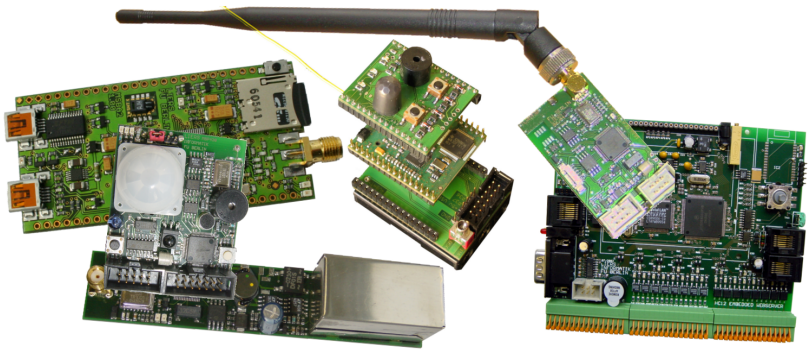


19589 - PS Telematik-Projekt: Wireless Embedded Systems

First Steps

Bastian Blywis, Dr. Achim Liers

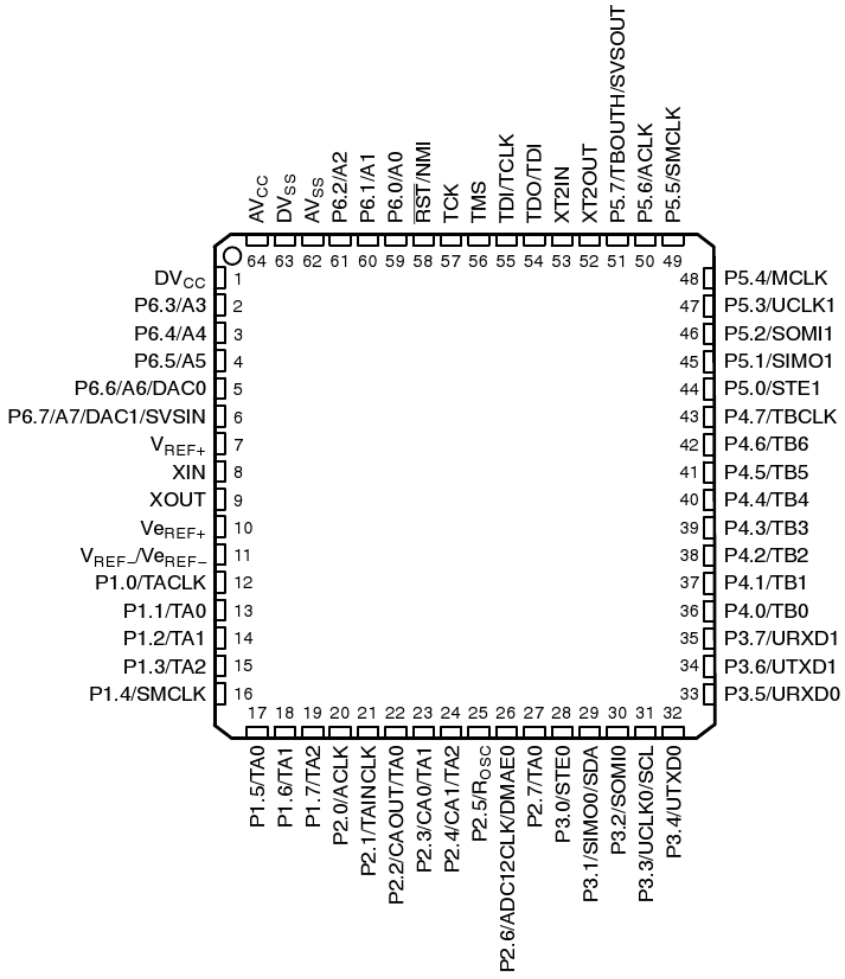
Department of Mathematics and Computer Science
Institute of Computer Science
08. October, 2008

Outline

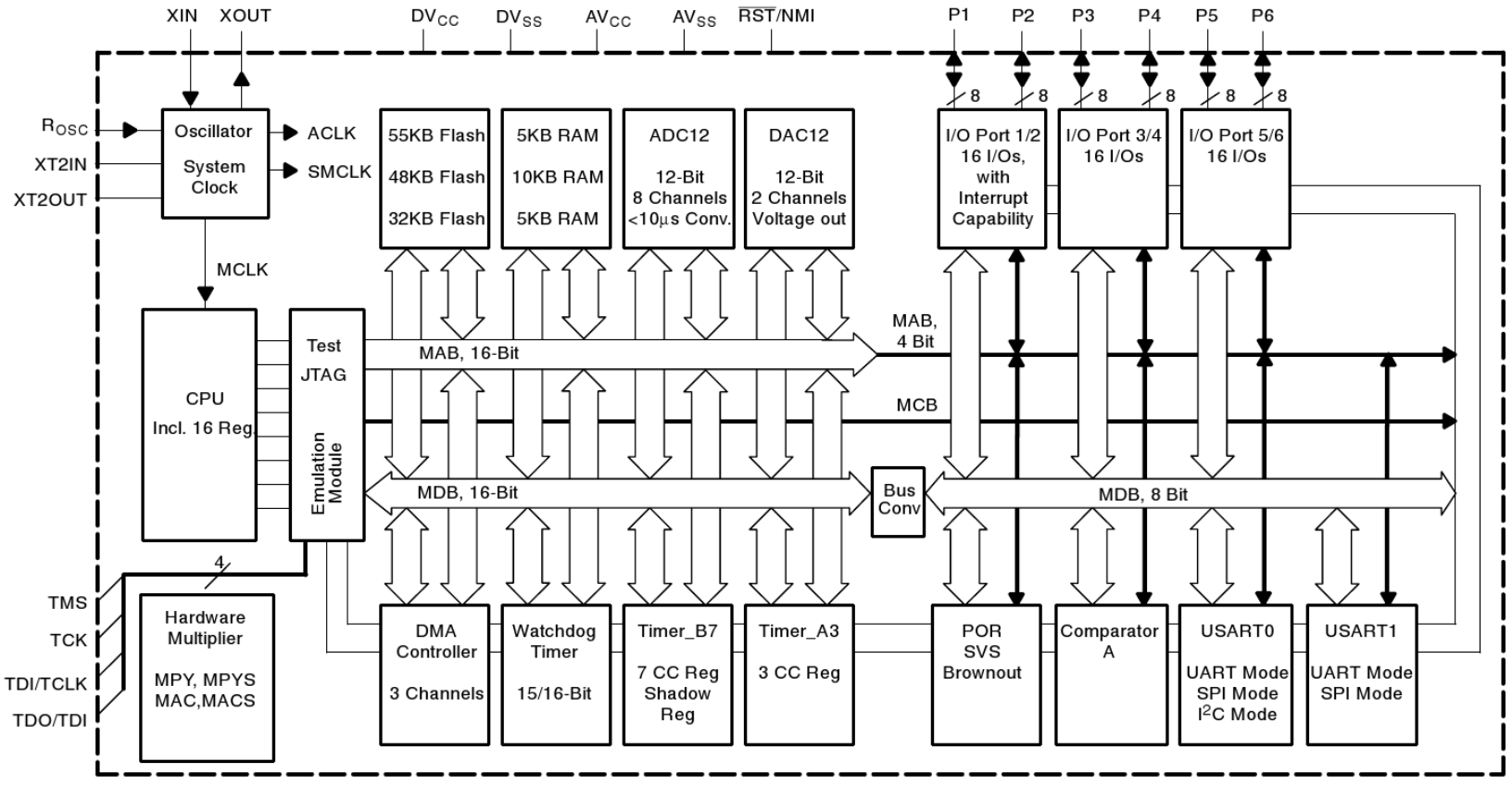
1. Embedded Systems
2. MSP430f1612
3. ScatterWeb^2
4. Software Architecture
5. Interrupt Handling
6. Debugging
7. Terminal Commands
8. Hardware
9. Stacked Modules
10. Source Code
11. Terminal Emulator
12. Write a Terminal Command
13. Parameter Parsing
14. Timer
15. Sending Frames/Packets
16. Receiving Frames/Packets
17. Tasks

Embedded Systems

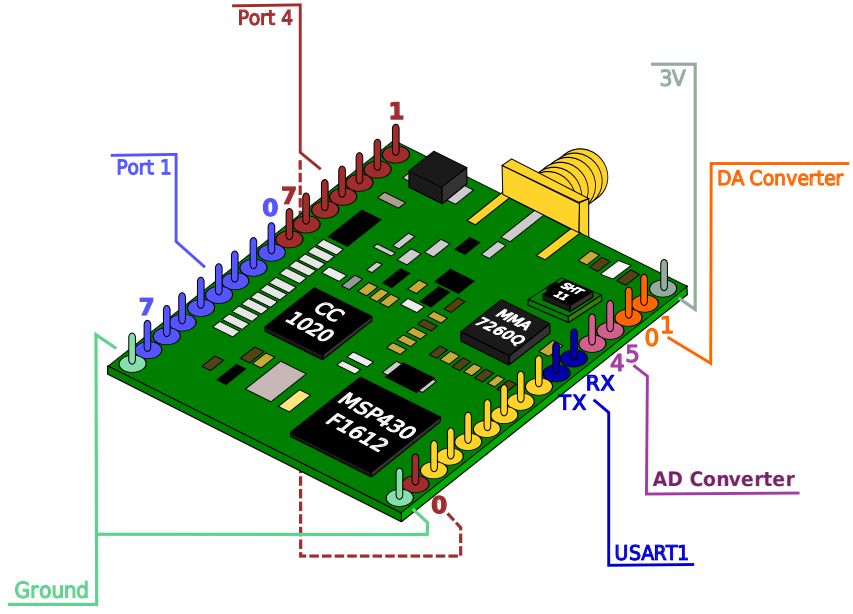
- Slow CPU
- Small RAM and (Flash) ROM Memory
- Often times no MMU
- Limited energy resources
- FPU might be missing
- Specialized OS
- Difficult to debug


AVCC
DVSS
AVSS
P6.2/A2
P6.1/A1
P6.0/A0
RST/NMI
TCK
TMS
TDI/TCLK
TDO/TDI
XT2IN
XT2OUT
P5.7/TBOUTH/SVSOUT
P5.6/ACLK
P5.5/SMCLK
64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49
DVCC 1
P6.3/A3 2
P6.4/A4 3
P6.5/A5 4
P6.6/A6/DAC0 5
P6.7/A7/DAC1/SVSIN 6
VREF+ 7
XIN 8
XOUT 9
VeREF+ 10
VREF-/VeREF- 11
P1.0/TACLK 12
P1.1/TA0 13
P1.2/TA1 14
P1.3/TA2 15
P1.4/SMCLK 16
48 P5.4/MCLK
47 P5.3/UCLK1
46 P5.2/SOMI1
45 P5.1/SIMO1
44 P5.0/STE1
43 P4.7/TBCLK
42 P4.6/TB6
41 P4.5/TB5
40 P4.4/TB4
39 P4.3/TB3
38 P4.2/TB2
37 P4.1/TB1
36 P4.0/TB0
35 P3.7/URXD1
34 P3.6/UTXD1
33 P3.5/URXD0
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
P1.5/TA0
P1.6/TA1
P1.7/TA2
P2.0/ACLK
P2.1/TAINCLK
P2.2/CAOUT/TA0
P2.3/CA0/TA1
P2.4/CA1/TA2
P2.5/ROSC
P2.6/ADC12CLK/DMAE0
P2.7/TA0
P3.0/STE0
P3.1/SIMO0/SDA
P3.2/SOMI0
P3.3/UCLK0/SCL
P3.4/UTXD0

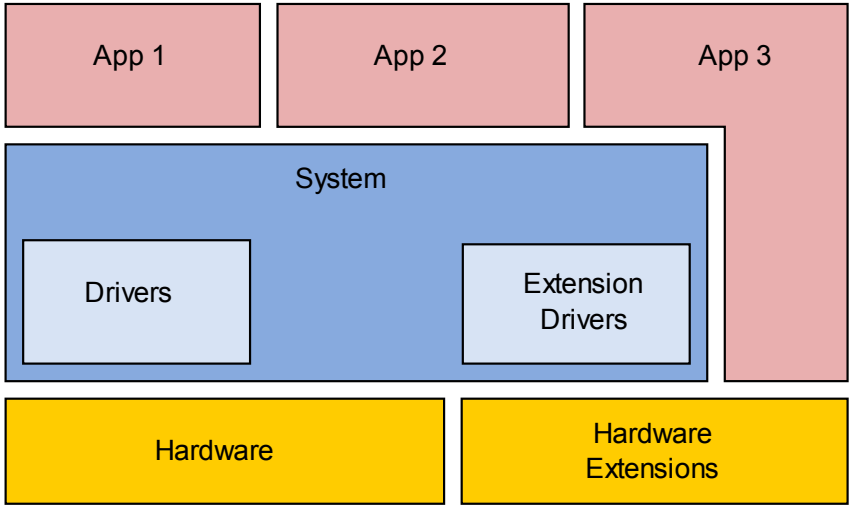
MSP430f1612 (internal)



- Small operation system for the MSB-430. . .
- . . . but more of a collection of code
- Event driven
- No (preemptive) multitasking
- Microcontroller in low power mode most of the time (99%)
- User space applications use kernel functions via libscatterweb
- Documentation needs work



- Hardware: Core module MSB-430
- Extension: MSB-430S
- OS kernel with drivers for core module. . .
- . . . and extension drivers (optional)
- Applications using kernel API (that's what you will develop)
- Applications using hardware directly (rare)



Event based system

1. System initially in low power mode
2. HW interrupt, e.g. voltage applied to I/O pin
3. CPU interrupts current program (if running)
4. registered interrupt service routine (ISR) in IVT gets called
5. ISR sets flag (depending on interrupt source)
6. ISR wakes up CPU
7. superloop detects set flag
8. registered event handler gets called → event handled
   - in kernel and/or
   - in user application
9. CPU reenters low power mode again

Debugging

- Limited
- Via JTAG interface (but doesn't work this well with MSPGCC)
- Write messages to serial out
- Configure MSB-430 via terminal commands

Example, logging:

```
printf( /* some text */, /* some arguments */ );
LOG_LOW( "PONG from %u", args->netheader->from );
```

Example, terminal command:

```
COMMAND(RST, 0, cmdargs) {
    System_reset(WDS_SWRESET);
}
```

Terminal Commands

Terminal commands can be invoked

- locally
- remote

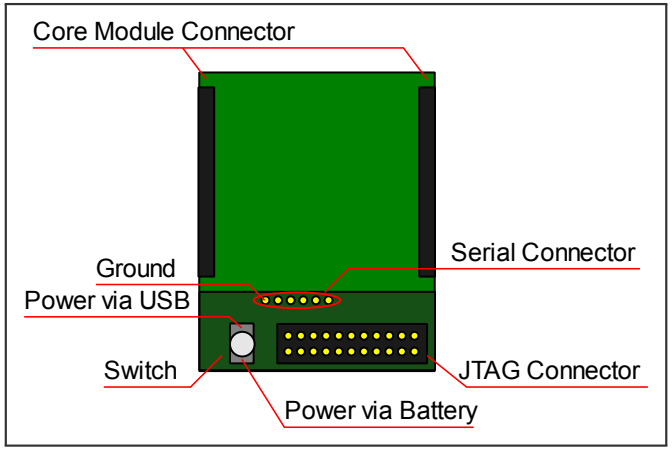
Example, local:

```
$ id
[id] 57
```

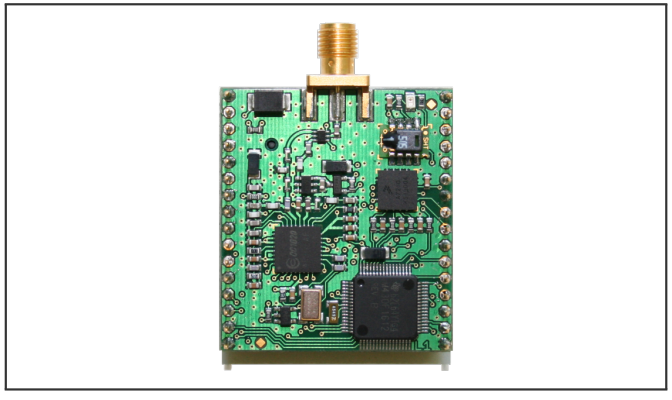
Example, remote:

```
$ @123 id
[123] 123
```

Hardware



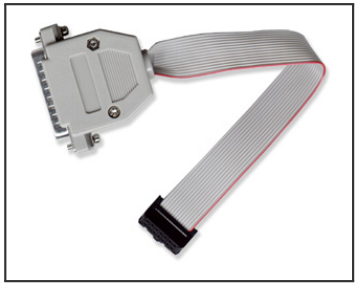
MSB-430T base module



MSB-430

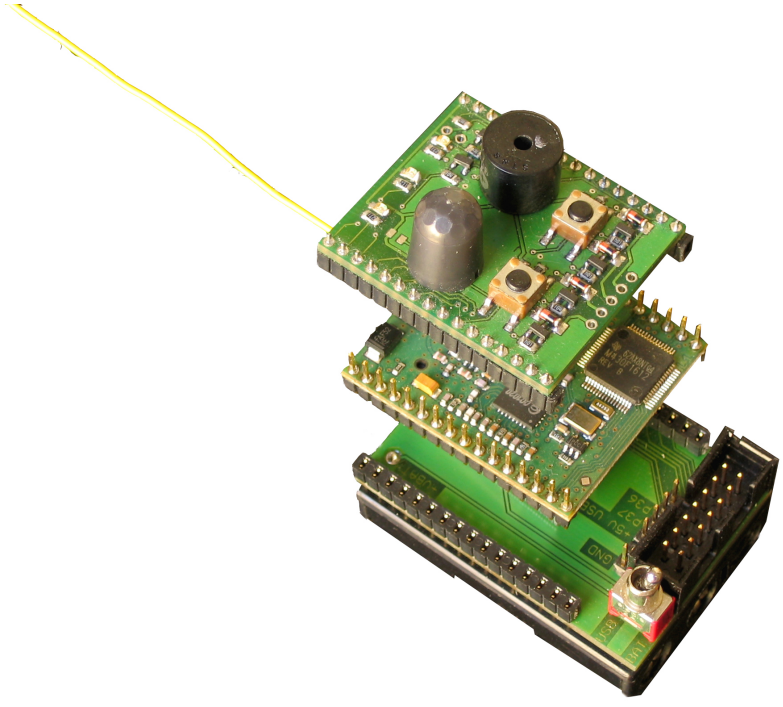
FTDI USB-to-Serial cable

JTAG Interface (USB)



JTAG Interface (parallel)

Stacked Modules



Stacked MSB Modules

Source Code

1. Open Cygwin Bash Shell
2. Download Source (do not use a windows network folder!!!)

```
$ cd SOME_DIR
$ mkdir SOME_NEW_DIR
$ svn co https://projects.mi.fu-berlin.de/svn/scatterweb/...
...MSB-430/lab-branches/WS0809/lab{1,2,3,4,5,6,7,8,9}
```

3. Make copy of [EMPTY] application

```
$ cd lab{1,2,3,4,5,6,7,8,9}/Applications
$ svn cp [EMPTY] first_steps
```

4. Compile application

```
$ cd first_steps
$ make
```

5. Flash to MSB-430

```
$ make flash
```

Terminal Emulator

1. Open a terminal emulator (e.g. TerraTerm, HyperTerminal, hterm)
   - 115200 Baud
   - Data Bits = 8
   - Parity = none
   - Stop Bits = 1
   - Flow Control = None
   - Send line ends with line feed (\r\n or cr-lf)
   - Local Echo = on
2. Reset device (use the hardware switch)
3. Toggle red LED

```
$ led 1
$ led 0
```

Write a Terminal Command

1. Open `src/ScatterWeb.Process.c`
2. Write a new terminal commando

```
COMMAND(HelloMSB, CMDFLAG_SERIAL, cmdargs) {
    printf("Hello User");
}
```

3. Implement a call counter

```
uint8_t counter = 0;

COMMAND(HelloMSB, CMDFLAG_SERIAL, cmdargs) {
    printf("Hello User - called %u times\r\n", counter++);
}
```

Parameter Parsing

- Parse a parameter

```
uint8_t counter = 0;

COMMAND(HelloMSB, CMDFLAG_SERIAL, cmdargs) {
    uint16_t n = String_parseUint16(cmdarg->args, NULL);
    while(n--) {
        printf("Hello User - called %u times", counter++);
    }
}
```

- You can use every function in the `stdlib`...
- ...unfortunately MSPGCC provides only a reduced `stdlib`

Timer

1. Call a function once 5 s after system start

```
void callMe() { /* some action */}

void Process_init() {
    System_registerCallback( C_RADIO, (fp_vp)Process_radioHandler );
    System_registerCallback( C_SENSOR, (fp_vp)Process_sensorHandler );
    Timers_add(5*1024, callMe, 0xFFFF);
}
```

2. Call a function every 5 s

```
void callMe() {
    print("foo bar");
    Timers_add(5*1024, callMe, 0xFFFF); // 1024 ticks == 1 second
}
```

3. View timer table with command `tmr`
4. Toggle the led in your periodic function

Sending Frames/Packets

1. Create header

```
netpacket_send_args_t npsargs;
Net_sendArgsInit(&npsargs);
```

2. Set necessary values

```
npsargs.netheader.to = ???; // choose a destination id
npsargs.netheader.type = USERDEFINED_PACKET+???; // choose a number
npsargs.netheader.flags = 0;
```

   Note: USERDEFINED_PACKET is defined in
   `Libraries/libscatterweb/include/ScatterWeb.Net.PacketTypes.h`
   Add your lab-number to the value to get a unique type id.

3. Add some data

```
uint8_t* data = "Hello world!";
npsargs.payload[0].buffer = (uint8_t*) data;
npsargs.payload[0].size = strlen(data)+1; // include terminating char
```

4. Send frame/packet

```
Net_send(&npsargs);
```

Receiving Frames/Packets

1. Goto `Process_radioHandler()`
2. Include case for your type

```
switch ( args->netheader->type ) {
    case PING_REQUEST_PACKET:
        Net_sendPong(args->netheader->from);
        break;
    case PING_REPLY_PACKET:
        LOG_LOW( "PONG from %u", args->netheader->from );
        break;
    case USERDEFINED_PACKET+???:
        // do something
        break;
    default:
        return false;
}
```

3. Test the sending and receiving of data with your two nodes

Tasks

Traffic Generator

- Write a command that can initiate and terminate the periodic sending of data.
- The interval shall be configurable as parameter.
- Limit the number of sent packets by another parameter.

Dump Network Traffic

- Write a command that switches on/off the dumping of received packets.
- Print the raw header and data to the serial port.
- Decode the header fields in your output, e.g. `To:  123`.
- Enable promiscous mode; receive all packets and not only multicasts and unicast destined for your node

End of the First Steps

Terminal Command Makro


```
#define COMMAND(_cmd, _flags, _args ) \
    COMMAND_HEADER(_cmd, _flags, _args, 0)

#define COMMAND_HEADER(_cmd, _flags, _args, _ptr ) \
    extern void cmd_##_cmd (const cmd_args_t* _args ); \
    const uint8_t cmd_name_##_cmd [] = #_cmd; \
    __attribute__((section(".commands"))) \
    const command_t cmd_##_cmd##_tableentry =  \
        (_flags | CMDFLAG_ASCII) , cmd_##_cmd , \
        (uint16_t)cmd_name_##_cmd , _ptr \
    ; \
    __attribute__((noinline)) \
void cmd_##_cmd (const cmd_args_t* _args)
```