td>214h</td>
<td>user address range 3DCh - 214h</td>
</tr>
<tr>
<td>212h</td>
<td>(do not set the stack pointer)</td>
</tr>
<tr>
<td>200h</td>
<td>RAM-area for monitor</td>
</tr>
</tbody>
</table>

**RAM**
instruction set and addressing modes

- CPU
- ROM
- RAM
- Watch
- ADC 12+2 bit
- JTAG
- Power
- FLL
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- Basic Timer
- Watch
- ADC 12+2 bit
instruction set

- fifty-one instructions
  - twenty-seven basic instructions ⇒ RISC
  - twenty-four emulated instructions ⇒ CISC

- byte and word processing

- seven address modes for source

- four address modes for destination

- all instructions appropriate for all modules
<table>
<thead>
<tr>
<th>Format I</th>
<th>Format II</th>
<th>Format III</th>
</tr>
</thead>
<tbody>
<tr>
<td>source,destination</td>
<td>source or destination</td>
<td>+/- 9bit offset (Word)</td>
</tr>
<tr>
<td>ADD(.B)</td>
<td>CALL</td>
<td>JMP</td>
</tr>
<tr>
<td>ADDC(.B)</td>
<td>PUSH(.B)</td>
<td>JC</td>
</tr>
<tr>
<td>AND(.B)</td>
<td>RETI</td>
<td>JNC</td>
</tr>
<tr>
<td>BIC(.B)</td>
<td>RRA(.B)</td>
<td>JEQ</td>
</tr>
<tr>
<td>BIS(.B)</td>
<td>RRC(.B)</td>
<td>JNE</td>
</tr>
<tr>
<td>BIT(.B)</td>
<td>SWPB</td>
<td>JGE</td>
</tr>
<tr>
<td>CMP(.B)</td>
<td>SXT</td>
<td>JL</td>
</tr>
<tr>
<td>DADD(.B)</td>
<td></td>
<td>JN</td>
</tr>
<tr>
<td>MOV(.B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB(.B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBC(.B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR(.B)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 12 instructions
- 7 instructions
- 8 instructions

- 1 .. 6 cycles
- 1 .. 5 cycles
- 2 cycles

- 1 .. 3 word
- 1 .. 2 word
- 2 word
emulated instructions are:
- basic instruction to the user
- replaced with a basic instruction by the assembler

emulated instructions benefits:
- increased processing speed
- increased ROM code efficiency
- supply users with familiar instructions
- no additional effort in CPU:
  RISC with CISC-like instruction set
### Instruction Set - Emulated Instructions

#### Arithmetic
- ADC(.B)
- DADC(.B)
- DEC(.B)
- DECD(.B)
- INC(.B)
- INCD(.B)
- SBC(.B)

#### Logical
- INV(.B)
- RLA(.B)
- RLC(.B)

#### Data
- CLR(.B)
- CLRC
- CLRN
- CLRZ
- POP(.B)
- SETC
- SETN
- SETZ
- TST(.B)

#### Program Flow
- BR
- DINT
- EINT
- NOP
- RET

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>Logical</th>
<th>Data</th>
<th>Program Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC(.B)</td>
<td>INV(.B)</td>
<td>CLR(.B)</td>
<td>BR</td>
</tr>
<tr>
<td>DADC(.B)</td>
<td>RLA(.B)</td>
<td>CLRC</td>
<td>DINT</td>
</tr>
<tr>
<td>DEC(.B)</td>
<td>RLC(.B)</td>
<td>CLRN</td>
<td>EINT</td>
</tr>
<tr>
<td>DECD(.B)</td>
<td></td>
<td>CLRZ</td>
<td>NOP</td>
</tr>
<tr>
<td>INC(.B)</td>
<td></td>
<td>POP(.B)</td>
<td>RET</td>
</tr>
<tr>
<td>INCD(.B)</td>
<td></td>
<td>SETC</td>
<td></td>
</tr>
<tr>
<td>SBC(.B)</td>
<td></td>
<td>SETN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SETZ</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>TST(.B)</td>
<td></td>
</tr>
</tbody>
</table>

7 instructions  | 3 instructions  | 9 instructions  | 5 instructions
How to emulate instructions?

<table>
<thead>
<tr>
<th>emulated instruction</th>
<th>basic instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC.B dst</td>
<td>increment destination</td>
</tr>
<tr>
<td>INCD.B dst</td>
<td>double-incr. destination</td>
</tr>
<tr>
<td>CLRN</td>
<td>clear negative bit</td>
</tr>
<tr>
<td>EINT</td>
<td>enable interrupt</td>
</tr>
<tr>
<td>INV dst</td>
<td>invert destination</td>
</tr>
</tbody>
</table>

The trick is to take constant numbers in basic instructions by using special registers, the constant generators.
### CPU register

<table>
<thead>
<tr>
<th>special register</th>
<th>universal register</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 programm counter PC</td>
<td>R4 universal register</td>
</tr>
<tr>
<td>R1 stack pointer</td>
<td>SP</td>
</tr>
<tr>
<td>R2 status register</td>
<td>SR</td>
</tr>
<tr>
<td>R3 constant generator</td>
<td>CG2</td>
</tr>
<tr>
<td>R5 universal register</td>
<td></td>
</tr>
<tr>
<td>R6 universal register</td>
<td></td>
</tr>
<tr>
<td>R7 universal register</td>
<td></td>
</tr>
<tr>
<td>R8 universal register</td>
<td></td>
</tr>
<tr>
<td>R9 universal register</td>
<td></td>
</tr>
<tr>
<td>R10 universal register</td>
<td></td>
</tr>
<tr>
<td>R11 universal register</td>
<td></td>
</tr>
<tr>
<td>R12 universal register</td>
<td></td>
</tr>
<tr>
<td>R13 universal register</td>
<td></td>
</tr>
<tr>
<td>R14 universal register</td>
<td></td>
</tr>
<tr>
<td>R15 universal register</td>
<td></td>
</tr>
</tbody>
</table>

### R3 constant register

Under different addressing modes:

<table>
<thead>
<tr>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
### CPU register

#### special register
- R0 programm counter PC
- R1 stack pointer SP

#### universal register
- R4 universal register
- R5 universal register

#### R2 status register

<table>
<thead>
<tr>
<th>15</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>SCG1</td>
<td>SCG0</td>
<td>OSC\text{off}</td>
<td>CPU\text{off}</td>
<td>GIE</td>
<td>N</td>
<td>Z</td>
<td>C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Under different addressing modes:

- **0 0 0 0 0 0**
- **0 0 0 0 0 4**
- **0 0 0 0 8**
### instruction set

#### instruction table (example)

<table>
<thead>
<tr>
<th>source form</th>
<th>operand</th>
<th>operation</th>
<th>status bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>src, dst</td>
<td>src+dst-&gt;dst</td>
<td>* * * *</td>
</tr>
<tr>
<td>MOV</td>
<td>src, dst</td>
<td>src -&gt; dst</td>
<td>- - - -</td>
</tr>
</tbody>
</table>
Orthogonality is, when all instructions with all address modes are valid for all operands.

Orthogonality in MSP430...
all single operand instructions use all seven address modes.
and all double operand instructions use all seven source address modes and all four destination address modes.
address modes

- seven address modes for source
- four address modes for destination

<table>
<thead>
<tr>
<th>mode</th>
<th>source</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>register mode</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>indexed mode</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>symbolic mode</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>absolute mode</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>indirect mode</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>indirect autoincrement mode</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>immediate mode</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>
address modes

register addressing mode

ADD R7,R8 ; ( R7 ) + ( R8 ) \(\Rightarrow\) ( R8 )

MOV R5,R6 ; ( R5 ) \(\Rightarrow\) ( R6 )

CLR R5 ; #0 \(\Rightarrow\) ( R5 )

XOR #1,R9 ; Toggle Bit 0 in R9

The operand is contained in one of the registers R0 to R15.

This is the fastest addressing mode and needs the least memory.
indexed addressing mode

MOV 2(SP),R7 ; Move 2nd item of Stack to R7
MOV.B R5,9(R10) ; LSByte (R5) ⇒ ((R10)+9)
ADDC -2(R5),4(R7) ; ((R5)−2)+((R7)+4)+C ⇒ ((R7)+4)

The address of the operand is the sum of the index and the contents of the register.

The indexed mode with index zero may used for “indirect register addressing” of the destination operand.

BIS #8,0(R4) ; Set Bit 3 at address (R4)
symbolic addressing mode (PC relative addressing)

ADD    EDE,TONI       ; (EDE) + (TONI) ⇒ (TONI)
MOV    TONI,EDE       ; Move (TONI) to (EDE)
MOV    R5,TONI        ; (R5) ⇒ (TONI)
MOV    EDE,R8         ; (EDE) ⇒ (R8)

The content of the words EDE / TONI is used for the operation.

Any address in the 64k memory space is addressable both as a source and as a destination.
address modes

absolute addressing mode

```
ADD &CCR1,&CCR2 ; (CCR1) + (CCR2) ⇒ (CCR2)
MOV &P1IN,&P2OUT ; Move (P1IN) to (P2OUT)
MOV R5,&ACTL ; (R5) ⇒ (ADC Control Register)
MOV &TACTL,R8 ; (TACTL) ⇒ (R8)
```

The contents of the fixed addresses are used for the operation.

Used for hardware peripheral modules that are located at an absolute address; used for Position Independent Code.
indirect register addressing mode

ADD  @R8,R9 ; ((R8)) + (R9) ⇒ (R9)
MOV  @R10,0(R12) ; (R10)) ⇒(R12)+0)
MOV  @R5,&ACTL ; (R5)) ⇒ (ADC Control Register)
MOV.B @R4,R8 ; Byte addressed by R4 ⇒ (R8)

The registers are used as a pointer to the operand.

The registers are not modified.

Only for the source operand addressing.
address modes

indirect register addressing with autoincrement

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD @R8+, R9</td>
<td>((R8) + (R9) \Rightarrow (R9), (R8)+2 \Rightarrow (R8))</td>
</tr>
<tr>
<td>MOV @R10+, 0(R12)</td>
<td>((R10)) \Rightarrow ((R12)+0), (R10)+2 \Rightarrow (R10))</td>
</tr>
<tr>
<td>MOV @R5+, &amp;ACTL</td>
<td>((R5)) \Rightarrow (ADC Register), (R5)+2 \Rightarrow (R5))</td>
</tr>
<tr>
<td>MOV.B @R4+, R8</td>
<td>Byte ((R4)) \Rightarrow (R8), (R4)+1 \Rightarrow (R4))</td>
</tr>
</tbody>
</table>

The registers are used as a pointer to the operand.

The registers are incremented accordingly afterwards.

Only for the source operand addressing.
address modes

immediate register addressing mode

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Immediate Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>#0100h,4(R9)</td>
<td>Test Bit 8=1 ? in the 3rd word of a table starting at (R9)</td>
</tr>
<tr>
<td>MOV.B</td>
<td>#01Fh,0(R12)</td>
<td>01Fh (\Rightarrow) (R12)+0</td>
</tr>
<tr>
<td>MOV</td>
<td>#0010,&amp;ACTL</td>
<td>0Ah (\Rightarrow) (ADC Control Register)</td>
</tr>
<tr>
<td>ADD</td>
<td>#0A00h,R8</td>
<td>0A00h + (R8) (\Rightarrow) (R8)</td>
</tr>
</tbody>
</table>

Any immediate 8 or 16 bit constant can be used with the instruction.

Only for the source operand.
steps of software development

tools for software development

- editor
- assembler
- simulator
- debugger
<table>
<thead>
<tr>
<th>Label</th>
<th>instruction</th>
<th>operand(s)</th>
<th>; comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>mov</td>
<td>#0FC17h, R15</td>
<td>; load the value #0FC17h in register R15</td>
</tr>
</tbody>
</table>
**software development**

**assembler directives**

- **.sect**
  - .sect defines initialized named section and associates subsequent code or data with this section
  - example: .sect “INIT”,0214h

- **.set and .equ**
  - .set and .equ directives set a constant value to symbol
  - example: LCDCTL .equ 030h

- **.byte and .word**
  - .byte places one or more 8-bit values into consecutive bytes of current section
  - .word places one or more 8-bit values into consecutive bytes of current section

- **.end**
  - .end terminates assembly, should be the last source statement
programming techniques - stack and subroutines

- CPU
- ROM
- RAM
- Watch
- ADC 12+2 bit
- Power
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port

microcontroller basics
Prof. Dr. Matthias Sturm HTWK Leipzig
stack and subroutine

main process

register area

- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

memory

- stack
- stack
- stack

- instruction A
- instruction B
- instruction C
- instruction D
- instruction B
- instruction C
- instruction B
- instruction C
- instruction E
- instruction B
- instruction C
- instruction F
- instruction B
- instruction C
- instruction G
- JMP

MP

stack area

- low address
- main program
- No subroutine

high address
**stack and subroutine**

**main process**

**register area**
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

**memory**
- Instruction A
- Instruction B
- Instruction C
- Instruction D
- Instruction B
- Instruction C
- Instruction F
- MP

**stack area**

**main program**

**No subroutine**

**low address**

**high address**

**microcontroller basics**

Prof. Dr. Matthias Sturm HTWK Leipzig
**Stack and Subroutine**

**Main Process**

**Register Area**
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

**Memory**
- Stack
- Instruction A
- Instruction B
- Instruction C
- Instruction D
- Instruction E
- Instruction B
- Instruction C
- Instruction F
- Instruction B
- Instruction C
- Instruction G
- JMP MP

**Stack Area**

**Main Program**

**No Subroutine**

**Microcontroller Basics**

Prof. Dr. Matthias Sturm HTWK Leipzig
stack and subroutine

main process

**register area**
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

**memory**
- stack
- instruction A
- instruction B
- instruction C
- instruction D
- instruction E
- instruction B
- instruction C
- instruction F
- instruction B
- instruction C
- instruction G
- JMP

**No subroutine**

- No subroutine

**main program**

- MP

**stack area**

- low address
- high address

**microcontroller basics**
Prof. Dr. Matthias Sturm HTWK Leipzig
**stack and subroutine**

**subroutine process**

<table>
<thead>
<tr>
<th>register area</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3 constant register</td>
<td>stack</td>
</tr>
<tr>
<td>R2 flag register</td>
<td>stack</td>
</tr>
<tr>
<td>R1 stack pointer</td>
<td>stack</td>
</tr>
<tr>
<td>R0 program counter</td>
<td>stack area</td>
</tr>
<tr>
<td></td>
<td>instruction A</td>
</tr>
<tr>
<td></td>
<td>CALL SR</td>
</tr>
<tr>
<td></td>
<td>instruction D</td>
</tr>
<tr>
<td></td>
<td>CALL SR</td>
</tr>
<tr>
<td></td>
<td>instruction E</td>
</tr>
<tr>
<td></td>
<td>CALL SR</td>
</tr>
<tr>
<td></td>
<td>instruction F</td>
</tr>
<tr>
<td></td>
<td>CALL SR</td>
</tr>
<tr>
<td></td>
<td>instruction G</td>
</tr>
<tr>
<td></td>
<td>RET</td>
</tr>
</tbody>
</table>

**main program**

**subroutine**

<table>
<thead>
<tr>
<th></th>
<th>low address</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP</td>
<td>stack area</td>
</tr>
<tr>
<td></td>
<td>high address</td>
</tr>
</tbody>
</table>

**microcontroller basics**

Prof. Dr. Matthias Sturm HTWK Leipzig
stack and subroutine

subroutine process

register area
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

memory
- stack
- stack
- stack

MP
- instruction A
- CALL SR
- instruction D
- CALL SR
- instruction E
- CALL SR
- instruction F
- CALL SR
- instruction G
- JMP MP

SR
- instruction B
- instruction C
- RET

low address

stack area

mainprogram

subroutine

high address
stack and subroutine

subroutine process

register area

- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

memory

- stack
- stack
- stack
- MP
  - instruction A
  - CALL SR
  - instruction D
  - CALL SR
  - instruction E
  - CALL SR
  - instruction F
  - CALL SR
  - instruction G
  - JMP MP
- SR
  - instruction B
  - instruction C
  - RET

low address

stack area

main program

subroutine

high address

microcontroller basics

Prof. Dr. Matthias Sturm HTWK Leipzig
**stack and subroutine**

**subroutine process**

**register area**
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

**memory**
- stack
- stack
- address of instruction D
- instruction A
- CALL SR
- instruction D
- CALL SR
- instruction E
- CALL SR
- instruction F
- CALL SR
- instruction G
- JMP MP
- instruction B
- instruction C
- RET

**microcontroller basics**

Prof. Dr. Matthias Sturm HTWK Leipzig
stack and subroutine

subroutine process

register area

R3 constant register
R2 flag register
R1 stack pointer
R0 program counter

memory

stack
stack
address of instruction D
instruction A
CALL SR
instruction D
CALL SR
instruction E
CALL SR
instruction F
CALL SR
instruction G
MP

SR

CALL SR
instruction C
RET

JMP MP

low address
stack area

mainprogram

subroutine

high address
Stack and subroutine

Subroutine process

Register area
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

Memory
- Stack
- Stack
- Address of instruction D
- Instruction A
- CALL SR
- Instruction D
- CALL SR
- Instruction E
- CALL SR
- Instruction F
- CALL SR
- Instruction G
- JMP MP
- Instruction B
- Instruction C
- RET

MP

SR

Low address

Stack area

Main program

Subroutine

High address
stack and subroutine

subroutine process

**Register area**
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

**Memory**
- Stack
- Stack
- Address of instruction D

**MP**
- Instruction A
- CALL SR
- Instruction D
- CALL SR
- Instruction E
- CALL SR
- Instruction F
- CALL SR
- Instruction G
- JMP MP
- Instruction B
- Instruction C
- RET

**Subroutine**
- Low address
- Stack area
- Mainprogram
- High address
stack and subroutine

subroutine process

register area

- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

memory

stack
stack
address of instruction D

MP
- instruction A
- CALL SR
- instruction D
- CALL SR
- instruction E
- CALL SR
- instruction F
- CALL SR
- instruction G
- JMP MP
- instruction B
- instruction C
- RET

SR

low address

stack area

mainprogram

subroutine

high address
stack and subroutine

subroutine process

register area
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

memory
- stack
- stack
- address of instruction D
- instruction A
- CALL SR
- instruction D
- CALL SR
- instruction E
- CALL SR
- instruction F
- CALL SR
- instruction G
- JMP MP
- instruction B
- instruction C
- RET

MP

SR

low address
stack area
main program
subroutine
high address
stack and subroutine

subroutine process

register area

- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

memory

<table>
<thead>
<tr>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
</tr>
<tr>
<td>address of instruction D</td>
</tr>
</tbody>
</table>

| instruction A    |
| CALL SR         |
| instruction D   |
| CALL SR         |
| instruction E   |
| CALL SR         |
| instruction F   |
| CALL SR         |
| instruction G   |
| JMP MP          |
| instruction B   |
| instruction C   |
| RET             |

MP

SR

low address

stack area

mainprogram

subroutine

high address
Prof. Dr. Matthias Sturm HTWK Leipzig

microcontroller basics

stack and subroutine

subroutine process

register area

- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

memory

- stack
- stack
- address of instruction D

MP

- instruction A
- CALL SR
- instruction D
- CALL SR
- instruction E
- CALL SR
- instruction F
- CALL SR
- instruction G
- JMP MP

SR

- instruction B
- instruction C
- RET

low address

stack area

mainprogram

subroutine

high address

microcontroller basics
**stack and subroutine**

**subroutine process**

- **register area**
  - R3 constant register
  - R2 flag register
  - R1 stack pointer
  - R0 program counter

- **memory**
  - stack
  - stack
  - address of instruction D
  - instruction A
  - CALL SR
  - instruction D
  - CALL SR
  - instruction E
  - CALL SR
  - instruction F
  - CALL SR
  - instruction G
  - JMP MP
  - instruction B
  - instruction C
  - RET

- **Subroutine process**
  - Main program
  - Stack area
  - High address
  - Low address
### Stack and Subroutine

#### Subroutine Process

<table>
<thead>
<tr>
<th>Memory</th>
<th>Stack Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP</td>
<td>instruction A</td>
</tr>
<tr>
<td></td>
<td>CALL SR</td>
</tr>
<tr>
<td></td>
<td>instruction D</td>
</tr>
<tr>
<td></td>
<td>CALL SR</td>
</tr>
<tr>
<td></td>
<td>instruction E</td>
</tr>
<tr>
<td></td>
<td>CALL SR</td>
</tr>
<tr>
<td></td>
<td>instruction F</td>
</tr>
<tr>
<td></td>
<td>CALL SR</td>
</tr>
<tr>
<td></td>
<td>instruction G</td>
</tr>
<tr>
<td></td>
<td>JMP MP</td>
</tr>
<tr>
<td>SR</td>
<td>instruction B</td>
</tr>
<tr>
<td></td>
<td>instruction C</td>
</tr>
<tr>
<td></td>
<td>RET</td>
</tr>
</tbody>
</table>

- **R3** constant register
- **R2** flag register
- **R1** stack pointer
- **R0** program counter

- **Register Area**

- **Low Address**

- **Main Program**

- **Stack Area**

- **High Address**

---

*Prof. Dr. Matthias Sturm HTWK Leipzig*
subroutine process

register area

- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

memory

- stack
- stack
- address of instruction E
- instruction A
- CALL SR
- instruction D
- CALL SR
- instruction E
- CALL SR
- instruction F
- CALL SR
- instruction G
- JMP MP
- instruction B
- instruction C
- RET

stack area

mainprogram

subroutine

low address

high address
Subroutine process

**Register area**
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

**Memory**
- Stack
- Stack
- Address of instruction E
- Instruction A
- CALL SR
- Instruction D
- CALL SR
- Instruction E
- CALL SR
- Instruction F
- CALL SR
- Instruction G
- JMP MP
- Instruction B
- Instruction C
- RET

**Subroutine**
- Low address
- Stack area
- Main program
- High address

Microcontroller basics
Prof. Dr. Matthias Sturm HTWK Leipzig
**microcontroller basics**

**stack and subroutine**

**subroutine process**

---

**register area**

- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

**memory**

- Stack
- Stack
- Address of instruction E
- Instruction A
- CALL SR
- Instruction D
- CALL SR
- Instruction E
- CALL SR
- Instruction F
- CALL SR
- Instruction G
- Instruction B
- Instruction C
- RET

---

**low address**

**stack area**

**main program**

**subroutine**

**high address**
**stack and subroutine**

**subroutine process**

- **register area**
  - R3 constant register
  - R2 flag register
  - R1 stack pointer
  - R0 program counter

- **memory**
  - stack
  - stack
  - address of instruction E
  - instruction A
  - CALL SR
  - instruction D
  - CALL SR
  - instruction E
  - CALL SR
  - instruction F
  - CALL SR
  - instruction G
  - JMP MP
  - instruction B
  - instruction C
  - RET

- **MP**
- **SR**

- **low address**
- stack area
- mainprogram
- subroutine
- high address

**microcontroller basics**

Prof. Dr. Matthias Sturm HTWK Leipzig
stack and subroutine

subroutine process

register area
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

memory
- stack
- stack
- Address of instruction E
- instruction A
- CALL SR
- instruction D
- CALL SR
- instruction E
- CALL SR
- instruction F
- CALL SR
- instruction G
- JMP MP
- instruction B
- instruction C
- RET

MP

SR

low address
stack area
main program
subroutine
high address

microcontroller basics
Prof. Dr. Matthias Sturm HTWK Leipzig
**stack and subroutine**

**subroutine process**

- **register area**
  - R3 constant register
  - R2 flag register
  - R1 stack pointer
  - R0 program counter

- **memory**
  - stack
  - stack
  - address of instruction E
  - instruction A
  - CALL SR
  - instruction D
  - CALL SR
  - instruction E
  - CALL SR
  - instruction F
  - CALL SR
  - instruction G
  - JMP MP
  - instruction B
  - instruction C
  - RET

- **stack area**
  - low address

- **main program**
  - MP

- **subroutine**
  - high address

- **microcontroller basics**

Prof. Dr. Matthias Sturm HTWK Leipzig
subroutine process

- **register area**
  - R3 constant register
  - R2 flag register
  - R1 stack pointer
  - R0 program counter

- **memory**
  - stack
  - stack
  - address of instruction E
  - instruction A
  - CALL SR
  - instruction D
  - CALL SR
  - instruction E
  - CALL SR
  - instruction F
  - CALL SR
  - instruction G
  - JMP MP
  - instruction B
  - instruction C
  - RET

- **stack area**
- **main program**
- **subroutine**
- **high address**
- **low address**
Save data on the stack

• before you save data on the stack you need to initiate the stack by writing the stack address to the stack pointer (R1)

• to push data on the stack use the PUSH instruction
• to pop data from the stack use the POP instruction

• function: last-in first-out

• the stack pointer uses always two Bytes of stack space

remember:
• be careful not to damage addresses on the stack
• avoid underflow and overflow of the stack
Conclusion

• before you call a subroutine you need to initiate the stack by writing the stack address to the stack pointer (R1)

• to call a subroutine use the CALL instruction
• to return from subroutine use the RET instruction

• before a subroutine call the microcontroller saves the following instruction address on the stack

remember:

• the stack location have to be RAM
• the stack is growing to lower addresses
• the stack pointer points to the latest used stack address
programming techniques - interrupts

- CPU
- ROM
- RAM
- Watch
- ADC 12+2 bit
- Power
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port

microcontroller basics

Prof. Dr. Matthias Sturm HTWK Leipzig
generally

- An interrupt request (usually called an interrupt) is generated by an interrupt source.

- An interrupt need to be served by a special program, the interrupt service routine (ISR).

- An interrupt can take place every time independent of program flow (in difference to subroutine calls).

- Interrupts can be maskable or non-maskable.

- Different interrupt sources have different priority.

- To react on an interrupt the most microcontroller containing a vector interrupt system.
**interrupt**

**interrupt process**

**register area**
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

**interrupt sources**
- int. source 1 / local enable
- int. source 2 / local enable

**interrupt logic**
- global interrupt enable GIE
- priority check

**memory**
- stack
- stack
- stack
- instruction A
- instruction B
- instruction C
- instruction D
- instruction E
- instruction X
- instruction Y
- instruction Z
- instruction U
- instruction V
- instruction W
- interrupt vector ISR1
- interrupt vector ISR2
- JMP MP
- RETI

**low address**

**stack area**

**main program**

**interrupt service routine 1**

**interrupt service routine 2**

**interrupt vector table**

**high address**
interrupt

interrupt process

register area

- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

interrupt sources

- int. source 1 / local enable
- int. source 2 / local enable

interrupt logic

- global interrupt enable GIE
- priority check

memory

- stack
- stack
- stack

- instruction A
- instruction B
- instruction C
- instruction D
- instruction E
- JMP MP

- instruction X
- instruction Y
- instruction Z
- RETI

- instruction U
- instruction V
- instruction W
- RETI

interrupt vector ISR1

interrupt vector ISR2

low address

stack area

mainprogram

interrupt service routine 1

interrupt service routine 2

interrupt vector table

high address
interrupt

interrupt process

register area
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

interrupt sources
- int. source 1 / local enable
- int. source 2 / local enable

interrupt logic
- global interrupt enable GIE
- priority check

memory
- stack
- instruction A
- instruction B
- instruction C
- instruction D
- instruction E
- instruction X
- instruction Y
- instruction Z
- instruction U
- instruction V
- instruction W
- JMP MP
- RETI

MP

interrupt vector ISR1
- interrupt vector ISR2

stack area

mainprogram

low address

stack

interrupt service routine 1

interrupt service routine 2

interrupt vector table

high address
Interrupt process

**Interrupt sources**
- Int. source 1 / local enable
- Int. source 2 / local enable

**Interrupt logic**
- Global interrupt enable (GIE)
- Priority check
- Hard wired

**Register area**
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

**Memory**
- Stack
- Instruction A
- Instruction B
- Instruction C
- Instruction D
- Instruction E
- Instruction X
- Instruction Y
- Instruction Z
- Instruction U
- Instruction V
- Instruction W
- Instruction vector ISR1
- Instruction vector ISR2

**Interrupt vectors ISR1 & ISR2**
- JMP MP
- RETI

**Interrupt service routine 1 & 2**

**Mainprogram**

**Stack area**

**Low address**

**High address**
**interrupt process**

**register area**
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

**interrupt sources**
- int. source 1 / local enable
- int. source 2 / local enable

**interrupt logic**
- global interrupt enable GIE
- priority check

**memory**
- stack
- instruction A
- instruction B
- instruction C
- instruction D
- instruction E
- instruction X
- instruction Z
- instruction W
- instruction U
- instruction V
- instruction Y
- instruction Z
- RETI
- instruction U
- instruction V
- instruction W
- RETI

**interrupt vector table**
- interrupt vector ISR1
- interrupt vector ISR2

**mainprogram**
- R0 program counter
- R1 stack pointer
- R3 constant register

**stack area**
- stack

**low address**
- instruction A
- instruction B
- instruction C
- instruction D
- instruction E
- instruction X
- instruction Z
- instruction W

**high address**
- instruction U
- instruction V
- instruction W
- RETI

**interrupt service routine 1**
- ISR1

**interrupt service routine 2**
- ISR2

**priority check**
- hard wired
interrupt

interrupt process

register area
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

interrupt sources
- int. source 1 / local enable
- int. source 2 / local enable

interrupt logic
- global interrupt enable (GIE)
- priority check

memory
- stack
- instruction D address
- instruction A
- instruction B
- instruction C
- instruction D
- instruction E
- JMP MP
- instruction X
- instruction Y
- instruction Z
- RETI
- instruction U
- instruction V
- instruction W
- RETI

interrupt vector ISR1
interrupt vector ISR2

low address

stack area

main program

MP

interrupt service routine 1

interrupt service routine 2

interrupt vector table

high address

microcontroller basics

Prof. Dr. Matthias Sturm HTWK Leipzig
**Interrupt**

**Interrupt process**

**Register area**
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

**Interrupt sources**
- Int. source 1 / local enable
- Int. source 2 / local enable

**Interrupt logic**
- Global interrupt enable GIE
- Priority check

**Memory**
- Stack
- Flag register
- Instruction D address
- Instruction A
- Instruction B
- Instruction C
- Instruction D
- Instruction E
- JMP MP

**Interrupt service routines**
- ISR1
  - Instruction X
  - Instruction Y
  - Instruction Z
  - RETI
- ISR2
  - Instruction U
  - Instruction V
  - Instruction W
  - RETI

**Interrupt vector table**
- Interrupt vector ISR1
- Interrupt vector ISR2

**Microcontroller basics**
interrupt

interrupt process

register area

R3 constant register
R2 flag register
R1 stack pointer
R0 program counter

interrupt sources

int. source 1 / local enable
int. source 2 / local enable

interrupt logic

global interrupt enable GIE
priority check

memory

stack
flag register
instruction D address

MP

instruction A
instruction B
instruction C
instruction D
instruction E
JMP MP

ISR1

instruction X
instruction Y
instruction Z
RETI

ISR2

instruction U
instruction V
instruction W
RETI

Interrupt vector ISR1
Interrupt vector ISR2

low address
stack area
mainprogram
interrupt service routine 1
interrupt service routine 2
interrupt vector table
high address

microcontroller basics

Prof. Dr. Matthias Sturm HTWK Leipzig
interrupt sources
- int. source 1 / local enable
- int. source 2 / local enable

interrupt logic
- global interrupt enable GIE
- priority check

interrupt process

register area
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

memory
- stack
- flag register
- instruction D address

MP
- instruction A
- instruction B
- instruction C
- instruction D
- instruction E

ISR1
- instruction X
- instruction Y
- instruction Z
- RETI

ISR2
- instruction U
- instruction V
- instruction W
- RETI

hard wired
- interrupt vector ISR2

stack area
- low address

main program
- high address

interrupt service routine 1
- interrupt service routine 2

interrupt vector table
interrupt

interrupt process

**register area**
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

**interrupt sources**
- int. source 1 / local enable
- int. source 2 / local enable

**interrupt logic**
- global interrupt enable GIE
- priority check

**memory**
- stack
- flag register
- instruction D address
- instruction A
- instruction B
- instruction C
- instruction D
- instruction E
- JMP MP
- instruction X
- instruction Y
- instruction Z
- RETI
- instruction U
- instruction V
- instruction W
- RETI
- interrupt vector ISR1
- interrupt vector ISR2

**mainprogram**
- MP

**stack area**
- low address
- stack

**interrupt service routine 1**
- ISR1

**interrupt service routine 2**
- ISR2

**interrupt vector table**
- high address

microcontroller basics

Prof. Dr. Matthias Sturm HTWK Leipzig
interrupt

interrupt process

register area

- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

interrupt sources

- int. source 1 / local enable
- int. source 2 / local enable

interrupt logic

- global interrupt enable GIE
- priority check

memory

- stack
- flag register
- instruction D address

- MP
  - instruction A
  - instruction B
  - instruction C
  - instruction D
  - instruction E
  - JMP MP

- ISR1
  - instruction X
  - instruction Y
  - instruction Z
  - RETI

- ISR2
  - instruction U
  - instruction V
  - instruction W
  - RETI

interrupt vector ISR1
interrupt vector ISR2

low address

stack area

main program

interrupt service routine 1
interrupt service routine 2
interrupt vector table

high address

microcontroller basics
Prof. Dr. Matthias Sturm HTWK Leipzig
interrupt

interrupt process

register area
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

memory
- stack
- flag register
- instruction D address
- instruction A
- instruction B
- instruction C
- instruction D
- instruction E
- JMP MP

interrupt sources
- int. source 1 / local enable
- int. source 2 / local enable

interrupt logic
- global interrupt enable GIE
- priority check

low address

stack area

mainprogram

interrupt service routine 1

interrupt service routine 2

interrupt vector table

low address

high address

microcontroller basics

Prof, Dr. Matthias Sturm HTWK Leipzig
interrupt

interrupt process

register area
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

interrupt sources
- int. source 1 / local enable
- int. source 2 / local enable

interrupt logic
- global interrupt enable GIE
- priority check

memory
- stack
- flag register
- instruction D address

MP
- instruction A
- instruction B
- instruction C
- instruction D
- instruction E
- JMP MP

ISR1
- instruction X
- instruction Y
- instruction Z
- RETI

ISR2
- instruction U
- instruction V
- instruction W
- RETI

interrupt vector table
- Interrupt vector ISR1
- Interrupt vector ISR2

low address
stack area
mainprogram
interrupt service routine 1
interrupt service routine 2
interrupt vector table
high address
**Interrupt**

**Interrupt Process**

**Register Area**
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

**Interrupt Sources**
- Int. source 1 / local enable
- Int. source 2 / local enable

**Interrupt Logic**
- Global interrupt enable (GIE)
- Priority check

**Memory**
- Stack
- Flag register
- Instruction D address

**Interrupt Logic**
- Interrupt vector ISR1
- Interrupt vector ISR2

**Instruction Area**
- Instruction A
- Instruction B
- Instruction C
- Instruction D
- Instruction E

**Instruction X**
- Instruction U
- Instruction V
- Instruction W

**JMP MP**

**Interrupt Service Routine 1 (ISR1)**
- Instruction X
- Instruction Y
- Instruction Z
- RETI

**Interrupt Service Routine 2 (ISR2)**
- Instruction U
- Instruction V
- Instruction W
- RETI

**Priority Check**

**Hard Wired**

**Interrupt Vector Table**

**Microcontroller Basics**

Prof. Dr. Matthias Sturm HTWK Leipzig
interrupt

interrupt process

register area
- R3 constant register
- R2 flag register
- R1 stack pointer
- R0 program counter

interrupt sources
- int. source 1 / local enable
- int. source 2 / local enable

interrupt logic
- global interrupt enable GIE
- priority check

memory
- stack
- flag register
- instruction D address
- instruction A
- instruction B
- instruction C
- instruction D
- instruction E
- JMP MP
- instruction X
- instruction Y
- instruction Z
- RETI
- instruction U
- instruction V
- instruction W
- RETI
- interrupt vector ISR1
- interrupt vector ISR2

low address
stack area
mainprogram

interrupt service routine 1
interrupt service routine 2
interrupt vector table

microcontroller basics
Prof. Dr. Matthias Sturm HTWK Leipzig
### Interrupt Process

#### Register Area
- R3: Constant register
- R2: Flag register
- R1: Stack pointer
- R0: Program counter

#### Memory
- Stack
- Flag register
- Instruction D address
- Instruction A
- Instruction B
- Instruction C
- Instruction D
- Instruction E
- JMP MP

#### Interrupt Sources
- Int. source 1 / local enable
- Int. source 2 / local enable

#### Interrupt Logic
- Global interrupt enable (GIE)
- Priority check

#### Interrupt Vectors
- ISR1
- ISR2
- Instruction U
- Instruction V
- Instruction W
- RETI

#### Stack Area
- Low address
- Main program

#### Interrupt Service Routines
- Service routine 1
- Service routine 2

#### Interrupt Vector Table
- High address

---

*Image: Inserted for representation purposes.*
## Interrupt Vector Table

<table>
<thead>
<tr>
<th>Int. Source</th>
<th>Int. Flag</th>
<th>system Int.</th>
<th>Word address</th>
<th>mirror address</th>
<th>priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>power-up reset</td>
<td>RSTI</td>
<td>reset</td>
<td>0FFFEh</td>
<td>03FEh</td>
<td>15, highest</td>
</tr>
<tr>
<td>external reset</td>
<td>WDI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>watchdog timer</td>
<td>WDTIFG</td>
<td>maskable</td>
<td>0FFF4h</td>
<td>03F4h</td>
<td>10</td>
</tr>
<tr>
<td>Oscillator fault</td>
<td>NMIIFG</td>
<td>nonmaskable</td>
<td>0FFFCh</td>
<td>03FCh</td>
<td>14</td>
</tr>
<tr>
<td>Dedicated I/O</td>
<td>P0.0IFG</td>
<td>maskable</td>
<td>0FFFAh</td>
<td>03FAh</td>
<td>13</td>
</tr>
<tr>
<td>Dedicated I/O</td>
<td>P0.1IFG</td>
<td>maskable</td>
<td>0FFF8h</td>
<td>03F8h</td>
<td>12</td>
</tr>
<tr>
<td>ADC timer</td>
<td>ADCIFG</td>
<td>maskable</td>
<td>0FFE8h</td>
<td>03E8h</td>
<td>4</td>
</tr>
<tr>
<td>timer flags located in module register</td>
<td>flags located in module register</td>
<td>maskable</td>
<td>0FFE8h</td>
<td>03E8h</td>
<td>4</td>
</tr>
<tr>
<td>basic timer</td>
<td>BTIFG</td>
<td>maskable</td>
<td>0FFE2h</td>
<td>03E2h</td>
<td>1</td>
</tr>
<tr>
<td>I/O port 0</td>
<td>P0.2..7IFG</td>
<td>maskable</td>
<td>0FFE0h</td>
<td>03E0h</td>
<td>0, lowest</td>
</tr>
</tbody>
</table>
interrupt

programming interrupts
program structure

- initiate
- main program
- Interrupt service routine 1
- Interrupt service routine 2
- Interrupt service routine 3

Priority falling
interrupt

program steps

- initiate
  - stack and stack pointer
  - resources of the main program
  - resources of interrupt subroutines
  - local interrupt enable
  - global interrupt enable
- main program
  - instructions of main program
  - loop inside the main program
- interrupt subroutine 1
  - instructions of the interrupt subroutine
  - end ISR with RETI instruction
- interrupt subroutine 2
  - instructions of the interrupt subroutine
  - end ISR with RETI instruction

- Don’t forget to initiate the interrupt vector table!
system clock generator

- Power
- FLL
- JTAG
- CPU
- ROM
- RAM
- Watch
- ADC 12+2 bit

Bus conv.

- 8 bit Timer Port
- Basic Timer
- LCD
- I/O Port

MAB

MDB
features

- One crystal - no external components
- Stable processor frequency - no accumulating error
- Fast start up

Low power Oscillator for 32.768 kHz crystal

FLL

\[ f_{MCLK} = (N + 1) \times f_{ACLK} \]

MCLK
Main System Clock \( (f_{System}) \)

ACLK
Auxiliary Clock

XIN

XOUT
system clock generator

frequency locked loop

ACLK

: ( N + 1 )

N

MCLK

f_{System}

M

SCFQCTL

SCFI1

SCFI0

U/D

CLK

frequency integrator

10 bit

modulator

DC Gen

digitally controlled oscillator (DCO)

Control of operation mode

OscOff, SCG0, SCG1

set interrupt flag in SFR's

... FN_4 FN_3 FN_2 ...

SCFI0

set interrupt flag in SFR's

... 2^6 2^5 2^4 2^3 2^2 ...

SCFI1

Control of operation mode

M

2^6 2^5 2^4 2^3 2^2 2^1 2^0

10 bit

frequency integrator

ACLK

: ( N + 1 )

N

M

SCFQCTL

SCFI1

SCFI0

U/D

CLK

frequency integrator

10 bit

modulator

DC Gen

digitally controlled oscillator (DCO)

Control of operation mode

OscOff, SCG0, SCG1

set interrupt flag in SFR's

... FN_4 FN_3 FN_2 ...

SCFI0

set interrupt flag in SFR's

... 2^6 2^5 2^4 2^3 2^2 ...

SCFI1

Control of operation mode

M

2^6 2^5 2^4 2^3 2^2 2^1 2^0

10 bit

frequency integrator

ACLK

: ( N + 1 )

N

M

SCFQCTL

SCFI1

SCFI0

U/D

CLK

frequency integrator

10 bit

modulator

DC Gen

digitally controlled oscillator (DCO)

Control of operation mode

OscOff, SCG0, SCG1

set interrupt flag in SFR's

... FN_4 FN_3 FN_2 ...

SCFI0

set interrupt flag in SFR's

... 2^6 2^5 2^4 2^3 2^2 ...

SCFI1

Control of operation mode

M

2^6 2^5 2^4 2^3 2^2 2^1 2^0
system clock generator

system clock frequency

MCLK = f(System) = ( N + 1 ) × f(crystal)

N in the range of 3 .. 127

MCLK_{max} = f(System)_{max} = 3.3 \text{ MHz}

- three register used for control
  - system clock frequency control register \text{ SCFQCTL } 0052h
  - system clock frequency integrator 0 register \text{ SCFI0 } 0050h
  - system clock frequency integrator 1 register \text{ SCFI1 } 0051h

- oscillation fault interrupt flag \text{ OFIFG } \text{ IFG 1.1 } 0002h
- oscillation Fault interrupt enable \text{ OFIE } \text{ IE 1.1 } 0000h

use byte instructions
## System Clock Generator

### System Clock Operation Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Crystal Osc.</th>
<th>DC Generator</th>
<th>DCO Loop Control</th>
<th>Comments</th>
<th>Typical Current at Vcc=3V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>conditions after PUC</td>
<td>3000µA (400µA/C325)</td>
</tr>
<tr>
<td>LPM1</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>loop control off, CPU off</td>
<td>70µA</td>
</tr>
<tr>
<td>LPM2</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>DCO and loop control off</td>
<td>6µA</td>
</tr>
<tr>
<td>LPM3</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>only crystal osc. On</td>
<td>1.3µA</td>
</tr>
<tr>
<td>LPM4</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>all functions disabled</td>
<td>0.1µA</td>
</tr>
</tbody>
</table>

### R2 Status Register

```
<table>
<thead>
<tr>
<th>15</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>SCG1</td>
<td>SCG0</td>
<td>OSCoff</td>
<td>CPUoff</td>
<td>GIE</td>
<td>N</td>
<td>Z</td>
<td>C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Microcontroller basics

Prof. Dr. Matthias Sturm HTWK Leipzig
system clock generator

crystal buffer output

CBCTL  crystal buffer control register

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CBSEL1</td>
<td>CBSEL0</td>
<td>CBE</td>
<td></td>
</tr>
</tbody>
</table>

microcontroller basics

Prof. Dr. Matthias Sturm HTWK Leipzig
system clock generator

program example

- configures the system clock to 1.05MHz by crystal frequency of 32768Hz

```
START  mov.b  #1Fh, &SCFQCTL ; set multiplication
        ; factor of PLL to 32
```
periphery

- CPU
- ROM
- RAM
- JTAG
- FLL
- Power
- Watch
- ADC 12+2 bit
- Bus conv.
- MAB
- MDB
- MAB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- Power
- JTAG
- Bus conv.
### Periphery Register

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFh</td>
<td>Interrupt vectors</td>
</tr>
<tr>
<td>FFE0h</td>
<td>Monitor</td>
</tr>
<tr>
<td>FFDFh</td>
<td>16 kB OTP</td>
</tr>
<tr>
<td>EA00h</td>
<td>Unused</td>
</tr>
<tr>
<td>C000h</td>
<td>512 Byte RAM</td>
</tr>
<tr>
<td>03FFh</td>
<td>16 bit periphery</td>
</tr>
<tr>
<td>0200h</td>
<td>RAM</td>
</tr>
<tr>
<td>01FFh</td>
<td>8 bit periphery</td>
</tr>
<tr>
<td>0100h</td>
<td>Special function register</td>
</tr>
<tr>
<td>00FFh</td>
<td></td>
</tr>
<tr>
<td>0010h</td>
<td></td>
</tr>
<tr>
<td>000Fh</td>
<td></td>
</tr>
<tr>
<td>0000h</td>
<td></td>
</tr>
</tbody>
</table>
### Periphery Register / Special Function Register

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000h</td>
<td>special function register</td>
</tr>
<tr>
<td>0001h</td>
<td>interrupt enable 1</td>
</tr>
<tr>
<td>0002h</td>
<td>interrupt flag register 1</td>
</tr>
<tr>
<td>0003h</td>
<td>interrupt flag register 2</td>
</tr>
<tr>
<td>0004h</td>
<td>module enable 1</td>
</tr>
<tr>
<td>0005h</td>
<td>module enable 2</td>
</tr>
<tr>
<td>0006h</td>
<td>reserved</td>
</tr>
<tr>
<td>0007h</td>
<td>reserved</td>
</tr>
<tr>
<td>0100h</td>
<td>8 bit periphery</td>
</tr>
<tr>
<td>0101h</td>
<td>16 bit periphery</td>
</tr>
<tr>
<td>01FFh</td>
<td>512 Byte RAM</td>
</tr>
<tr>
<td>0200h</td>
<td>RAM</td>
</tr>
<tr>
<td>03FFh</td>
<td>special function register</td>
</tr>
</tbody>
</table>

---

**Microcontroller Basics**

Prof. Dr. Matthias Sturm HTWK Leipzig
### Periphery Register / Byte Modules

<table>
<thead>
<tr>
<th>Address Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0080h - 008Fh</td>
<td>reserved</td>
</tr>
<tr>
<td>0070h - 007Fh</td>
<td>USART register</td>
</tr>
<tr>
<td>0060h - 006Fh</td>
<td>reserved</td>
</tr>
<tr>
<td>0050h - 005Fh</td>
<td>system clock generator register</td>
</tr>
<tr>
<td>0040h - 004Fh</td>
<td>basic timer, 8-Bit timer/counter, timer/port register</td>
</tr>
<tr>
<td>0030h - 003Fh</td>
<td>LCD register</td>
</tr>
<tr>
<td>0020h - 002Fh</td>
<td>digital I/O port P3 and P4 control register</td>
</tr>
<tr>
<td>0010h - 001Fh</td>
<td>digital I/O port P0, P1 and P2 control register</td>
</tr>
</tbody>
</table>

#### Memory Map

- **512 Byte RAM**
- **16 bit periphery**
- **8 bit periphery**
- **Special function register**
### Periphery Register / Word Modules

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0180h - 018Fh</td>
<td>reserved</td>
</tr>
<tr>
<td>0170h - 017Fh</td>
<td>timer_A</td>
</tr>
<tr>
<td>0160h - 016Fh</td>
<td>timer_A</td>
</tr>
<tr>
<td>0150h - 015Fh</td>
<td>reserved</td>
</tr>
<tr>
<td>0140h - 014Fh</td>
<td>reserved</td>
</tr>
<tr>
<td>0130h - 013Fh</td>
<td>multiplier</td>
</tr>
<tr>
<td>0120h - 012Fh</td>
<td>watchdog timer</td>
</tr>
<tr>
<td>0110h - 011Fh</td>
<td>analog-to-digital converter</td>
</tr>
<tr>
<td>0100h - 010Fh</td>
<td>reserved</td>
</tr>
</tbody>
</table>

#### RAM

- 512 Byte RAM
- 16 bit periphery
- 8 bit periphery
- Special function register
general port P0

features

- 8 Bit parallel port
  - bit programmable
  - individual function select
  - interrupt source selection
    (three interrupt vectors)
### general port P0

- **six register used for control**

<table>
<thead>
<tr>
<th>Register Type</th>
<th>Register</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>input register</td>
<td>P0IN</td>
<td>0010h</td>
</tr>
<tr>
<td>output register</td>
<td>P0OUT</td>
<td>0011h</td>
</tr>
<tr>
<td>direction register</td>
<td>P0DIR</td>
<td>0012h</td>
</tr>
<tr>
<td>interrupt flags</td>
<td>P0IFG</td>
<td>0013h</td>
</tr>
<tr>
<td>interrupt edge select</td>
<td>P0IES</td>
<td>0014h</td>
</tr>
<tr>
<td>interrupt enable</td>
<td>P0IE</td>
<td>0015h</td>
</tr>
</tbody>
</table>

*use byte instructions*
general port P0

Pad logic

interrupt edge select

interrupt flag

P0IES.x

P0IFG.x

P0IE.x

schematic of bits P0.7 to P0.3

output buffer

P0BIT.x

P0OUT.x

P0DIR.x

P0IN.x

P0IRA.x
**Program Example**

- **Configures P0.3 and P0.4 as input and checks the pins**

```assembly
bic.b #018h,&P0DIR ; set P0.3 and P0.4 to inputs

main clr R15 ; clear R15
bit.b #008h,&P0IN ; check pin P0.3
jnc P04 ; if not pressed -> check pin P0.4
mov.b #015h,R15 ; if pressed -> load pattern R15
P04 bit.b #010h,&P0IN ; check pin P0.4
jnc next ; if not pressed -> goto next
mov.b #016h,R15 ; if pressed -> load pattern R15
next nop
```
parallel port feature

- five outputs
- one bidirectional channel

- two register used for control
  - TP O/P data register
    - TPD 004Eh
  - TP data enable register
    - TPE 004Fh

use byte instructions
program example

- configures TP0 .. TP5 as output and writes a pattern

```
mov.b #0FFh,&TPE ; enable outputs of Timer/Port
mov.b #015h,R15 ; load pattern R15
OUT mov.b R15,&TPD ; output pattern
```
ADC

parallel port feature

- six inputs

- two register used for control
  - input data register \( \text{AIN} \) \( 0110h \)
  - input enable register \( \text{AEN} \) \( 0112h \)

use word instructions
configures all AD-pins as inputs and checks two pins

```
mov #0FFh,&AEN ; all AD-pins dig.inputs
MAIN bit #01h,&AIN ; check pin AIN.0
    jnc A1 ; if LO -> check pin AIN.1
    mov.b #015h,R15 ; if HI -> load pattern R15
A1 bit #02h,&AIN ; check pin AIN.1
    jnc next ; if LO -> goto next
    bis.b #02Ah,R15 ; if HI -> add pattern to R15
next nop
```
Three functions of the Basic Timer 1

- LCD driver frequency
- basic timings
- real-time clock
basic timer 1

- three register used for control
  - basic timer 1 control register **BTCTL** 0040h
  - basic timer 1 counter 1 register **BTCNT1** 0046h
  - basic timer 1 counter 2 register **BTCNT2** 0047h
  - basic timer interrupt flag **BTIFG** IFG2.7 0003h
  - basic timer interrupt enable **BTIE** IE2.7 0001h

**BTCTL** basic timer 1 control register

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSEL</td>
<td>Hold</td>
<td>DIV</td>
<td>FRFQ1</td>
<td>FRFQ0</td>
<td>IP2</td>
<td>IP1</td>
<td>IP0</td>
</tr>
</tbody>
</table>

use byte instructions
basic timer 1 - examples

simple generation of timings

LCD timing

eexample MUX4

LCD data sheet
\( f_{\text{framing}} = 100\text{Hz}..30\text{Hz} \)

\( ACLK = 32768\text{Hz} \)

FRFQ:
\( f_{\text{LCD}} = 8 \times 100\text{Hz}..8 \times 30\text{Hz} \)
\( f_{\text{LCD}} = 800\text{Hz}..240\text{Hz} \)

select \( f_{\text{LCD}} \)
\( 1024\text{Hz}/512\text{Hz}/256\text{Hz}/128\text{Hz} \)

\( f_{\text{LCD}} = 256\text{Hz} \)
\( FRFQ1 = 1 / FRFQ2 = 0 \)
program example

- configures basic timer1 as clock source for LCD module

```assembly
mov.b #050h,&BTCTL ; set up Basic Timer
; for LCD operation
bis.b #BTME,&ME2 ; enable Basic Timer

mov.b #0FFh,&LCDCTL ; set up LCD driver
mov   #0008h,R7 ; clear display
LOOP  clr.b LCDM-1(R7)
       dec   R7
       jnz   LOOP
```
**LCD driver module**

**features**
- different driving methods
  - static
  - 2MUX
  - 3MUX
  - 4MUX
- memory structure for the segment bits
- up to 30 segment lines per module
- up to 15 digits in 4MUX mode
- On/Off of analog generator capability
- select groups of segment/digital outputs

use byte instructions
7 segment display

4MUX mode
A digit consists of seven segments driven from two segment and four common lines.
LCD driver module

Display memory, 4MUX drive

<table>
<thead>
<tr>
<th>MDB</th>
<th>BIT</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

MAB

- 003Fh: a b c h f g e d digit 15
- 003Eh: a b c h f g e d digit 14
- 003Dh: a b c h f g e d digit 13
- 003Ch: a b c h f g e d digit 12
- 003Bh: a b c h f g e d digit 11
- 003Ah: a b c h f g e d digit 10
- 0039h: a b c h f g e d digit 9
- 0038h: a b c h f g e d digit 8
- 0037h: a b c h f g e d digit 7
- 0036h: a b c h f g e d digit 6
- 0035h: a b c h f g e d digit 5
- 0034h: a b c h f g e d digit 4
- 0033h: a b c h f g e d digit 3
- 0032h: a b c h f g e d digit 2
- 0031h: a b c h f g e d digit 1

segm. n+1 segm. n
; STK/EVK LCD
a .equ 01h
b .equ 02h
c .equ 10h
d .equ 04h
e .equ 80h
f .equ 20h
g .equ 08h
h .equ 40h

;; --- character definitions
LCD_Tab .byte a+b+c+d+e+f ; displays "0"
.byte b+c ; displays "1"
.byte a+b+d+e+g ; displays "2"
.byte a+b+c+d+g ; displays "3"
.byte b+c+f+g ; displays "4"
.byte a+c+d+f+g ; displays "5"
.byte a+c+d+e+f+g ; displays "6"
.byte a+b+c ; displays "7"
.byte a+b+c+d+e+f+g ; displays "8"
.byte a+b+c+d+f+g ; displays "9"
.byte a+b+c+e+f+g ; displays "A"
.byte c+d+e+f+g ; displays "B"
.byte a+d+e+f ; displays "C"
.byte b+c+d+e+g ; displays "D"
.byte a+d+e+f+g ; displays "E"
.byte a+e+f+g ; displays "F"
LCD driver module

- one register used for control

LCD control and mode register

| LCDCTL | LCD control and mode register | LCDCTL | 0030h |

Use word instructions and absolute address mode.

- Segment selection
- Mode selection
program example

```assembly
mov.b #0FFh,&LCDCTL ; set up LCD driver
mov    #0008h,R7    ; clear display
LOOP  clr.b LCDM-1(R7)
dec    R7
jnz    LOOP
MAIN   mov    &NUMBER,R6    ; load value of NUMBER in R6
clr    R7            ; register R7 := 00h
PRINT  mov    R6,R8       ; move R6 to R8
and    #000Fh,R8      ; keep lower 4 bits
mov.b  LCD_Tab(R8),LCDM(R7) ; display digit on LCD
rra    R6            ; rotate right four times
rra    R6
rra    R6
rra    R6
inc    R7
cmp    #04h,R7       ; all 4 digits one
jne    PRINT        ; no -> go to next digit
nop
```
ADC

- Power
- FLL
- CPU
- JTAG
- ROM
- RAM
- Watch
- ADC 12+2 bit
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port

microcontroller basics

Prof. Dr. Matthias Sturm HTWK Leipzig
ADC features

- Programmable 12bit or 14 bit resolution
- Four programmable ranges
- Conversion time <100μsec, 96(12 bit) / 132(14 bit) ADCLK cycles
- Sample&Hold
- Eight Analog/Digital inputs
- Programmable current source
- Ratiometric or Absolute measurement
- Low current consumption, typ. 200μA
- Power-Down feature to stop current consumption
- Large supply voltage range, 2.5V ..... 5.5V
- external or internal reference supply
ADC

Input Buffer Enable (AEN) and Input Buffer (AIN) are connected to the Analog Mux. The analog signals are then decoded by the Resistor Decode and Capacitor ARRAY. The range is selected using ACTL6 and ACTL7. The AIN signals are then input to the Current Mux. The MCLK signal determines the delay time for the conversion. The Successive Approximation Register (SAR) is used for the conversion process. The conversion result is then stored in the Conversion Result (ADAT) register and is output as a digital signal. The bits are delayed and then output as a 16-bit word (MDB, 16 bit).
ADC

radiometric measurement

\[ N = 2^{14} / \left[ R_2 / (R_2 + R_1) \right] \]

\[ N = 2^{14} \times 0.25 \times R_1 / R_2 \]
ADC

**absolute measurement**

SVCC ~ AVCC
SVCC switch closed

0 ≤ SVCC ≤ AVCC
SVCC switch open
ADC

- four register used for control
  - input register: AIN 0110h
  - input enable register: AEN 0112h
  - ADC control register: ACTL 0114h
  - ADC data register: ADAT 0118h
- end-of-conversion flag: ADIFG/IFG2.2 0004h
  - clear by software

ACTL  ADC control register

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ADCLK</td>
<td>Pd</td>
<td>range select</td>
<td>c. source</td>
<td>AD input sel.</td>
<td>V_ref</td>
<td>CS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

use word instructions
ADC

program example

- configures the ADC and works as converter

```assembly
mov   #0900h,&ACTL            ; set up ADC:
    ; - external reference
    ; - ADC input A0
    ; - no current source
    ; - automatic range select
    ; - power on, ADCLK = MCLK
bis.b #020h,&P0DIR            ; switch on external
bis.b #020h,&P0OUT            ; on-board reference

MAIN  bis   #CS,&ACTL         ; start conversion
EOC   bit.b #ADCIFG,&IFG2     ; wait for end of
    ; conversion
jnc   EOC
bic.b #ADCIFG,&IFG2          ; clear flag
mov   &ADAT,R6               ; load conv. result in R6
```
8-bit interval timer / counter

features

☑️ three major functions
- serial communication or data exchange
- pulse counting or pulse accumulation
- timer

☑️ three register used for control
- T/C control register \( TCCTL \) 0042h
- pre-load register \( TCPLD \) 0043h
- counter register \( TCDAT \) 0044h
- P0.1 or counter interrupt flag \( P0IFG \) IFG1.3 0002h
- P0.1 or counter interrupt enable \( P0IE.1 \) IE1.3 0000h
- P0.1 interrupt edge select \( P0IES.1 \) P0IES 0014h

use byte instructions
8-bit interval timer / counter

interrupt logic
P0.1 and 8bit T/C

counter, preload
and control register

serial communication
support

---

Microcontroller Basics
Prof. Dr. Matthias Sturm HTWK Leipzig
8-bit interval timer / counter

serial asynchronous data format
8-bit interval timer / counter

serial asynchronous data format (reading)

begin of frame is the falling edge of start bit

level detection in the middle of the bit

eight data bit

start bit

stop bit
timer / port module

- CPU
- ROM
- RAM
- Watch
- ADC 12+2 bit
- Power
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- JTAG
- Bus conv.
- MAB
- MDB
- 8 bit Timer
- Timer Port
- Basic Timer
- LCD
- I/O Port
- RAM
- FLL
- JTAG
- Bus conv.
timer / port module

features

- six tri-state output ports (one bidirectional)
- precision comparator CMPI for slope A/D conversion or standard digital input CIN
- two 8-bit counters, cascadable for 16-bit counter
- three clock sources for counting up the counter
- three interrupt sources

use byte instructions
timer / port module

Enable Control

TPSEL0

ENB ENA

CMP

MCLK

ACLK

TPDSEL1 TPSSEL0

TPSEL1

0 1 2 3

0

Set_EN1FG

Set_RC1FG

Set_RC2FG

Data Register

TPE

Data Enable Register

TPD

Control Register

TPCTL

Vcc/4

CPON

CIN

CMPI

TP0

TP.0

TPD.0

TPE.0

TP1

TP.1

TPD.1

TPE.1

TP2

TP.2

TPD.2

TPE.2

TP3

TP.3

TPD.3

TPE.3

TP4

TP.4

TPD.4

TPE.4

TP5

TP.5

TPD.5

TPE.5

CLK1

RC1

EN1

CLK2

RC2

EN2
five register used for control

- TP control register
- TP counter 1 register
- TP counter 2 register
- TP O/P data register
- TP data enable register

- timer/port interrupt enable

use byte instructions
timer / port module

program example

- pulse with measurement with timer / port

```assembly
mov.b #060h,&TPCTL ; set clock for TPCNT1 enable TPCNT1 with TP.5-IN
mov.b #00h,&TPE ; disable outputs
mov.b #080h,&TPD ; select 16-bit mode
clr.b &TPCNT2 ; set timer start values
clr.b &TPCNT1
bis.b #TPIE,&IE2 ; enable Timer/Port interrupt
eint ; global interrupt enable
MAIN jmp MAIN ; infinite loop

; Timer/Port Interrupt Service Routine
TPISR bit.b #EN1FG,&TPCTL ; detect interrupt source
jnc TPI_EX ; jump if source = overflow
clr R5 ; prepare counter result
mov.b &TPCNT2,R5 ; read high part of counter result
swpb R5 ; swap low- and high part
clr R6 ; prepare counter result
mov.b &TPCNT1,R6 ; read low part of counter
bis R6,R5 ; build the counter result by logic OR
OUT ... ; print counter result to LCD

clr.b &TPCNT2 ; set timer start values
clr.b &TPCNT1
TPI_EX bic.b #(RC2FG+EN1FG),&TPCTL ; clear interrupt flags
reti ; return from interrupt
```
watchdog timer module

[Diagram of microcontroller basics with various components and modules such as CPU, ROM, RAM, Watch, ADC, Power, FLL, JTAG, Bus conv., 8 bit Timer, Timer Port, Basic Timer, LCD, I/O Port, etc.]
watchdog timer module

features

- eight software selectable time intervals
- two operation modes
  - watchdog
    - expiration of time interval generates a system reset
  - interval timer
    - expiration of time interval generates a interrupt request
- write the watchdog control register is only possible using a password
**Watchdog Timer Module**

- **Password Compare:**
  - Read: HighByte is 069h
  - Write: HighByte is 05Ah, otherwise security key is violated

- **WDT Control Register**
  - HOLD
  - NMI
  - CNTCL
  - IS1
  - IS0

- **16-bit Counter**
  - EN
  - CLK
  - N
  - Q:
    - Q6
    - Q9
    - Q13
    - Q15
  - IS1
  - IS0

- **Interrupts:**
  - **IRQ:** Interrupt Request
  - **IRQA:** Interrupt Request Accepted

- **Power-up Circuitry**
  - **POR**
  - **NMI**
  - **PUC**
  - **Resetwd1**
  - **Resetwd2**

- **Logic:**
  - **WDTQn**
  - **WDTIFG**
  - **IE1.0**
  - **PUC**

- **Protection:**
  - **WDTIE**

- **Additional Logic:**
  - **SSEL**
  - **VCC**
  - **MCLK, ACLK**

**Timer/Counter and Interrupt Logic**

**Power-on Reset POR and Power-up Clear PUC Logic**
**watchdog timer module**

- **one register used for control**
  - **watchdog timer control register**  
    - **WDTCTL**  
    - Address: 0120h
  - Use word instructions

### WDTCTL - watchdog control register (write)

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Password: (5Ah)

### WDTCTL - watchdog control register (read)

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Password: (69h)
watchdog timer module

program example

- Disables the watchdog timer

```asm
WDTCTL .equ 0120h
WDTHold .equ 80h
WDT_wrkey .equ 05A00h

Marke  mov  #(WDTHold+WDT_wrkey),&WDTCTL  ; stop Watchdog
```
MSP430 roadmap

Price

- x11x
- x310
- x320
- x330

CPU, Timer/Port
- CPU, A/D converter
- CPU, H/W MPY
- CPU, USART, Timer_A

Complexity

- with LCD driver
- without LCD driver
- More to come

More to come
architectural overview: configuration ‘330
Special thanks for listening the

MSP430

microcontroller basics
a description based on TI's MSP430

author and speaker
Prof. Dr. Matthias Sturm

based on TI's design seminar MSP430