Freie Universität Berlin

Summer Term 2008

19589 - PS Telematik Projekt: Embedded Sensor Web

Achim Liers

Bastian Blywis

Marco Hutta

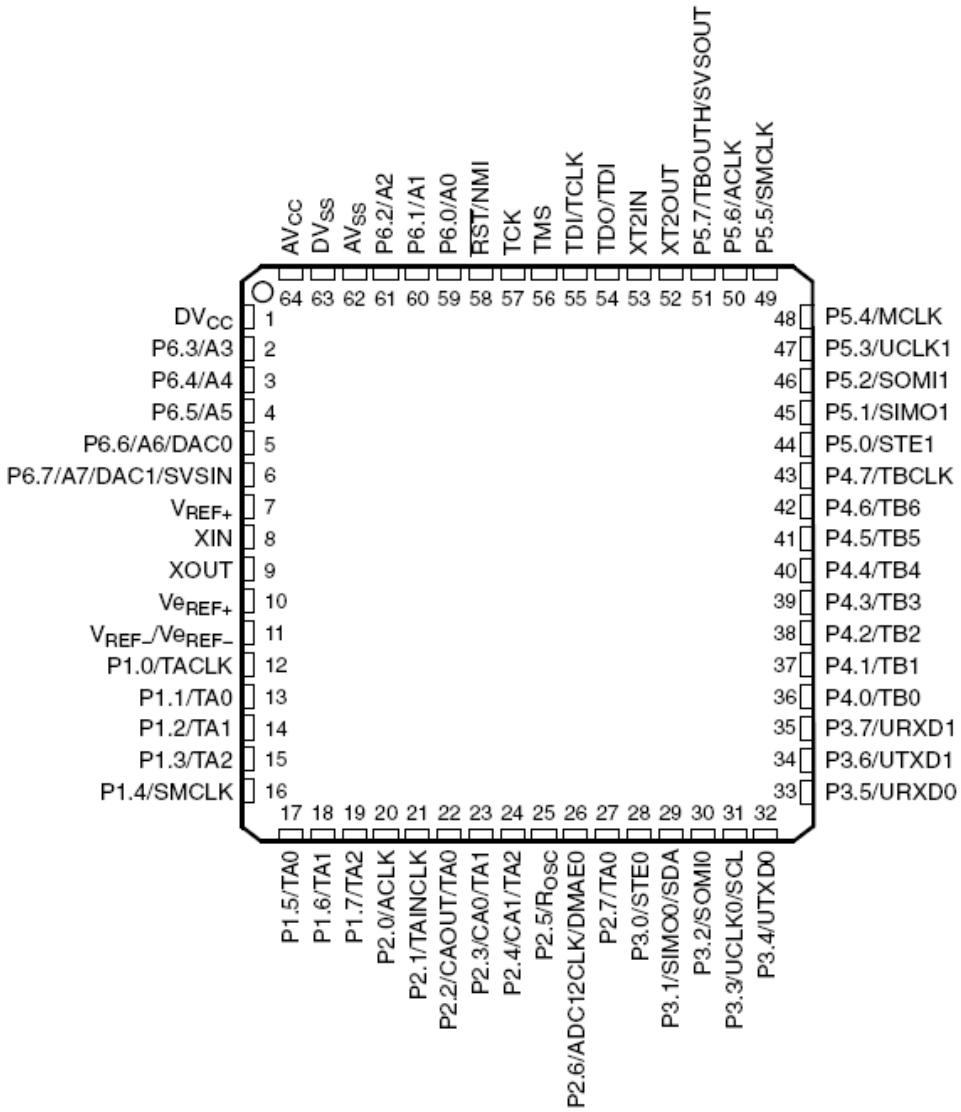
Freie Universität Berlin

ScatterWeb² on the MSB-430

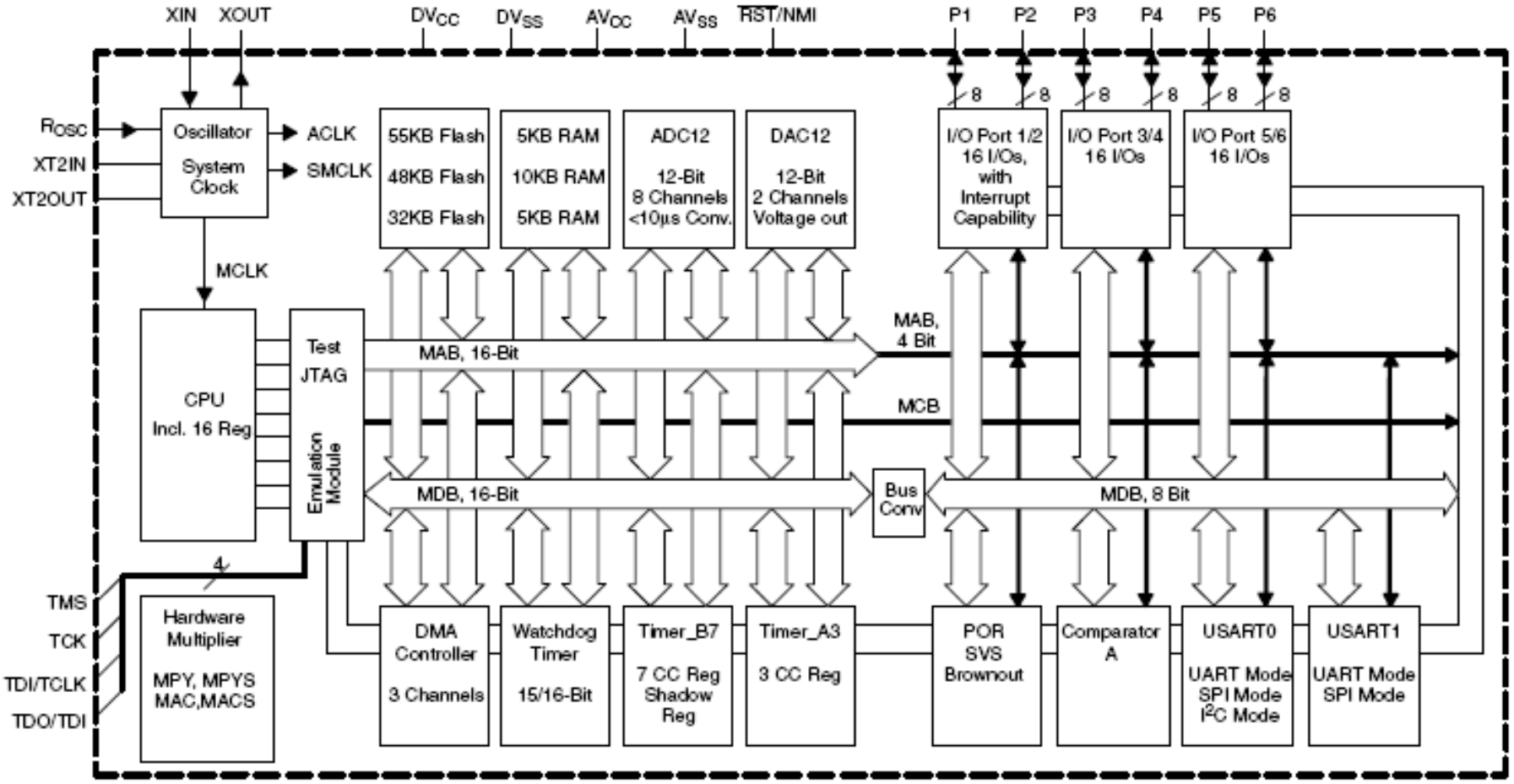
Embedded Systems

- slow CPU
- limited RAM and (Flash) ROM (additional non volatile memory?)
- often no MMU
- often limited power supply
- FPU might be missing
- specialized OS
- more difficult to debug

MSP430f1612

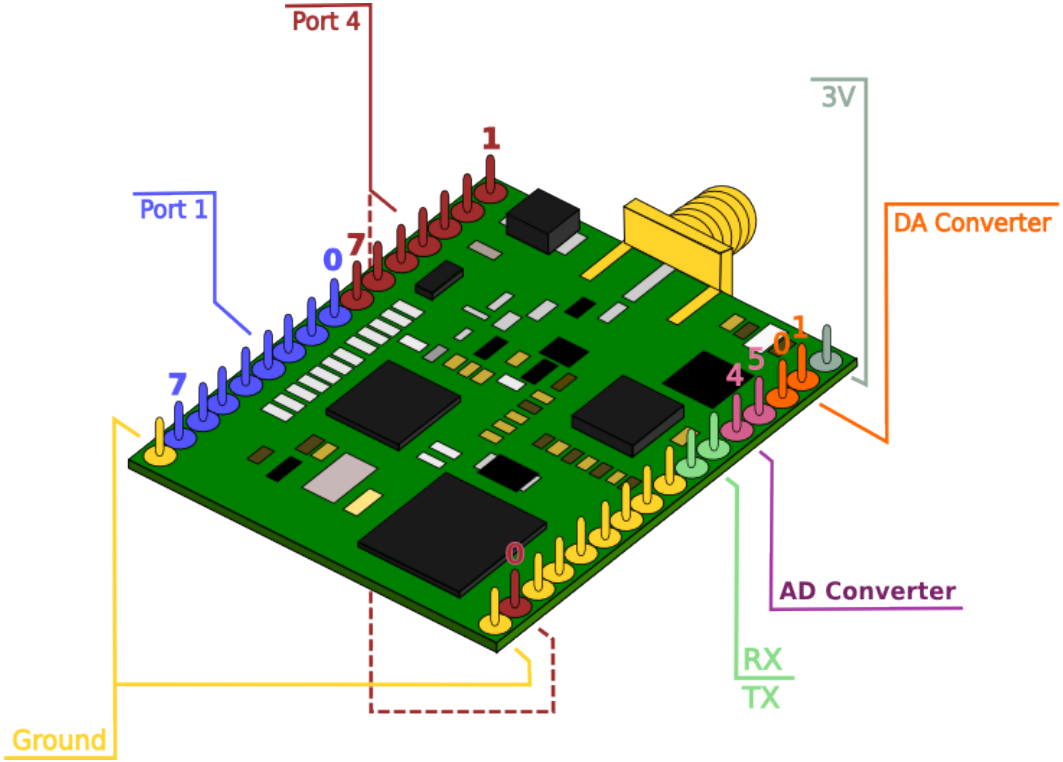


MSP430f1612

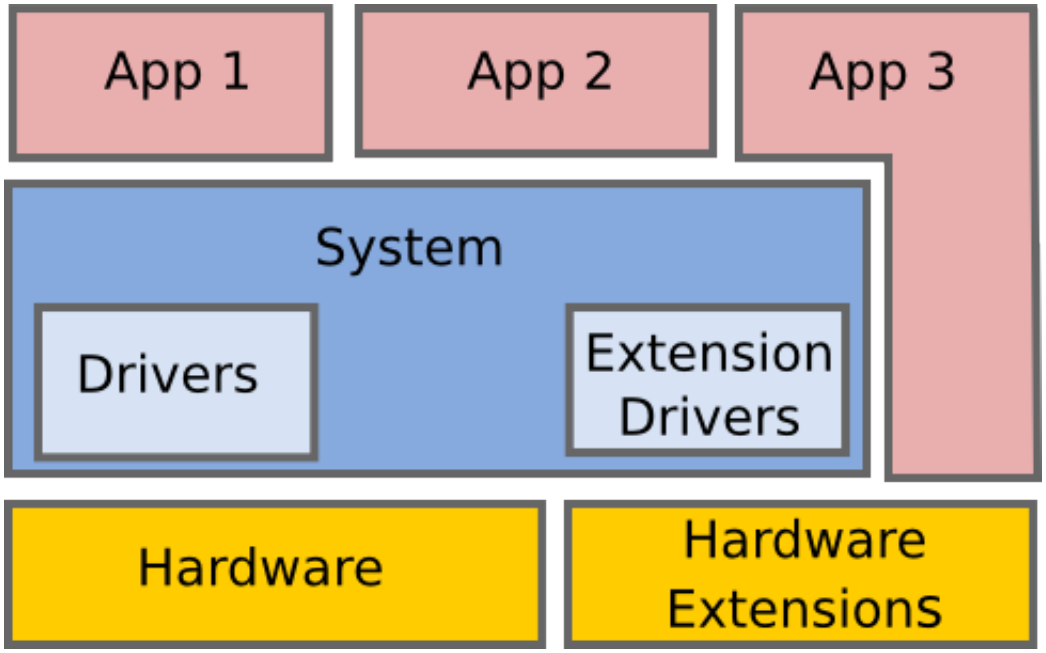


ScatterWeb²

- Small operation system for the MSB-430
- Event driven
- Current status: no (preemptive) multitasking
- Microcontroller in Low Power Mode (LPM) most times (99%)
- Kernel functions in libscatterweb used by user application



ScatterWeb² Architecture



Interrupt Handling

1. HW interrupt, e.g. voltage changed at input pin
2. CPU interrupts current program (if running)
3. registered ISR in IVT gets called
4. ISR sets flag (depending on interrupt source)
5. ISR wakes up CPU
6. superloop detects set flag
7. registered event handler gets called ⇒ event handled
   - in kernel and/or
   - in user application
8. CPU reenters low power mode

Debugging

- Limited
- Usually via JTAG interface (but doesn't work this well with MSPGCC)
- Write messages to serial out:

```
LOG_LOW( "PONG from %u", args->netheader->from );
```

- Configure MSB-430 via terminal command:

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

- Local:

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

- Remote - Send messages via transceiver:

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

Terminal Command

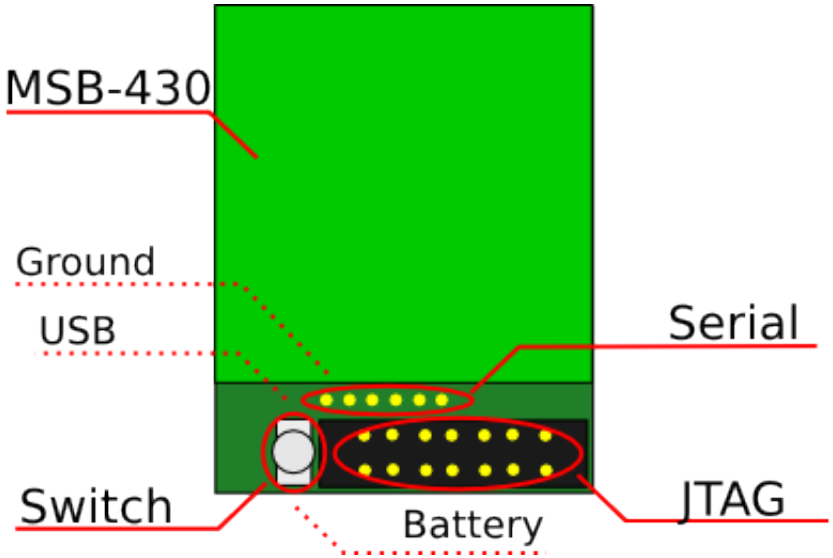
```
#define COMMAND_HEADER(_cmd, _flags, _args, _ptr ) \
  extern void cmd_##_cmd (const cmd_args_t* _args ); \
  const uint8_t cmd_name_##_cmd [] = #_cmd; \
  __attribute__((section(".commands"))) \
  const command_t cmd_##_cmd##_tableentry = { \
     (_flags | CMDFLAG_ASCII) , {cmd_##_cmd} , \
   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


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

Get your account registered as repository user during the break!

Let's get started

Connect The Hardware



Source Code

1)Open Cygwin Bash Shell

2)Download Source

```
$ cd SOME_DIR
$ mkdir SOME_NEW_DIR
$ svn co https://projects.mi.fu-berlin.de/svn/scatterweb/MSB-430/lab-branches/SS08/Team1 OR Team2
```

3)Make copy of [EMPTY] Application

```
$ cd Team?/Applications
$ svn cp [EMPTY] MY_NEW_APPLICATION
```

4)Compile Application

```
$ cd MY_NEW_APPLICATION
$ make
```

5)Flash to MSB-430

```
$ make flash
```

Terminal Emulator

1)Open a terminal emulator (e.g. TerraTerm, HyperTerminal)
- 115200 Baud
- Data Bits = 8
- Parity = none
- Stop Bits = 1
- Flow Control = None
- Send line ends with line feed (\r\n)
- Local Echo = on

2)Reset device (use the hardware switch)

3)Toggle red LED

```
$ slr 1
$ slr 0
```

Terminal Command

1)Open `src/ScatterWeb.Process.c`

2)Write a new terminal commando

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

3)Output number of called times

```
uint8 counter = 0;
COMMAND(HelloMSB, CMDFLAG_SERIAL, cmdargs) {
   printf(“Hello User - called %u times\r\n”, counter);
}
```

Terminal Command continued

1)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);
}
```

Timer

1)Call a function every 5s

2)Initial call

```
void callMe();
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);
}
```

3)Periodic call

```
void callMe() {
    print(...);
    Timers_add(5*1024, callMe, 0xFFFF);
}
```

Sending Data Packets/Frames

1)Prepare arguments

```
netpacket_send_args_t npsargs;
Net_sendArgsInit(&npsargs);
npsargs.netheader.to = node;
npsargs.netheader.type = USERDEFINED_PACKET;
npsargs.netheader.flags = 0;
```

2)Send packet

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

Receiving Data Packets/Frames

1)Look inside Process_radioHandler()

```
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;
}
```

Reading Sensor Values

Sensor values can be measured/read at any time. We use the accelerometer in this example. The access of different sensors can differ. No uniform API is available.

- The general read function:

```
sdata_t Data_read(uint8_t channel);
```

- Read acceleration sensor values:

```
sdata_t d = Data_read(1); // channel 1 == accelerometer
```

Exercise: Develop a terminal command to read the acceleration sensor values every 0.5s and write them to the serial port.

Exercise: The same as above, but this time send and display the values on a remote sensor node.