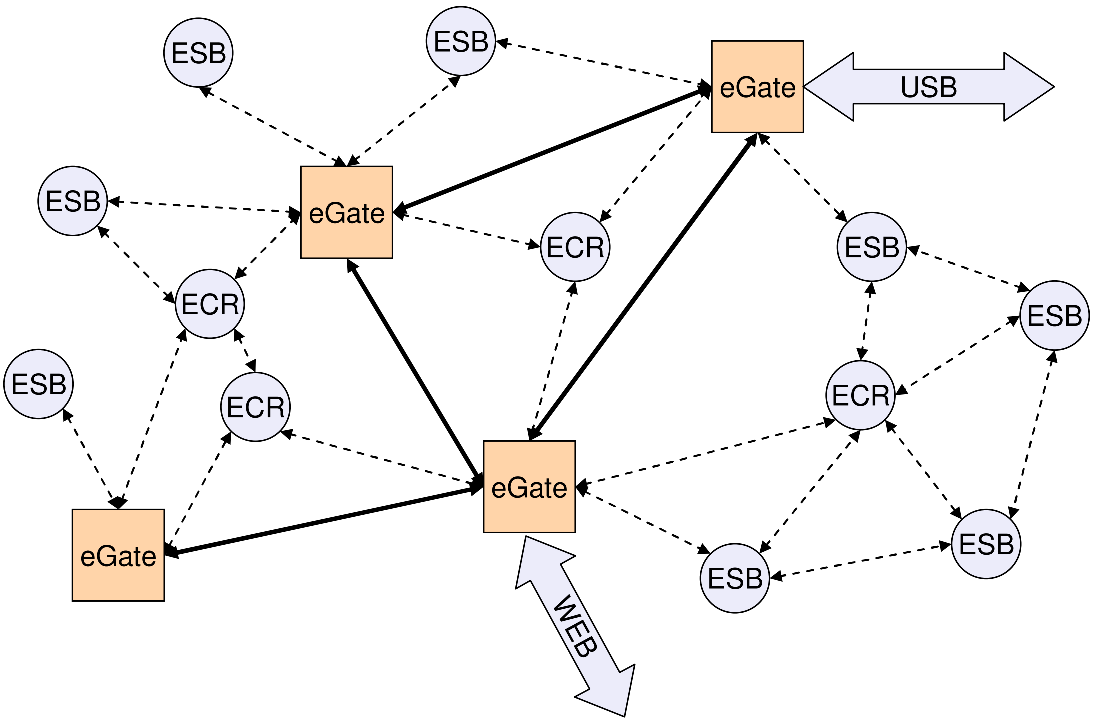
User Guide
(for ScatterWeb)

A platform for teaching & prototyping
wireless sensor networks



scatterweb.mi.fu-berlin.de

Computer Systems & Telematics

Freie Universität Berlin, Germany

Content

1 Platforms 5

1.1 Embedded Chip Radio (ECR) 5
1.2 Embedded Sensor Board (ESB) 6
1.3 Embedded EYE (EYE) 7
1.4 eGate/USB 8
1.5 eGate/WEB 9

2 Radio communiction 10

3 ScatterWeb.API 12

3.1 ScatterWeb.Comm 16
3.2 ScatterWeb.Configuration 19
3.3 ScatterWeb.Data 21
3.4 ScatterWeb.IO 33
3.5 ScatterWeb.Messaging 35
3.6 ScatterWeb.Net 38
3.7 ScatterWeb.String 43
3.8 ScatterWeb.Synchronisation 46
3.9 ScatterWeb.System 48
3.10 ScatterWeb.Threading 56
3.11 ScatterWeb.Time 64
3.12 ScatterWeb.Timers 67

4 Usage 70

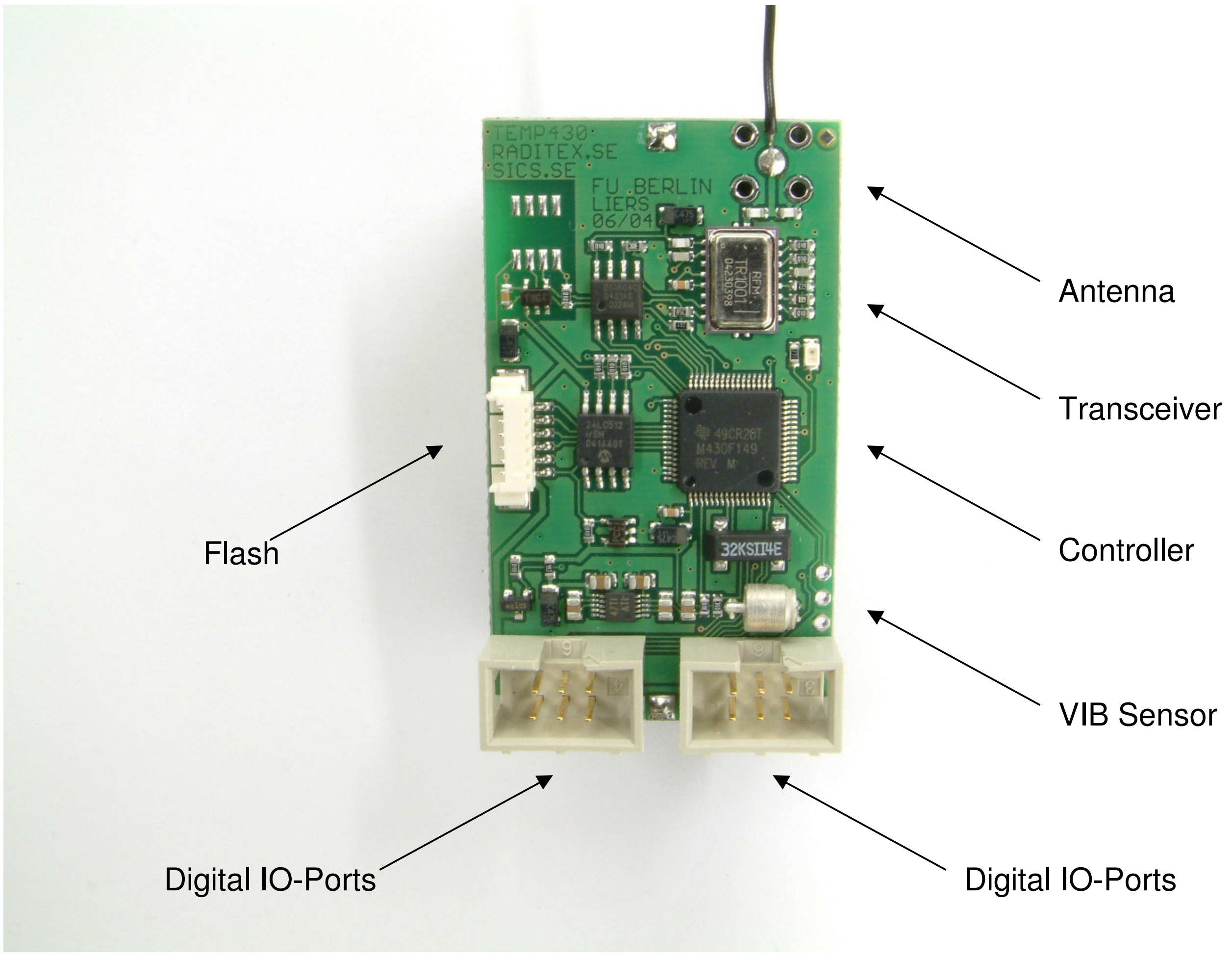
4.1 The [EMPTY] application 70
4.2 How to install 73
4.3 How to use 74
4.4 How to compile 75
4.5 How to flash 76

Appendix A – Terminal Command Listing 77

Appendix B – Terminal Command Availability 81

1 Platforms

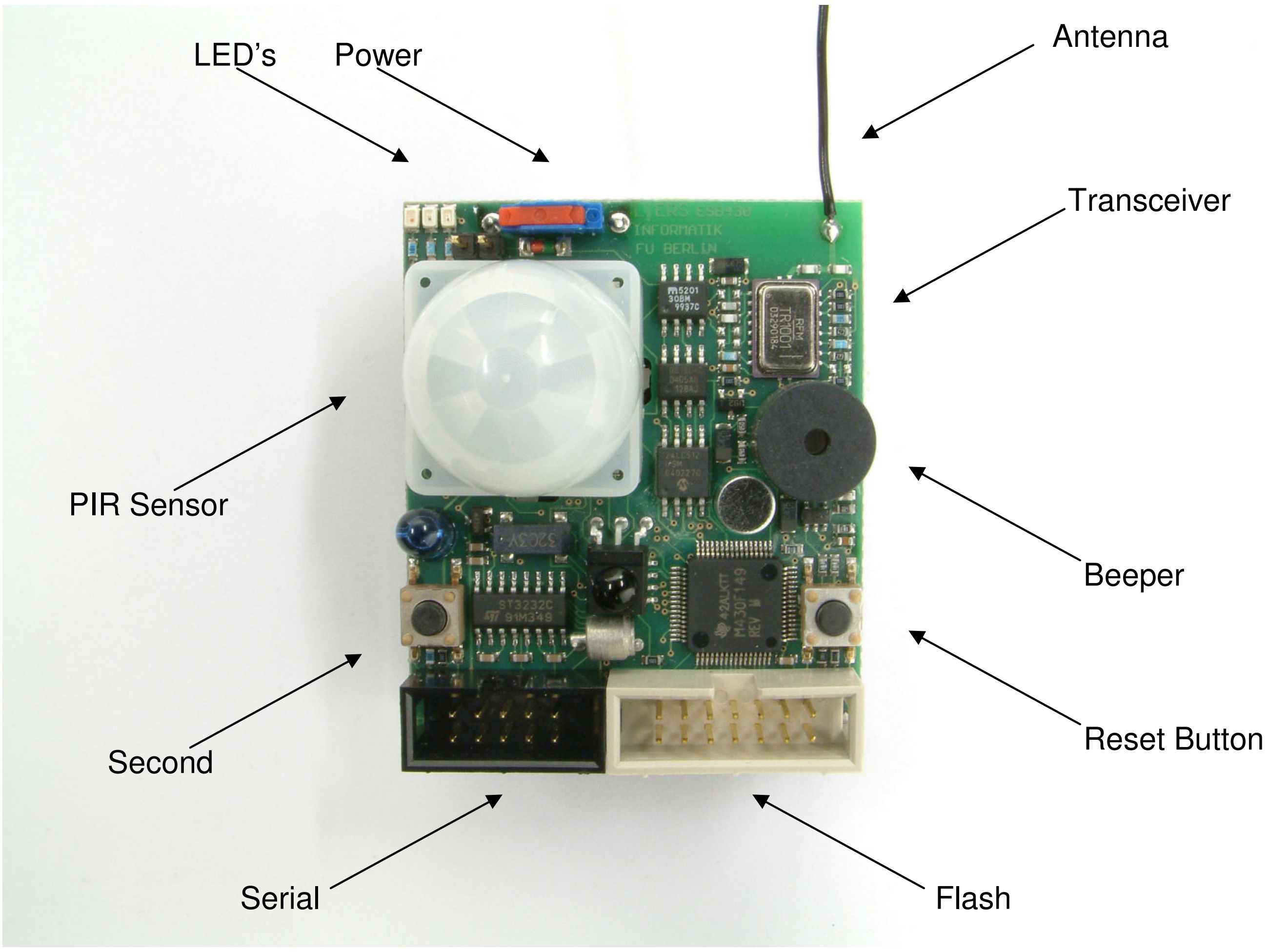
1.1 Embedded Chip Radio (ECR)



The ECR has additionally a Temperature Sensor, a red LED and a 64 Kbytes EEPROM.

Notice: You need an appropriate flash cable (obtainable on order) and the JTAG (also obtainable on order) to flash the ECR.

1.2 Embedded Sensor Board (ESB)



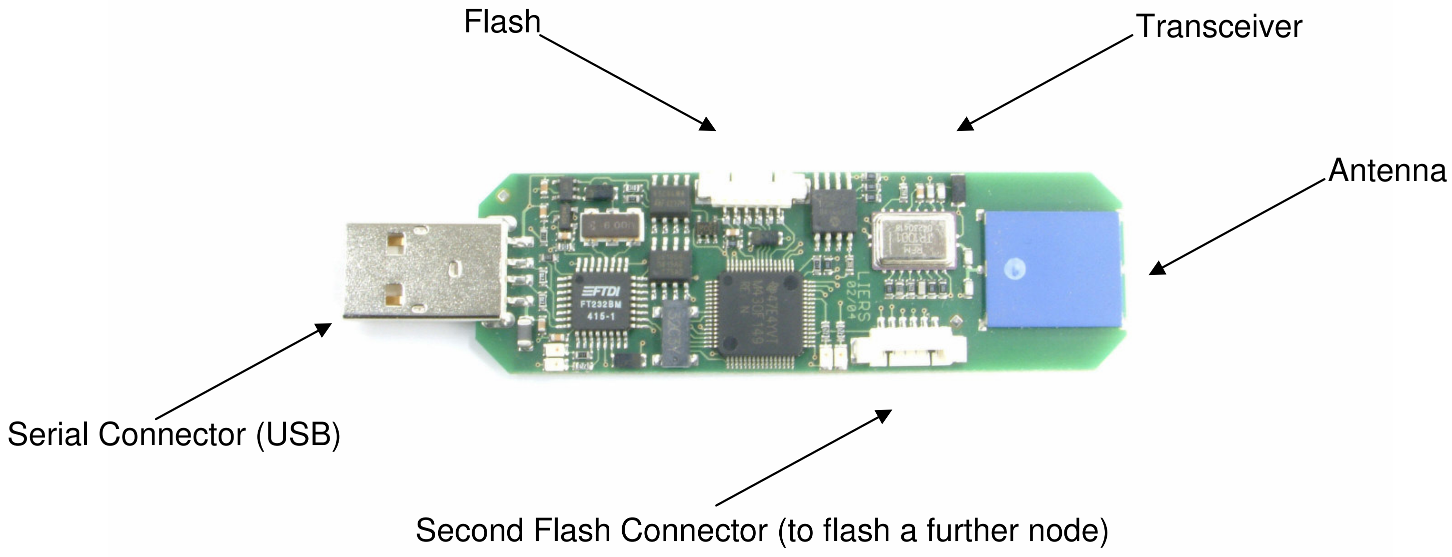
The ESB has additionally a Temperature Sensor, a VIB Sensor, IR to send and to receive, a Microphone and a 64 Kbytes EEPROM. The ESB supports the RC5-infrared-standard.

Notice: You need the JTAG (obtainable on order) to flash the ESB and an appropriate serial cable (also obtainable on order) to interconnect the ESB and the PC/PDA.

1.3 Embedded EYE (EYE)

The camera module has a serial connector to connect it to the ESB.

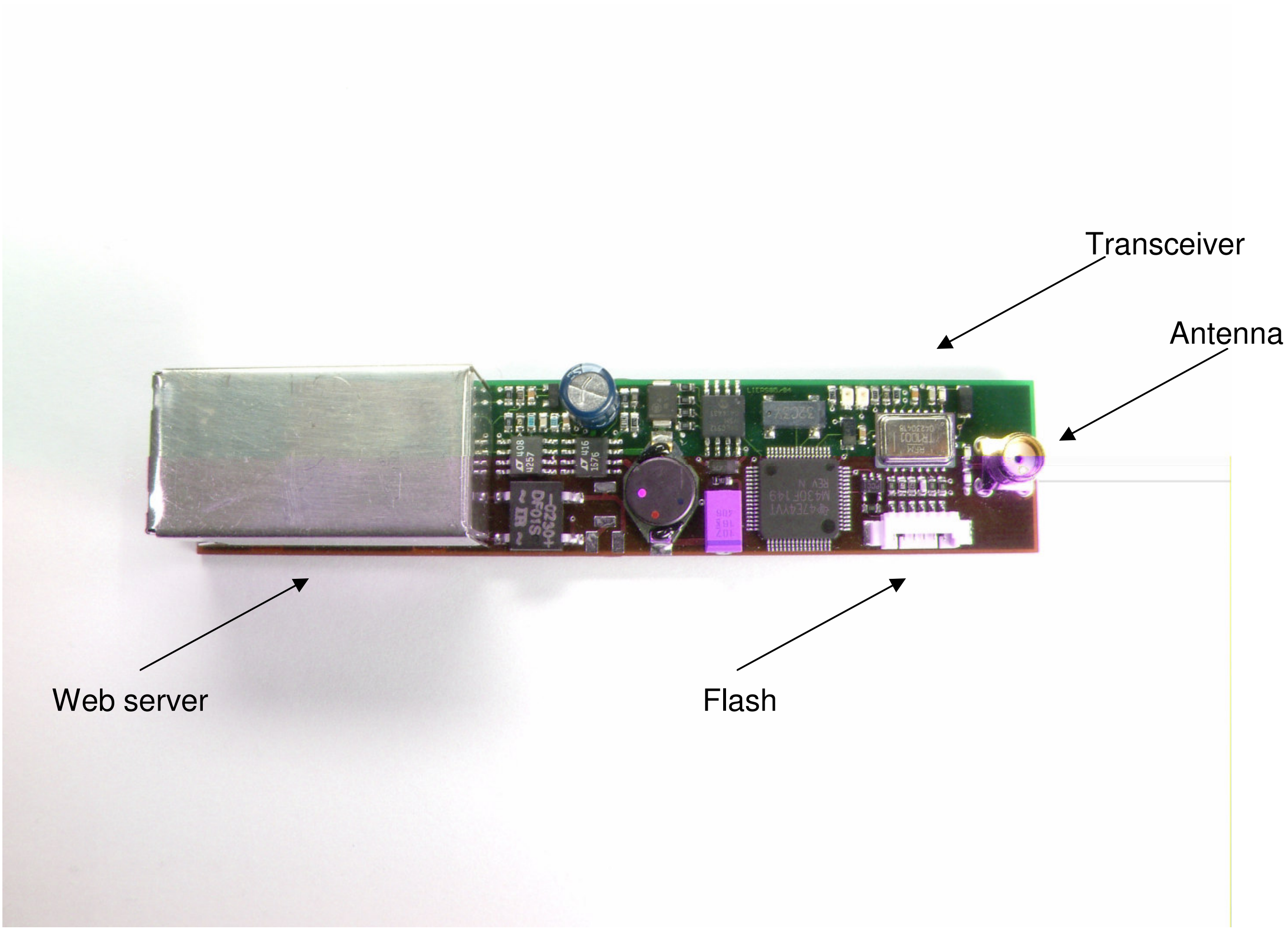
1.4 eGate/USB



The eGate/USB has additionally a red, a yellow and a green LED and a 64 Kbytes EEPROM.

Notice: You need an appropriate flash cable (obtainable on order) and the JTAG (also obtainable on order) to flash the eGate/USB.

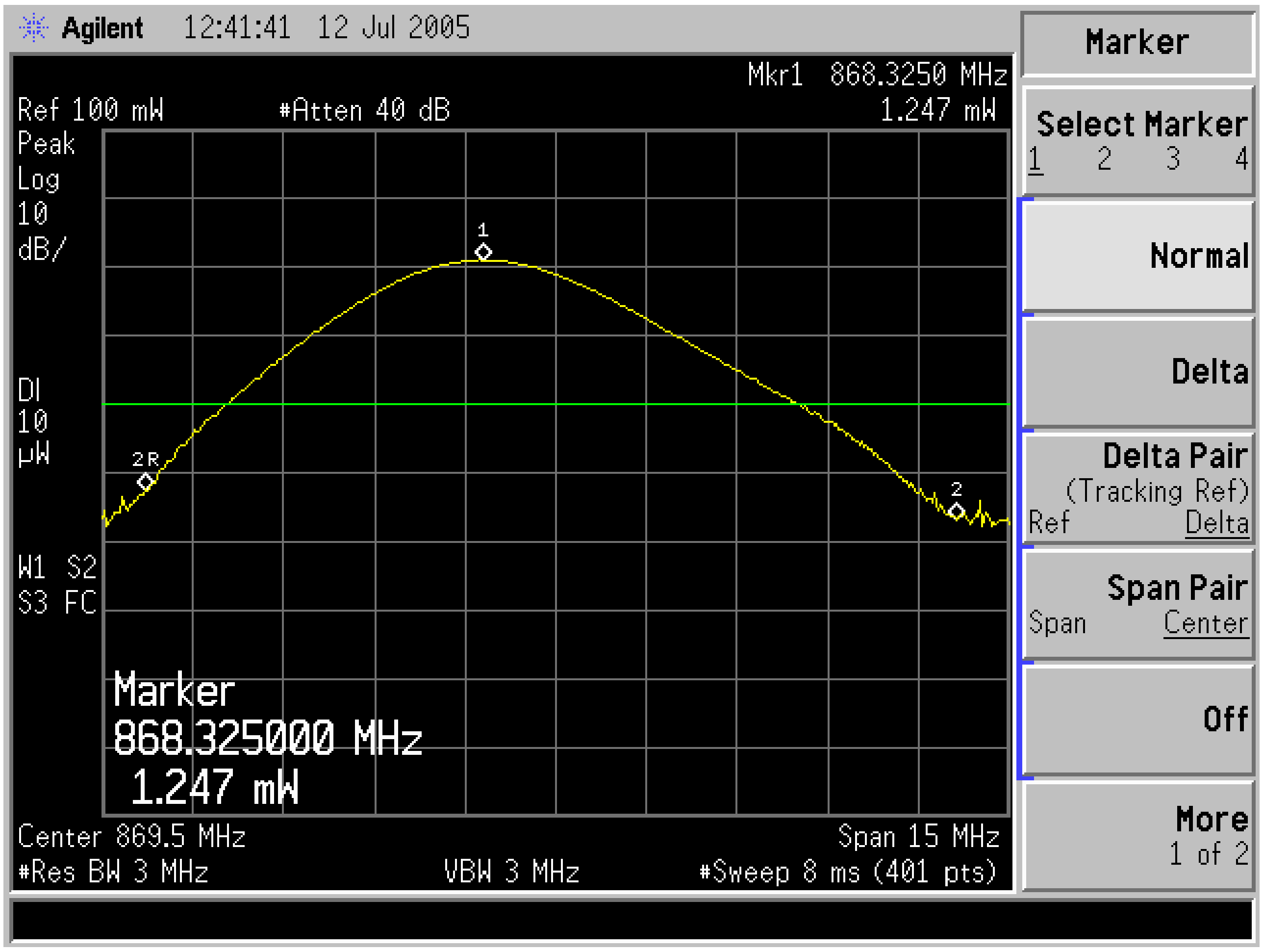
1.5 eGate/WEB

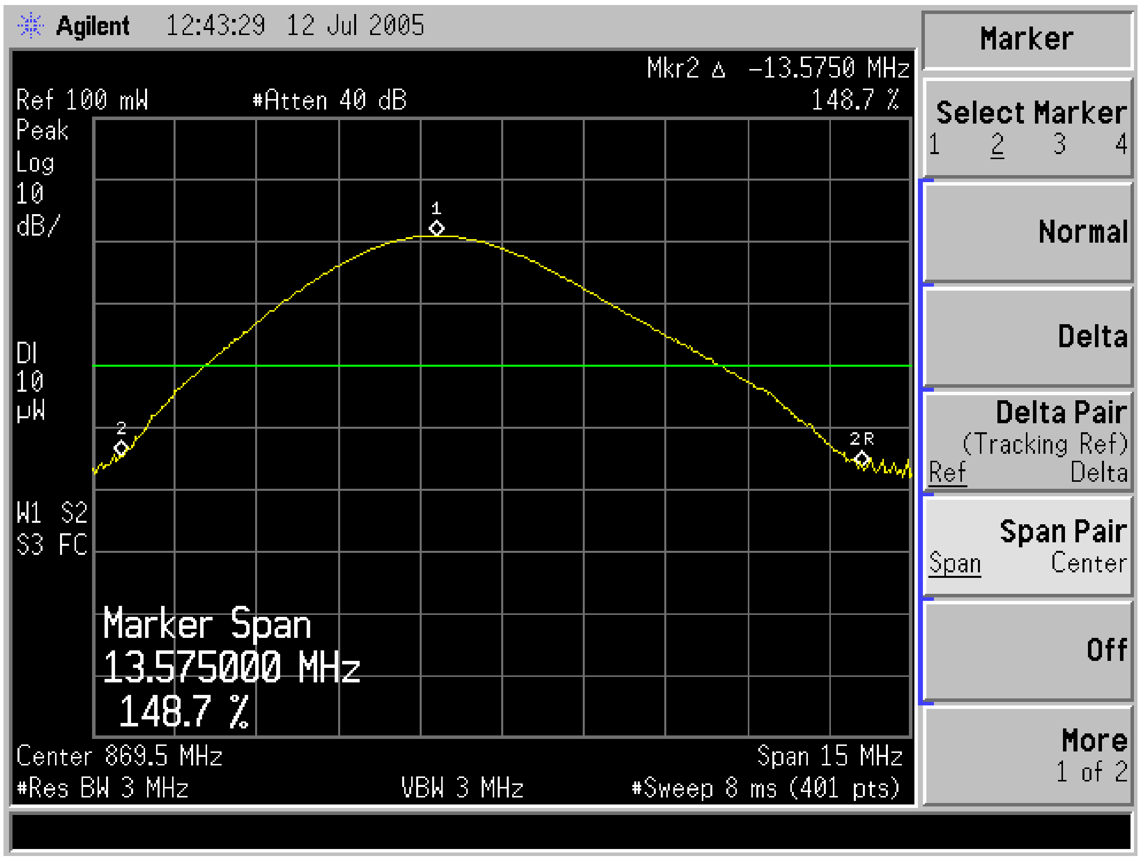


The eGate/WEB has additionally a red, a yellow and a green LED and a 64 Kbytes EEPROM.

Notice: You need an appropriate flash cable (obtainable on order) and the JTAG (also obtainable on order) to flash the eGate/WEB.

2 Radio communiction




Agilent 12:43:29 12 Jul 2005

Mkr2 Δ -13.5750 MHz
148.7 %

Ref 100 mW #Atten 40 dB

Peak
Log
10
dB/

Dl
10
µW

W1 S2
S3 FC

Marker Span
13.575000 MHz
148.7 %

Center 869.5 MHz Span 15 MHz
#Res BW 3 MHz VBW 3 MHz #Sweep 8 ms (401 pts)

3 ScatterWeb.API

As the name already implies, sensor networks need sensors, typically lots of sensors, to collect data from their environment. All sensor nodes may form an ad-hoc network with some nodes acting as data sources, some as relays, and some as data sinks collecting data. Nodes may act in all three roles at the same time. The ESBs developed by the CST Group forms a sound basis for research in sensor networks, the development of university level courses in CS/EE based on real hardware, and prototyping of sensor net applications. The ESBs are part of the ScatterWeb approach comprising a whole family of embedded web servers, sensors and actors. Typical communication scenarios of sensor networks based on the ESBs are:

- ESBs communicate via the serial port with a standard PC for application development.
- ESBs communicate with mobile phones via the serial port to connect to wide-area mobile phone networks. This enables remote configuration of ESBs via short messages (SMS) as well as reception of sensor data on arbitrary mobile phones world-wide.
- ESBs communicate via their radio interface with other ESBs or an embedded web server (EWS) to form a truly embedded, highly flexible sensor network solution.

Main components of the ESB are the sensors, a micro controller for processing of all data, and a transceiver handling communication with other nodes. Software for the following features is already available for the ESB 430:

- Configuration and analysis of all sensor data: vibration, noise, movement, temperature, infrared, (luminosity).
- Control of peripherals: timer, transceiver, serial port, EEPROM, IR-send/receive, LEDs, beeper, switches.
- Sending and receiving of data packets via the radio interface, transmission of sensor data, communication with other systems using the same radio interface (ESBs, ESWs).
- Sending and receiving of short messages (SMS) via connected mobile phone (serial port). Bluetooth connection available upon request (standard for the EWS). Just send the ESB an SMS triggering temperature monitoring every Tuesday morning from 7:00 a.m. until 11:43 a.m., if and only if there is movement in the room but no light - it's possible with a single SMS!
- Sending (e.g. to control home entertainment devices) and receiving (e.g. from conventional remote controls) of RC5 packets via infrared.

- Periodic polling of sensor data and (depending on settings) automatic transmission via the radio interface, the serial port (terminal or mobile phone).
- Simple and easy configuration and control of the ESB via terminal commands over the serial interface or radio.

Software

The software of the sensor nodes was first separated into two parts (firmware & application) to provide in-field programming via radio. This has recently changed because that reprogramming strategy created several problems. Now the whole thing (except the reprogramming code, which should not change anymore, see progamming.c) is reprogrammable. Anyway the separation between firmware and application is kept because it proved good as hardware abstraction layer and to provide some OS-like concepts. The Firmware provides abstract functions to interface and use the hardware, contains the main execution loop + interrupts and provides some OS-like concepts. The firmware supports all ScatterWeb/MSP430 devices: ESB (and EYE), eGate, ECR. The application (userapp, program) is built on top of the firmware.

Directory tree

- Applications:	Contains application directories with PSPad project files.
- System:	Contains firmware sources and binaries.

Major Changes

- Firmware and userapp are not that strictly separated anymore. This made reprogramming easier and all restrictions from the old firmware <-> userapp separation are gone. Reprogramming code is now in programming.h, flasher.h contains the radio transfer of code images.
- For simplicity all 3 firmwares from ESB, EYE, eGate and ECR have been integrated into one with the use of #ifdef at some hardware dependent places. When compiling the command line one must always define one of: ESB, eGate, ECR, EYE. This is done in the makefiles with the variable DEVICE. The makefiles are now taking every source file they see in the directories and don't need to be changed for new/removed files anymore (Look into the makefiles and at new directory structure).
- Radio software has changed. Sending is now also done interrupt driven and comes with a ring buffer to store several packets to send. Additional new features are crc16 checksums and automatic acknowledgements (except broadcasts) with feedback to the sender of every packet.

- The terminal software has been moved to the firmware as many commands between devices were the same. Command creation has changed and most commands are executable from a remote device (with reply). Commands can now be placed where they are needed, Macros and a special section in flash build a table to automatically search all commands (See ScatterWeb.Messaging.h).
- Light sensor software has been removed due to complex handling. Timer slots are also needed for constant dco checking & adjusting (ScatterWeb.Timers.h) and a software RTC (ScatterWeb.Time.h) as the RTC is mostly abandoned because of its big current usage, it's only used as an well calibrated temperature sensor (ScatterWeb.System.h).
- ScatterWeb.Time.h represents the new software RTC (see currentTime). ScatterWeb.System.h has been completely cleared; only the function for reading the temperature and ensuring RTC 32 kHz output at initialization are left. fiveMsTimer is replaced through currentTime or use of timers (ScatterWeb.Timers.h), its still left for internal timer use.
- ScatterWeb.Timers.h module has been added, no need to poll for an elapsed time anymore. ScatterWeb.System.h includes DCO checking, so DCO is recalibrated the whole time, not at startup only!
- Other modules have been more separated (ScatterWeb.Configuration.h (was in ScatterWeb.IO.h), programming.h (was in flasher.h)).
- The watchdog has been rewritten. It's now correct and possible to see where a reset came from. Additionally a feature has been added to watch stack growth and detect a stack overflow. For bad conditions like that a panic function has been added. The firmware can disable the application if a panic occurred.
- An abstract printf like function has been implemented for string outputting: String_abstractWrite. This is used by the functions String_write (to write a string to memory), Comm_print (to write normal output to the serial port) and Comm_log (to write debug output to the serial port). No other methods should be used for writing to the serial port, sendRS232 is not public anymore. This is because serial port output is now controlled via Comm_logLevel. It allows several log levels; some debug bits for special debugging and total disabling of all output (see ScatterWeb.Comm.h).
- A cooperative scheduler exists to create simple task functions: ScatterWeb.Threading.h.
- The main super loop goes to sleep when everything is processed, all things that need to be handled are interrupt-driven (see main). Every interrupt which needs processing from the super loop wakes the processor up with WAKEUPLPM1, which also sets a flag in runModule. When all flags in runModule are reset (after processing from super loop tasks) the loop goes to LPM1 again. If a function which needs calls as often as possible is added to the super loop you have to comment out the lines about sleeping!
- Typedefs for UINT8, UINT16, UINT32 have been introduced because it is better style and much shorter.

Important things to know:

- All routines should return fast. If a routine occupies the processor for a long time (for example outputting a long text which works with the blocking sendRS232Char() function) it should call System_startWatchdog() repeatedly (normally done in main()) to avoid a watchdog reset. If a watchdog reset occurs the firmware will disable the programm if PANIC_DISABLES_USERAPP is set.
- You don't need to change the makefiles anymore. Makefiles in firmware and application automatically compile every .c file in the src directories and look into every .c and .h file for changes.
- No initialization of global values is done. You can write "UINT16 var = 49;" but var will never be initialized, do initialization in code! (Initialization may work again if the standard startup routine is used.)
- Don't use dynamic memory allocation (malloc/free)! Dynamic memory allocation bears the following problems on MCUS:

    - Much harder to debug.
    - Unknown behavior at compile time.
    - Memory fragmentation and no way (missing MMU) to defragment it.
    - No memory protection between stack and heap which grow towards each other.
    - Additionally current msp430-gcc malloc implementation seems to be buggy.

Sometimes an embedded engineer said: "Think small on MCUs. If you are planning to use dynamic memory allocation, you are thinking much too BIG."

- If weird things happen look into linker map "out.map"
- Sometimes compilation fails or hangs with existing objects files from another machine.

Do a "make clean" or simply delete all object (*.o) files in firmware and userapp directories in this situation!

3.1 ScatterWeb.Comm

```
\ScatterWeb 2.x\System\src\ScatterWeb.Comm.h:
#ifndef ScatterWeb_Comm
#define ScatterWeb_Comm
  #define BPS_2400       (0x01)
  #define BPS_14400      (0x02)
  #define BPS_19200      (0x03)
  #define BPS_38400      (0x04)
  #define BPS_57600      (0x05)
  #define BPS_115200     (0x06)
  UINT8 Comm_logLevel;
  bool Comm_init(UINT8 bps);
  bool Comm_print(const char *fmt, ...);
  bool Comm_log(UINT8 level, const char* fmt, ...);
  bool Comm_on();
  bool Comm_off();
  bool Comm_registerByteLevel(fp_char_t fp);
  bool Comm_clear();
#endif
```

The ScatterWeb.Comm represents the standard output for applications. The init function for the UART Comm_init() provides setup for all standard baud rates: BPS_2400, BPS_14400, BPS_19200, BPS_38400, BPS_57600, and BPS_115200. All UART communication should be done with Comm_print() or Comm_log() as the serial output with this functions is controlled via Com_logLevel and struct Configuration.boot_logLevel in ScatterWeb.Configuration. All sending functions are blocking. Receiving is handled by an interrupt and you can choose byte-access or line-access.

- byte-access: To gain access to the byte level, register a function that expect to get pass an UINT8 as parameter (see Comm_registerByteLevel(fp_char_t fp)).
- line-access: To gain access to the line level, register a function that expect to get pass an void* as parameter (see C_SERIAL and System_registerCallback(UINT8 type, void(*fp_t)())). If the line is handled the receiving function should call Comm_clear().

```
bool Comm_init(UINT8 bps);
```

Function:	Initializes the serial
Parameter:	bps - the bps to be set
Return value:	<code>True</code> always.

```
bool Comm_print(const char *fmt, ...);
```

Function:	Sends bytes of data
Parameter:	fmt - function
	... - list of parameters
Return value:	<code>True</code> if serial is on;
	<code>False</code> otherwise.

```
bool Comm_log(UINT8 level, const char* fmt, ...);
```

Function:	Sends bytes of log data
Parameter:	level - log level
	fmt - function
	... - list of parameters
Return value:	<code>True</code> if serial is on;
	<code>False</code> otherwise.

```
bool Comm_on();
```

Function:	Switches the serial on
Parameter:	None
Return value:	<code>True</code> always.

```
bool Comm_off();
```

Function:	Switches the serial off
Parameter:	None
Return value:	<code>True</code> always.

```
bool Comm_registerByteLevel(fp_char_t fp);
```

Function:	Registers byte level and will be set by Comm_init(UINT8 bps)
Parameter:	Fp -pointer of function
Return value:	<code>True</code> if ECR or ESB defined
	<code>False</code> if eGate defined.

```
bool Comm_clear();
```

Function:	This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.
Parameter:	-
Return value:	-

3.2 ScatterWeb.Configuration

```
\ScatterWeb 2.x\System\src\ScatterWeb.Configuration.h:
#ifndef ScatterWeb_Configuration
#define ScatterWeb_Configuration
  #define EADR_CONFIG              (0x0F00)
  #define DEBUG_RADIO              (0x80)
  #define DEBUG_DCO                (0x40)
  #define DEBUG_SYNC               (0x20)
  #define NO                       (0x01)
  #define LOW                      (0x02)
  #define MEDIUM                   (0x03)
  #define HIGH                     (0x04)
  #define VERBOSE                  (0x05)
  #define FF_PROGRAMMABLE          (0x01)
  #define FF_DCOCHECKER            (0x02)
  typedef struct {
     UINT16 id;
     UINT16 rxReceiveLimit;
     UINT8 transmitPower;
     UINT8 boot_logLevel;
     UINT8 firmwareFlags;
  } config_t;
  extern config_t Configuration;
  bool Configuration_init();
  bool Configuration_save();
  bool Configuration_print();
#endif
```

The ScatterWeb.Configuration provides functions that allow you to programmatically access configuration settings. The configuration consists of the struct config_t defined in ScatterWeb.Configuration.h. The variable Configuration is of that type and represents the current config in RAM and can be accessed directly every time. Changes to Configuration will only be made permanent (power-off & resets) with a call to Configuration_save(), which copies the content of the Configuration to EEPROM[EADR_CONFIG]. Initial loading from EEPROM[EADR_CONFIG] is done by Configuration_init() and has to be called on initialization in main().

First time configuration

Uncomment the following definition of firstTimeConfig and it will get laid into Infomem (0x1000-0x1100, see .hex-File). Before Configuration_init() loads the regular config from EEPROM, it checks if the first int (which corresponds to the id) in Infomem is not 0xFFFF. If it is not, it copies the configuration from Infomem to EEPROM (clearing Infomem afterwards and loading the newly written data from EEPROM to Configuration).

```
bool Configuration_init();
```

Function: Initializes configuration settings for the configuration section

Parameter: None

Return value: <code>True</code> always.

```
bool Configuration_save();
```

Function: Saves configuration settings for the configuration section

Parameter: None

Return value: <code>True</code> always.

```
bool Configuration_print();
```

Function: Prints configuration settings for the configuration section

Parameter: None

Return value: <code>True</code> always.

3.3 ScatterWeb.Data

\ScatterWeb 2.x\System\src\ScatterWeb.Data.h:

```
#ifndef ScatterWeb_Data
#define ScatterWeb_Data
  #include "ScatterWeb.Time.h"
  #define REDSTATE            (Data_redLed())
  #define YELLOWSTATE         (Data_yellowLed())
  #define GREENSTATE          (Data_greenLed())
  #define SENSOR_MOVEMENT     (0x08)
  #define SENSOR_VIBRATION    (0x10)
  #define SENSOR_BUTTON       (0x80)
  #define SENSOR_MICROPHONE   (0x01)
  #define SENSOR_TEMPERATURE  (0x02)
  #define SENSOR_RC5          (0x04)
  #define SENSOR_VOLTAGE      (0x40)
  #define SENSOR_LIGHT        (0x20)
  typedef struct {
     UINT16 id;
     UINT8 sensor;
     time_t timeStamp;
     UINT16 value;
  } sdata_t;
  extern UINT8 Data_sensorFlags;
  bool Data_init();
  bool Data_batteryOn(UINT16 interval, UINT16 thresh);
  bool Data_batteryOff();
  bool Data_beeperOn();
  bool Data_beeperOff();
  bool Data_beeperToggle();
  bool Data_buttonOn(UINT16 delay);
  bool Data_buttonOff();
  bool Data_greenOn();
  bool Data_greenOff();
  bool Data_greenToggle();
  bool Data_redOn();
  bool Data_redOff();
  bool Data_redToggle();
  bool Data_yellowOn();
  bool Data_yellowOff();
  bool Data_yellowToggle();
  bool Data_microphoneOn(UINT16 delay, UINT16 diff);
  bool Data_microphoneOff();
  bool Data_movementOn(UINT16 delay);
  bool Data_movementOff();
  bool Data_temperatureOn
      (UINT16 interval, UINT16 range, UINT16 diff, UINT16 spd);
  bool Data_temperatureOff();
  bool Data_vibrationOn(UINT16 delay);
  bool Data_vibrationOff();
  bool Data_lightOn(UINT16 delay, UINT16 thresh);
  bool Data_lightOff();
  bool Data_greenLed();
  bool Data_redLed();
  bool Data_yellowLed();
  bool Data_RC5SendPHI(int data);
  bool Data_RC5Enable();
  bool Data_RC5Disable();
  bool Data_RC5SendNEC(UINT16 data1, UINT16 data2);
  bool Data_sensorHandler();
  void Data_RC5ReceiveHandler();
  void Data_RC5EventHandler();
#endif
```

The ScatterWeb.Data consists mostly of the functions that constitute the sensors architecture. The sensors architecture enables you to build components that efficiently manage data from multiple data sources. It provides the tools to initialize, enable, and disable data in this multiple tier system. To get sensor events you must set a callback function in System_callbacks[C_SENSOR], for disabling that set System_callbacks[C_SENSOR] = NULL. Each sensor is separately enabled and disabled with its enable/disable function. The enable function takes some sensor specific arguments. The first argument is the interval (or debounce delay) for reading the sensor internally, it can also be seen as a reaction time (for example for Vib: If Vibration has been signaled how long the sensor must stay silent to signal NoVibration?).

For temperature 4 arguments are needed to specify a funnel function: Data_temperatureOn(Interval, StartRange, Diff, StepsPerDiff):

- Interval - How often is the temperature read in systems ticks (~1ms).
- StartRange - How much must the temperature differ from the last value to get signaled.
- Diff - How much the Range is decreased every DiffStep measures.
- StepsPerDiff - How much measures until Range is decremented with Diff.

Attention: Temperature data is in fixed point 2s complement format!!!

This approach also includes simple standard evaluation methods:

- Periodic Reading: To signal every x ms do Data_temperatureOn(x, 0, 0, 0);
- On change Reading: To signal only on temperature changes greater than x (temperature is read every y ms) do Data_temperatureOn(y, x, 0, 0):
- Periodic & On Change combined: To signal on every change greater than x or after a time y*z has passed (temperature is read every z ms) do Data_temperatureOn(z, x, x, y).

Examples:

- Data_temperatureOn(5120, 0, 0, 0) - Every 5 seconds.
- Data_temperatureOn(2048, 0x0080, 0, 0) - Only on change greater 1℃ (temperature is read every 2 seconds).
- Data_temperatureOn(6144, 0x0080, 0x0080, 20)- Every 20*6s=2min or if changed more than 1℃ (temperature is read every 6 seconds).
- Data_temperatureOn(6144, 0x0200, 0x0040, 10) - Funnel function: Measure every 6 seconds and decrease range from 2℃ for 0,5℃ every 10*6s=1min (temperature is read every 6 seconds).

Power for sensors (and RS232 voltage converter) must be enabled, see System_powerOn() / System_powerOff(). The IR Receiver is handled in file ScatterWeb.Data.c. A good example for using the sensors is ScatterWeb.Event.c in [EMPTY]\ESB project. It's controlling the beeper. It's receiving RC5 via IR Receiving Diode.

RC5: 1780us bit length (Manchester encoded, so half bit length of 890us is important). Transferred packet (2 start + toggle bit + 5 address bits + 6 command bits)):

| S | S | T | A4 | A3 | A2 | A1 | A0 | C5 | C4 | C3 | C2 | C1 | C0 |

irdata format:

| ? | ? | error | newData | T | A4 | A3 | A2 | A1 | A0 | C5 | C4 | C3 | C2 | C1 | C0 |

Some common addresses and commands:

Address:	Device:	Command:	
0	TV1	0...9	Numbers 0...9 (channel select)
1	TV2	12	Standby
5	VCR1	16	Master Volume +
6	VCR2	17	Master Volume -
17	Tuner	18	Brightness +
18	Audio Tape	19	Brightness -
20	CD Player	50	Fast rewind
		52	Fast run forward
		53	Play
		54	Stop
		55	Recording

It supports methods for sending with IR diode. These methods are timed to the system clock (4505600 MHz) and disable all interrupts and so will halt the system timer for some milliseconds. Radio packet transmission & reception are also disrupted.

```
bool Data_init();
```

Function:	Initializes the sensors
Parameter:	None
Return value:	<code>True</code> if sensors available;
	<code>False</code> otherwise.

```
bool Data_batteryOn(UINT16 interval, UINT16 thresh);
```

Function:	Enables the battery sensor
Parameter:	interval - interval of measurement
	thresh - thresh of measurement
Return value:	<code>True</code> if battery sensor available;
	<code>False</code> otherwise.

```
bool Data_batteryOff();
```

Function:	Disables the battery sensor
Parameter:	None
Return value:	<code>True</code> if battery sensor available;
	<code>False</code> otherwise.

```
bool Data_beeperOn();
```

Function:	Enables the beeper
Parameter:	None
Return value:	<code>True</code> if beeper sensor available;
	<code>False</code> otherwise.

```
bool Data_beeperOff();
```

Function:	Disables the beeper
Parameter:	None
Return value:	<code>True</code> if beeper sensor available;
	<code>False</code> otherwise.

```
bool Data_beeperToggle();
```

Function:	Toggles the beeper
Parameter:	None
Return value:	<code>True</code> if beeper sensor available; <code>False</code> otherwise.

```
bool Data_buttonOn(UINT16 delay);
```

Function:	Enables the button
Parameter:	delay - delay of measurement
Return value:	<code>True</code> if button sensor available; <code>False</code> otherwise.

```
bool Data_buttonOff();
```

Function:	Disables the button
Parameter:	None
Return value:	<code>True</code> if button sensor available; <code>False</code> otherwise.

```
bool Data_greenOn();
```

Function:	Enables the green LED
Parameter:	None
Return value:	<code>True</code> if green led available; <code>False</code> otherwise.

```
bool Data_greenOff();
```

Function:	Disables the green LED
Parameter:	None
Return value:	<code>True</code> if green led available; <code>False</code> otherwise.

```
bool Data_greenToggle();
```

Function:	Toggles the green LED
Parameter:	None
Return value:	<code>True</code> if green led available; <code>False</code> otherwise.

```
bool Data_redOn();
```

Function:	Enables the red LED
Parameter:	None
Return value:	<code>True</code> if red led available; <code>False</code> otherwise.

```
bool Data_redOff();
```

Function:	Disables the red LED
Parameter:	None
Return value:	<code>True</code> if red led available; <code>False</code> otherwise.

```
bool Data_redToggle();
```

Function:	Toggles the red LED
Parameter:	None
Return value:	<code>True</code> if red led available; <code>False</code> otherwise.

```
bool Data_yellowOn();
```

Function:	Enables the yellow LED
Parameter:	None
Return value:	<code>True</code> if yellow led available; <code>False</code> otherwise.

```
bool Data_yellowOff();
```

Function:	Disables the yellow LED
Parameter:	None
Return value:	<code>True</code> if yellow led available;
	<code>False</code> otherwise.

```
bool Data_yellowToggle();
```

Function:	Toggles the yellow LED
Parameter:	None
Return value:	<code>True</code> if yellow led available;
	<code>False</code> otherwise.

```
bool Data_microphoneOn(UINT16 delay, UINT16 diff);
```

Function:	Enables the microphone sensor
Parameter:	delay - delay of measurement
	diff - diff of measurement
Return value:	<code>True</code> if microphone sensor available;
	<code>False</code> otherwise.

```
bool Data_microphoneOff();
```

Function:	Disables the microphone sensor
Parameter:	None
Return value:	<code>True</code> if microphone sensor available;
	<code>False</code> otherwise.

```
bool Data_movementOn(UINT16 delay);
```

Function:	Enables the movement sensor
Parameter:	delay - delay of measurement
Return value:	<code>True</code> if movement sensor available;
	<code>False</code> otherwise.

```
bool Data_movementOff();
```

Function:	Disables the movement sensor
Parameter:	None
Return value:	<code>True</code> if movement sensor available; <code>False</code> otherwise.

```
bool Data_temperatureOn(UINT16 interval, UINT16 range, UINT16 diff,
UINT16 spd);
```

Function:	Enables the temperature sensor
Parameter:	interval - interval of measurement
	range - range of measurement
	diff - diff of measurement
	spd - spd of measurement
Return value:	<code>True</code> if temperature sensor available; <code>False</code> otherwise.

```
bool Data_temperatureOff();
```

Function:	Disables the temperature sensor
Parameter:	None
Return value:	<code>True</code> if temperature sensor available; <code>False</code> otherwise.

```
bool Data_vibrationOn(UINT16 delay);
```

Function:	Enables the vibration sensor
Parameter:	delay - delay of measurement
Return value:	<code>True</code> if vibration sensor available; <code>False</code> otherwise.

```
bool Data_vibrationOff();
```

Function:	Disables the vibration sensor
Parameter:	None
Return value:	<code>True</code> if vibration sensor available; <code>False</code> otherwise.

```
bool Data_lightOn(UINT16 delay, UINT16 thresh);
```

Function:	Enables the light sensor
Parameter:	delay - delay of measurement
	thresh – threshold of measurement
Return value:	<code>True</code> if light sensor available;
	<code>False</code> otherwise.

```
bool Data_lightOff();
```

Function:	Disables the light sensor
Parameter:	None
Return value:	<code>True</code> if light sensor available;
	<code>False</code> otherwise.

```
bool Data_greenLed();
```

Function:	Returns the state of the green LED
Parameter:	None
Return value:	<code>True</code> if green Led is switched on;
	<code>False</code> otherwise.

```
bool Data_redLed();
```

Function:	Returns the state of the red LED
Parameter:	None
Return value:	<code>True</code> if red Led is switched on;
	<code>False</code> otherwise.

```
bool Data_yellowLed();
```

Function:	Returns the state of the yellow LED
Parameter:	None
Return value:	<code>True</code> if yellow Led is switched on;
	<code>False</code> otherwise.

```
bool Data_RC5SendPHI(int data);
```

Function:	Sends a RC5 PHI command
Parameter:	data - data to send
Return value:	<code>True</code> True if IR sensor available; <code>False</code> otherwise.

```
bool Data_RC5Enable();
```

Function:	Enables the RC5
Parameter:	None
Return value:	<code>True</code> True if IR sensor available; <code>False</code> otherwise.

```
bool Data_RC5Disable();
```

Function:	Disables the RC5
Parameter:	None
Return value:	<code>True</code> True if IR sensor available; <code>False</code> otherwise.

```
bool Data_RC5SendNEC(UINT16 data1, UINT16 data2);
```

Function:	Sends a RC5 NEC command
Parameter:	data1 - data1 to send data2 - data2 to send
Return value:	<code>True</code> True if IR sensor available; <code>False</code> otherwise.

```
bool Data_sensorHandler();
```

Function:	This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.
Parameter:	-
Return value:	-

```
void Data_RC5ReceiveHandler();
```

Function: This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.

Parameter: -

Return value: -

```
void Data_RC5EventHandler();
```

Function: This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.

Parameter: -

Return value: -

3.4 ScatterWeb.IO

```
\ScatterWeb 2.x\System\src\ScatterWeb.IO.h:
#ifndef ScatterWeb_IO
#define ScatterWeb_IO
  #define EEPROMADDRESS (0x00)
  #define EEPROMPAGEMASK (0x7F)
  bool IO_erase(UINT16 e_adr, UINT16 size);
  UINT8* IO_read(UINT16 e_adr, UINT8* r_adr, UINT16 size);
  bool IO_write(UINT16 e_adr, UINT8* r_adr, UINT16 size);
#endif
```

The ScatterWeb.IO contains functions that allow reading and writing to the EEPROM. Most important functions for writing/reading the EEPROM are IO_write() / IO_read(). The EEPROM offers 64k-memory (100000 write cycles) and is accessed with an I2C like bus. Currently use is storing configuration data (see ScatterWeb.Configuration.c) and temporary storage of a new software image before it gets flashed. (DEVICE ADDRESSING: Start | 1 0 1 0 A2 A1 A0 R/W | ...)

EEPROM memory map:

Region – Usage

- 0x0000 - 0x0CFF - Free for use.
- 0x0D00 - 0x0DFF - Application config (EADR_APPCONFIG).
- 0x0E00 - 0x0EFF - Used by flasher.c / EADR_FLASHERBUF: Free for use only if flasher_state==IDLE
- 0x0F00 - 0x0FFF - Never change something here. Used for configuration data (ScatterWeb.Configuration.c, EADR_CONFIG) and some reprogramming variables.
- 0x1000 - 0xFFFF - Programming image storage or other big data. Use according to EEPROM[EADR_IMAGESTATE].

```
bool IO_erase(UINT16 e_adr, UINT16 size);
```

Function:	Erase several bytes from EEPROM (means set them to 0xFF).
Parameter:	e_adr - start address of bytes to erase
	size - size of bytes to erase
Return value:	<code>True</code> if erase successful;
	<code>False</code> otherwise.

```
UINT8* IO_read(UINT16 e_adr, UINT8* r_adr, UINT16 size);
```

Function:	Reads size bytes from the EEPROM, uses sequential read (faster than readEEPROMByte).
Parameter:	e_adr - start address of bytes to read
	r_adr - array to store read bytes
	size - size of bytes to read
Return value:	UINT8* pointer of r_adr

```
bool IO_write(UINT16 e_adr, UINT8* r_adr, UINT16 size);
```

Function:	Writes size bytes to the EEPROM, uses sequential write (faster than writeEEPROMByte).
Parameter:	e_adr - start address of bytes to write
	r_adr - array of bytes to write
	size - size of bytes to write
Return value:	<code>True</code> if write successful;
	<code>False</code> otherwise.

3.5 ScatterWeb.Messaging

```
\ScatterWeb 2.x\System\src\ScatterWeb.Messaging.h:
#ifndef ScatterWeb_Messaging
#define ScatterWeb_Messaging
  #define SERIALONLY     (0x02)
  #define COMMAND(x, flags)  void term_##x (const UINT8* str); \
     __attribute__((section(".commands")))
     const command_t com_##x = { term_##x, #x, flags }; \
     void term_##x (const UINT8* str)
  typedef void(*fp_term)(const UINT8*);
  extern UINT8 term_reply[TERM_BUFSIZE];
  typedef struct command_struct {
     fp_term function;
     UINT8 code[3];
     UINT8 flags;
  } command_t;
  UINT8 messageSource;
  bool Messaging_on();
  bool Messaging_radioHandler();
  bool Messaging_doCommand(const UINT8* str, UINT8 flags);
  void Messaging_serialHandler();
#endif
```

The ScatterWeb.Messaging provides functions that allow you to connect to, monitor, and message on the network and send, receive, or peek messages. Control the MSP via RS232 or radio with simple ASCII commands.

Remote command execution

To issue an command over radio write @[id], answers will get [id] prep ended.

@85 rid

[85] 85

@85 tim

[85] 23080608,984 2001-09-25 03:16:48

Command creation

Use the COMMAND macro to easily create a new terminal command. The first argument to the Macros is always the three digit terminal command code. The second argument to the Macros is a 8bit flag field. For example the following:

```
COMMAND(xxx, 0){ print("test\r\n"); }
```

will evaluate to:

```
void term_xxx (const UINT8* str);
__attribute__((section(".commands"))) const command_t com_xxx = { term_xxx,
"xxx", 0 };\
void term_xxx (const UINT8* str) { print("test\r\n"); }
```

Besides creating the function header it first creates a forward declaration which is then needed in the following definition of a const command_t com_xxx struct in the .commands segment. com_xxx contains a pointer to the function, the three digit function code and a flag field. Because all those command_t entries are put into a special segment (.commands), the function doCommand can easily loop through this segment (going from __term_start to __term_end also provide by the linker script) and search for the command. This even makes it possible, to declare terminal commands where needed, not only in ScatterWeb.Messaging.c! (The firmware commands will still stay centralized in ScatterWeb.Messaging.c to have code & documentation of the firmware commands in one place.)

The needed modifications of the linker script can be seen in System/ldscript.x:

```
\dontinclude ldscript.x
\skip .text :
\until   }  > text
```

All commands must write their answer to term_reply and it should not be longer than TERM_BUFSIZE. If nothing is written to term_reply automatically an "OK" is generated. The only exception to that is when the bit SERIALONLY in the flag field is set. That means that the terminal command uses the serial functions directly (not writing its answer to term_reply) and can't be called over radio.

```
bool Messaging_on();
```

Function:	Enables the Messaging (started by system).
Parameter:	None
Return value:	<code>True</code> always.

```
bool Messaging_radioHandler();
```

Function:	Filters packets for Messaging.
Parameter:	-
Return value:	<code>True</code> always.

```
bool Messaging_doCommand(const UINT8* str, UINT8 flags);
```

Function:	Executes a command.
Parameter:	str - command to execute flags - flags of the command
Return value:	<code>True</code> always.

```
void Messaging_serialHandler();
```

Function:	This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.
Parameter:	-
Return value:	-

3.6 ScatterWeb.Net

```
\ScatterWeb 2.x\System\src\ScatterWeb.Net.h
#ifndef ScatterWeb_Net
#define ScatterWeb_Net
  #include "ScatterWeb.System.h"
  #define BROADCAST              (0xFFFF)
  #define UNKNOWN                (0x0000)
  #define ACK_PACKET               (0x00)
  #define SYNC_PACKET              (0x01)
  #define PING_PACKET              (0x02)
  #define PONG_PACKET              (0x03)
  #define MESSAGING_REQUEST_PACKET (0x04)
  #define MESSAGING_REPLY_PACKET   (0x05)
  #define SCANNER_REQUEST_PACKET   (0x06)
  #define SCANNER_REPLY_PACKET     (0x07)
  #define SENSOR_PACKET            (0x08)
  #define PICTURE_REQUEST_PACKET   (0x09)
  #define PICTURE_REPLY_PACKET     (0x10)
  #define FLASH_HEADER             (0x11)
  #define FLASH_HEADER_REPLY       (0x12)
  #define FLASH_DATA               (0x13)
  #define FLASH_REQ                (0x14)
  #define ROUTING_INNER_PACKET     (0x15)
  #define ROUTING_ROOT_PACKET      (0x16)
  #define ROUTING_INFO_PACKET      (0x17)
  #define ROUTING_TREE_PACKET      (0x18)
  #define RSA_REQUEST_PACKET       (0x19)
  #define RSA_REPLY_PACKET         (0x20)
  #define DHCP_PACKET              (0x21)
  #define DD_INTEREST_MESSAGE      (0x22)
  #define DD_MGMT_PACKET           (0x23)
  #define DD_DATA_MESSAGE          (0x24)
  #define DD_UPDATE_MESSAGE        (0x25)
  #define DD_ROUTING_MESSAGE       (0x26)
  #define DD_ROUTABLE_PACKET       (0x27)
  #define DSDV_UPDATE_PACKET       (0x28)
  #define DSDV_SEND_PACKET         (0x29)
  #define TRACKING_PACKET          (0x30)
  #define RTS_PACKET               (0x31)
  #define CTS_PACKET               (0x32)
  #define IS_RADIO_RECEIVING       (cdCounter)
  #define IS_RADIO_SENDING         ((P5OUT & 0xC0)==0x40)
  typedef struct {
     UINT16 to;
     UINT16 from;
     UINT8 type;
     UINT8 num;
     UINT8* header;
     UINT16 header_length;
     UINT8* data;
     UINT16 data_length;
  } packet_t;
  extern volatile packet_t rxPacket;
  extern packet_t sentPacket;
  extern volatile UINT8 txRetries;
  extern volatile UINT8 cdCounter;
  extern volatile UINT8 txState;
  bool Net_init();
  bool Net_on();
  bool Net_off();
  bool Net_setTxPower(UINT8 value);
  bool Net_send(packet_t* packet, fp_char_t callback);
  bool Net_rxHandler();
  bool Net_txHandler();
  char Net_esend(packet_t* packet, fp_char_t callback);
#endif
```

The ScatterWeb.Net implements a network service that enables you to develop applications that use network resources without worrying about the specific details of the individual protocol.

This file contains following functionality for radio:

- Setting of transmit power (in 100 steps).
- Interrupt driven rx & tx radio MAC logic (sending of packets with CSMA and random backoff, CRC-16 checksums, acknowledgement with automatic retransmission).

General

Radio software supports acknowledgements, so it is important to know that all packet types which are no broadcast will get acknowledged by radio software and get repeated several times if needed. The radio software uses the num field for acknowledged packets to recognize doubles. packet_t is a struct containing a packet header fields and pointers to packet data and is used for sending & receiving. As a packet may be often constructed out of a header portion and a data portion, packet_t supports two data pointers to get passed in for sending, so these portions don't need to be copied together. When receiving however, data pointer contains header and data consecutively and header and header_length are null.

How to receive packets

To receive packets of a specific type one must define a handler function and add it to Net_txHandler for firmware handlers or to the C_RADIO app callback function. When the handler is then called, the function can access the received packet in rxPacket.

How to send packets

For sending packets a packet_t must be initialized:

- to must contain the destination id of receiver or broadcast address.
- type must contain the packet type.
- header/data must be set to NULL or to the data that should get transported. If set, header_length/data_length should contain the length of the data.
- num can be set to transport a byte for not acknowledged packets, otherwise it will get overwritten.
- from is always overwritten with the id of this device.

Such a packet can then be queued into tx ring buffer with Net_send and a pointer to initialized packet_t. Additionally these two functions accept a callback pointer to functions which accept an UINT8. If set, this callback is called when the packet is removed from the tx ring buffer:

- For unacknowledged packets this is after sending the packet.
- For acknowledged packets this is after successfully receiving an ACK for this packet (callback argument is set to 1) or when all retries are made without success (callback argument is set to 0).

The callback function can check the variable txRetries to see how much retries it took (always 0 for unacknowledged packets).

```
bool Net_init();
```

Function:	Initializes the radio
Parameter:	None
Return value:	<code>True</code> always;

```
bool Net_on();
```

Function:	Enables the radio
Parameter:	None
Return value:	<code>True</code> always;

```
bool Net_off();
```

Function:	Disables the radio
Parameter:	None
Return value:	<code>True</code> always;

```
bool Net_setTxPower(UINT8 value);
```

Function:	Sets the transceiver power
Parameter:	value - tx power to set
Return value:	<code>True</code> if value are between 0 and 100;
	<code>False</code> otherwise.

```
bool Net_send(packet_t* packet, fp_char_t callback);
```

Function: Sends a packet over radio

Parameter: packet - packet to send

callback - will call after sending (without: use NULL)

Return value: <code>True</code> if packet is successfully buffered;

<code>False</code> otherwise.

```
bool Net_rxHandler();
```

Function: This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.

Parameter: -

Return value: -

```
bool Net_txHandler();
```

Function: This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.

Parameter: -

Return value: -

```
char Net_esend(packet_t* packet, fp_char_t callback);
```

Function: This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.

Parameter: -

Return value: -

3.7 ScatterWeb.String

```
\ScatterWeb 2.x\System\src\ScatterWeb.String.h
#ifndef ScatterWeb_String
#define ScatterWeb_String
  int String_writeUInt(fp_char_t writer, UINT16 number,
      UINT16 length);
  int String_writeULongInt(fp_char_t writer, UINT32 number,
      UINT16 length);
  bool String_fromByteToHex(UINT8 b, UINT8* cs);
  UINT8 String_fromHexToByte(const UINT8* cs);
  bool String_writeTemperature(fp_char_t writer, temp_t temp);
  bool String_abstractWrite(fp_char_t writer, const char *fmt,
      va_list argp);
  bool String_write(UINT8* dest, const char *fmt, ...);
  UINT16 String_parseInt(const UINT8* c, UINT8** ptr);
  UINT32 String_parseLongInt(const UINT8* c, UINT8** ptr);
#endif
```

The ScatterWeb.String represents text; that is, a series of Unicode characters.

```
int String_writeUInt(fp_char_t writer, UINT16 number, UINT16 length);
```

Function:	Converts int to String
Parameter:	witer - function to write
	number - number to write
	length - length to write
Return value:	position in output out[pos]

```
int String_writeULongInt(fp_char_t writer, UINT32 number, UINT16
length);
```

Function:	Converts long int to String
Parameter:	writer - function to write
	number - number to write
	length - length to write
Return value:	position in output out[pos]

```
bool String_fromByteToHex(UINT8 b, UINT8* cs);
```

Function:	Converts the byte *b into the string cs[0] cs[1].
Parameter:	b - byte
	cs - string
Return value:	<code>True</code> always.

```
UINT8 String_fromHexToByte(const UINT8* cs);
```

Function:	Converts the string at cs[0] cs[1] into a byte at *b.
Parameter:	cs – string
Return value:	value to convert into

```
bool String_writeTemperature(fp_char_t writer, temp_t temp);
```

Function:	Writes a temperature according to the format described in System_getTemperature ([+-]xxx.x).
Parameter:	writer - function to write
	temp - temperature
Return value:	<code>True</code> always.

```
bool String_abstractWrite(fp_char_t writer, const char *fmt, va_list
argp);
```

Function:	Simple printf implementation for RS232 output.
	Supports - amongst others:
	- %c: Prints a character.
	- %i: Prints an UINT16eger in decimal format.
	- %l: Prints an UINT32eger in decimal format
	- %b: Prints a byte in hex format.
	- %x: Prints an int in hex format.
Parameter:	writer - function to write
	fmt - function
	argp - list of arguments
Return value:	<code>True</code> always.

```
bool String_write(UINT8* dest, const char *fmt, ...);
```

Function:	Writes
Parameter:	dest - dest
	fmt - function
	... - list of parameters
Return value:	<code>True</code> always.

```
UINT16 String_parseInt(const UINT8* c, UINT8** ptr);
```

Function:	Parses int
Parameter:	c – c
	ptr - ptr
Return value:	parsed int

```
UINT32 String_parseLongInt(const UINT8* c, UINT8** ptr);
```

Function:	Parses long int
Parameter:	c - c
	ptr - ptr
Return value:	parsed long int

3.8 ScatterWeb.Synchronisation

```
\ScatterWeb 2.x\System\src\ScatterWeb.Synchronisation.h
#ifndef ScatterWeb_Synchronisation
#define ScatterWeb_Synchronisation
  #define ROOT_TIMEOUT          5
  #define FRAME_LENGTH          3
  #define INIT_LENGTH           4
  volatile struct {
     UINT8 seqNum;
     UINT16 masterRootId;
     UINT8 syncPeriods;
  } SyncStruct;
  bool Synchronisation_init();
  bool Synchronisation_isRoot();
  bool Synchronisation_registerSyncDone(fp_t fp);
  bool Synchronisation_registerSyncStart(fp_t fp);
  bool Synchronisation_radioHandler();
#endif
```

The ScatterWeb.Synchronisation represents the synchronisation.

```
bool Synchronisation_init();
```

Function:	Initializes the synchronisation (not started by system)
Parameter:	None
Return value:	<code>True</code> always.

```
bool Synchronisation_isRoot();
```

Function:	Checks if node is currently the root
Parameter:	None
Return value:	<code>True</code> if node is root; <code>False</code> otherwise.

```
bool Synchronisation_registerSyncDone(fp_t fp);
```

Function: Registers a function which is called if the sync ends

Parameter: fp - function to call

Return value: <code>True</code> always.

```
bool Synchronisation_registerSyncStart(fp_t fp);
```

Function: Registers a function which is called if the sync starts

Parameter: fp - function to call

Return value: <code>True</code> always.

```
bool Synchronisation_radioHandler();
```

Function: Filters packets for Synchronisation

Parameter: -

Return value: <code>True</code> always.

3.9 ScatterWeb.System

\ScatterWeb 2.x\System\src\ScatterWeb.System.h

```
#ifndef ScatterWeb_System
#define ScatterWeb_System
  #define WAKEUPLPM1(irqname, bit)  do {     \
     __asm__ __volatile__ ("bic %0, .L__FrameOffset_"
     #irqname "(r1)": : "i" ((uint16_t)LPM1_bits)); \
     runModule |= bit; \
     }while(0)
  #if !(defined(ESB) || defined(eGate) || defined(ECR))
     #error "No device specified: ECR, ESB, eGate"
  #endif
  #define TERM_BUFSIZE        (82)
  #define SERIAL_BUFSIZE      (82)
  #define RADIO_RXBUF_SIZE    (270)
  #define RADIO_TXBUF_SIZE    (400)
  #define MAX_TIMERS          (20)
  #define MAX_TASKS           (10)
  #define INFOMEM            __attribute__((section(".infomem")))
  #define INFOMEMNOBITS
     __attribute__((section(".infomemnobits")))
  #define   SET(x, y)     ((x) |= (y))
  #define   CLEAR(x, y)   ((x) &= (~y))
  #define MF_SCOS       (0x0001)
  #define MF_RADIO_RX   (0x0002)
  #define MF_RADIO_TX   (0x0004)
  #define MF_TIMER      (0x0008)
  #define MF_SERIAL_RX  (0x0010)
  #define MF_SENSORS    (0x0020)
  #define MF_RC5        (0x0080)
  #define INIT_FCS  0xffff
  #define C_SERIAL             1
  #define C_RADIO              2
  #define C_SENSOR             3
  #define NUMBER_CALLBACKS     4
  #define MICIFG    (0x01)
  #define MICIE     (0x02)
```

```
#define MICVOLTAGE        (ADC12MEM2)
#define EXTERNVOLTAGE     (ADC12MEM3)
#define BATTERYVOLTAGE    (ADC12MEM4)
#define RXPVOLTAGE        (ADC12MEM5)
#define BATTERYVOLTAGE    (ADC12MEM4)
enum {
   True = 1,
   False = 0,
};
typedef char bool;
typedef union { unsigned int u; signed int s; } temp_t;
typedef unsigned char UINT8;
typedef unsigned int UINT16;
typedef unsigned long UINT32;
typedef void(*fp_t)();
typedef void(*fp_string_t)(UINT8*);
typedef void(*fp_char_t)(UINT8);
typedef void(*fp_int_t)(UINT16);
typedef void(*fp_vp)(void*);
extern fp_vp callbacks[];
extern volatile UINT16 runModule;
extern volatile UINT16 rxpTemp;
extern volatile UINT16 rxpValue;
extern volatile UINT8 MIC;
bool System_registerCallback(UINT8 type, void(*fp_t)());
temp_t System_getTemperature();
bool System_setDCO();
bool System_startWatchdog();
bool System_stopWatchdog();
bool System_reset();
bool System_wait(UINT32 w);
bool System_waitA(UINT16 i);
bool System_powerOn();
bool System_powerOff();
UINT16 System_reprogram();
bool System_startConversion();
#include <msp430x14x.h>
```

```
  #include <io.h>
  #include <signal.h>
  #include <stdlib.h>
  #include <stdarg.h>
  #if defined(ECR)
    #include "../lib/ecr/programming.h"
    #include "../lib/ecr/flasher.h"
  #elif defined(ESB)
    #include "../lib/esb/programming.h"
    #include "../lib/esb/flasher.h"
  #elif defined(eGate)
  #elif defined(eGate)
    #include "../lib/egate/programming.h"
    #include "../lib/egate/flasher.h"
    #include "../lib/egate/FTerm.h"
    #include "../lib/egate/JTAGFunc.h"
    #include "../lib/egate/LowLevelFunc.h"
  #endif
  #include "ScatterWeb.Configuration.h"
  #include "ScatterWeb.Data.h"
  #include "ScatterWeb.Timers.h"
  #include "ScatterWeb.Time.h"
  #include "ScatterWeb.IO.h"
  #include "ScatterWeb.Comm.h"
  #include "ScatterWeb.Net.h"
  #include "ScatterWeb.Messaging.h"
  #include "ScatterWeb.Threading.h"
  #include "ScatterWeb.String.h"
  #include "ScatterWeb.Synchronisation.h"
#endif
```

The ScatterWeb.System contains fundamental functions that define commonly-used value and reference data types, events and event handlers, attributes, and processing exceptions.

ADC: Starting, enabling, disabling and ISR for ADU.

The main function of the ADU is to sample the unfiltered radio receive output and provide a "carrier detect" (see interrupt routine ADC12ISR and rxReceiveLimit in Configuration). Look for defines in ScatterWeb.System.h to access other ADC readings.

CLOCK: Methods for interfacing DS1629 (RTC with temperature) and LM71 (temperature sensor). For both devices the function getTemperature is implemented. It returns the temperature in fixed point [9.7] 2s complement format! DS1629 - initClock should be called at startup to check for the correct setting in the status register of the RTC. Most functions of the clock have been removed due to high current usage while reading the clock. Furthermore a software RTC has been implemented with a much finer resolution (see ScatterWeb.Time.c). The RTC is now only used as an well calibrated temperature sensor and only these functions remain here (for complete clock interfacing code look into attic directory). cl_start(), cl_stop(), cl_writeOnBus() and cl_readFromBus are the basic functions for interfacing the RTC (I2C like bus).

Ports (ESB/EYE): clockSDA P50, clockSCL P51, clockAlarm P15

Note: The sticks have no RTC and the ECR has the LM71 temperature sensor which is also realized in this file.

CRC: This file offers CRC calculation.

DCO: Initializing & Controlling the DCO (Digitally Controlled Oscillator)

If FF_DCOCHECKER in config_t.firmwareFlags is set Timers_init will add a timer every 100 ms for checkDCO and enable Timer_B1 interrupt. The interrupt continually measures the ACLK with the DCO (like in System_setDCO) and writes the result into dcoDiff. checkDCO takes samples every 100 ms of dcoDiff and adjusts the DCO if needed. As the DCO is temperature-sensitive and both serial and radio communication depend on the DCO calibrating the DCO continuously is recommended. Serial communication stops working at a temperature difference of ~10℃. A more low power option might be to not use checkDCO and just call System_setDCO in long intervals (a minute or even longer).

WATCHDOG: Hardware watchdog.

LPM: Power saving

The sensor board features low power modes to reduce power consumption. This increases battery-powered lifetime up to years and makes it even possible to run the sensor board solely with a gold cap / solar cell combo.

Following units can be switched off to reduce power consumption:

- Unit - Power consumption - Corresponding function – Comments
- ScatterWeb.Net - 4,7 - 5,2mA - Net_off(), Net_on()
- ScatterWeb.Data - 2,9mA - System_powerOff(), System_powerOn()

The MSP430 itself goes to sleep when nothing more has to be done (see super loop in main). Power consumption with everything disabled (Radio, Sensor) goes below 1mA. This file contains special definitions & configuration options for the firmware and includes all other headers. The firmware provides abstract functions to interface and use the hardware, contains the main execution loop + interrupts and provides some OS-like concepts. The firmware supports all ScatterWeb/MSP430 devices: ESB, EYE, eGate, ECR

The first reprogramming concept which produced the firmware/application separation was abandoned for two main reasons:

- It relied on a stable almost never changing firmware. At the moment this is absolutely wrong and it's questionable if it will ever be.
- It relied on the compiler/linker to produce always the same assembler output for the firmware when it gets always the same sources/object files. This was only partially right (it depended on too many factors: compiler version, compiler flags, even the machine the compiler is running on...) and could be wrong under other or future conditions.

Perhaps one needs to write his own compiler/linker (or at least modify an existing one) to really separate these two software parts on a MCU cleanly.

```
bool System_registerCallback(UINT8 type, void(*fp_t)());
```

Function:	Registers callback
Parameter:	type - type of callback
	() - function to call
Return value:	<code>True</code> always.

```
temp_t System_getTemperature();
```

Function:	Returns temperature
Parameter:	None
Return value:	current temperature.

```
bool System_setDCO();
```

Function:	Sets DCO
Parameter:	None
Return value:	<code>True</code> always.

```
bool System_startWatchdog();
```

Function:	Starts watchdog
Parameter:	None
Return value:	<code>True</code> always.

```
bool System_stopWatchdog();
```

Function:	Stops watchdog
Parameter:	None
Return value:	<code>True</code> always.

```
bool System_reset();
```

Function:	Resets System
Parameter:	None
Return value:	<code>True</code> always.

```
bool System_wait(UINT32 w);
```

Function: Waits - A simple looping waiting function. Rough estimate at 2,4576 MHz: w=1

=> 18us, w=10 => 62us, w=100 => 500us w=400 => 4ms.

This method has two big disadvantages: it blocks and it's not accurate, timers are better.

Parameter: w - ticks to wait

Return value: <code>True</code> always.

```
bool System_waitA(UINT16 i);
```

Function: Waits which is more precise than wait because it uses TimerA to produce its interval, also blocking.

Parameter: i - ticks to wait

Return value: <code>True</code> always.

```
bool System_powerOn();
```

Function: Enables the sensor and serial power

Parameter: None

Return value: <code>True</code> always.

```
bool System_powerOff();
```

Function: Disables the sensor and serial power

Parameter: None

Return value: <code>True</code> always.

```
UINT16 System_reprogram();
```

Function: This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.

Parameter: -

Return value: -

```
bool System_startConversion();
```

Function: This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.

Parameter: -

Return value: -

3.10 ScatterWeb.Threading

```
\ScatterWeb 2.x\System\src\ScatterWeb.Threading.h
#ifndef ScatterWeb_Threading
#define ScatterWeb_Threading
  extern volatile UINT16 Threading_currentThread;
  bool Threading_init();
  bool Threading_add(void* address);
  bool Threading_doEvents();
  bool Threading_sleep(UINT16 ticks);
  bool Threading_wait(UINT8* adr, UINT8 bitmask);
  bool Threading_eventHandler();
  bool Threading_packetHandler();
  bool Threading_sentPacketHandler(UINT16 p, UINT16 success);
  bool Threading_recv(UINT16 sender, UINT16 type, UINT16 timeout);
  bool Threading_recvp(UINT16 sender, UINT16 type, UINT16 timeout);
  bool Threading_send (UINT16 to, UINT16 type, UINT8* h, UINT16 hl,
      UINT8* d, UINT16 dl);
  bool Threading_esend(UINT16 to, UINT16 type, UINT8*h, UINT16 hl,
      UINT16 d, UINT16 dl);
#endif
```

The ScatterWeb.Threading provides functions that enable multithreaded programming.

ScatterOS

This file implements a simple cooperative scheduling mechanism which allows to run task functions with OS like blocking calls (which in fact do not block the processor but return control to the system and get later resumed).

The OS primitives at the moment are:

- Threading_doEvents - Simply returns control to the system. As this is a cooperative scheduler long tasks or tasks which make no other Threading_ calls must interrupt themselves with Threading_doEvents.
- Threading_sleep - Makes the task wait a specified amount of system ticks (may get longer if system is busy).
- Threading_wait - Makes the task wait for a flag set.
- Threading_recv / Threading_recvp - Makes the task wait for a packet of a given type (and a given sender if wanted). A timeout can be set and Threading_recvp works in promiscuous mode.
- Threading_send / Threading_esend - A synchronous send, task will continue after the packet was sent or lost (removed from radio rx ring buffer). Threading_esend is for sending data from EEPROMm.

These are not many but suit the needs at the moment.

Constraints

The goal was to write a task switching mechanism with minimal memory requirements. Each preemptive OS requires and most cooperative OS use one stack per task (Exceptions seen so far are PeerOS in the EYES Project, which preempts only lower priority tasks and lets them reside at the stack while executing a higher priority task, and Salvo, which seems to implement a very similar approach). To have a stack for each task increases memory requirements a lot, because the size of the stack for a task is unknown and must be generously set. The optimum would be the maximum stack depth of the task call tree. A compiler could easily calculate this if recursion and function pointers are not used, but i do not know if any compiler supports this feature. The second bad is that most times a task won't use its maximum stack depth. In contrast to theses approaches this mechanism uses no memory (only some static allocated memory for task table and a little bit of stack space like in every other function is needed). This puts some limits onto the task functions to make this possible. Task functions can't and should never allocate stack space. In fact you can allocate stack space and the scheduler will handle it properly (that was needed for calls with long argument list which get pushed onto the stack), but after a Threading_ call the stack content will always be undefined. That means that all state must be held in vars declared globally. Additionally they should be declared volatile so the compiler won't do any optimization on them (hold them in registers, also register content is undefined after a Threading_ call). Secondly it means that blocking Threading_ functions are only allowed to call within your top-level task function, not in functions called from your top-level task function!

The reason for that is again the stack, which is undefined after a Threading_ call and so will thrash the return address for Threading_ calls anywhere outside from your top-level task function.

How it works

A task is initialized with Threading_add, which gets a pointer to the task function. If a task can run (like a newly added task) sooner or later scos_call with the id of the task in the task table and a value is called. scos_call saves the current stack pointer, sets currentTask and calls the next-to-continue address of the task like a function with an int argument. Then the task runs until it calls a Threading_ function. These functions save the return address of the task on the stack as the next-to-continue address in the task table, change the state of the task and call scos_exit. scos_exit now just restores the stack pointer to the value scos_call saved and does a return, which now is like the return of scos_call and so the system is again in a clean state.

Related

The System Stack Pointer (SP)

The system stack pointer must always be aligned to even addresses because the stack is accessed with word data during an interrupt request service. The system SP is used by the CPU to store the return addresses of subroutine calls and interrupts. It uses a pre-decrement, post-increment scheme. The advantage of this scheme is that the item on the top of the stack is available. The SP can be used by the user software (PUSH and POP instructions), but the user should remember that the CPU also uses the SP. Figure 5–2 shows the system SP bits.

Call a subroutine:

```
CALL dst          SP – 2 –> SP, PC+2 –> stack, dst –> PC
```

Return from subroutine:

```
RET  MOV @SP+,PC
```

-fno-defer-pop

Always pop the arguments to each function call as soon as that function returns. For machines which must pop arguments after a function call, the compiler normally lets arguments accumulate on the stack for several function calls and pops them all at once. Perhaps a cleaner and more portable but also more memory consuming implementation would be possible with setjmp()/longjmp() macros.

Helper for scos_call

Had to be put into an extra function because of following problem:

The following instruction ("mov & savedSP, r1"::) is not really needed because if execution returns here, the stack pointer should be what we saved before. The important thing is to tell the compiler, that the usually caller preserved registers (r6-r11) may have changed! The problem is the indexing into the Tasks table. If following line is omitted the assembler output is:

```
scos_prolog(value, Tasks[currentTask].address, Tasks[currentTask].stackSize);
7154:  1e 42 50 06  mov    &0x0650,r14 ;0x0650
7158:  3b 40 54 06  mov    #1620,         r11     ;#0x0654
...
718a:  b0 12 42 71  call   #28994                 ;#0x7142
//__asm__ __volatile__("mov &savedSP, r1"::); // restore stack pointer
Tasks[currentTask].address = 0x0000;
718e:  1e 42 50 06  mov    &0x0650,r14 ;0x0650
...
71a0:  0f 5b        add    r11,    r15    ;
71a2:  8f 43 00 00  mov    #0,     0(r15) ;r3 As==00
Tasks[currentTask].state = T_DESTROYED;
...
71b8:  0f 5b        add    r11,    r15    ;
71ba:  8f 43 04 00  mov    #0,     4(r15) ;r3 As==00
```

That is bad. At the beginning the base address of Tasks is moved into r11 for reading out of the table for the scos_prolog call. r11 is later (after scos_prolog call) used to index into Tasks again, and the compiler can assume, that r11 did not change (according to normal calling conventions), but that may be wrong in this case. When the following instruction is used, the compiler omits the "add r11,r15" and addresses the table directly:

```
__asm__ __volatile__("mov &savedSP, r1"::); // restore stack pointer
718e: 11 42 4a 06  mov    &0x064a,r1   ;0x064a
Tasks[currentTask].address = 0x0000;
...
71a4: 8f 43 54 06   mov    #0,     1620(r15);r3 As==00
Tasks[currentTask].state = T_DESTROYED;
...
71ba: 8f 43 58 06   mov    #0,     1624(r15);r3 As==00
```

I don’t know why the compiler does that thing not always this way (it saves on instruction). I tried to force it by telling all registers get clobbered through this instruction:

```
__asm__ __volatile__("mov &savedSP, r1":: : "r15", "r14", "r13", "r12",
"r11", "r10", "r9", "r8", "r7", "r6");
restore stack pointer
```

But this produced the same behavior like the first, BUT instead of r11 being used it is now r5 which is even more worse. r5 is the frame pointer and strangely used by gcc. The compiler should never assume r5 is preserved over function calls, very strange why it does that here (even when r15 is appended to the list of clobbered registers). Anyway it works with the second. But that may change with -O flags or anything else, so best will be to put that into an extra function. Make sure this function does not get inline into scos_call again with -fno-inline for this file?!

```
bool Threading_init();
```

Function:	Initializes the threading
Parameter:	None
Return value:	<code>True</code> always.

```
bool Threading_add(void* address);
```

Function:	Adds thread
Parameter:	address - address of task
Return value:	<code>True</code> if successful; <code>False</code> otherwise.

```
bool Threading_doEvents();
```

Function:	Does events
Parameter:	None
Return value:	<code>True</code> always.

```
bool Threading_sleep(UINT16 ticks);
```

Function:	Sleeps for ticks
Parameter:	ticks - ticks to wait
Return value:	<code>True</code> always.

```
bool Threading_wait(UINT8* adr, UINT8 bitmask);
```

Function:	Waits until address == bitmask
Parameter:	adr - address bitmask - bitmask
Return value:	<code>True</code> always.

```
bool Threading_eventHandler();
```

Function:	This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.
Parameter:	-
Return value:	-

```
bool Threading_packetHandler();
```

Function: This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.

Parameter: -

Return value: -

```
bool Threading_sentPacketHandler(UINT16 p, UINT16 success);
```

Function: This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.

Parameter: -

Return value: -

```
bool Threading_recv(UINT16 sender, UINT16 type, UINT16 timeout);
```

Function: This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.

Parameter: -

Return value: -

```
bool Threading_recvp(UINT16 sender, UINT16 type, UINT16 timeout);
```

Function: This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.

Parameter: -

Return value: -

```
bool Threading_send
(UINT16 to, UINT16 type, UINT8* h, UINT16 hl, UINT8* d, UINT16 dl);
```

Function: This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.

Parameter: -

Return value: -

```
bool Threading_esend
(UINT16 to, UINT16 type, UINT8*h, UINT16 hl, UINT16 d, UINT16 dl);
```

Function:	This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.
Parameter:	-
Return value:	-

3.11 ScatterWeb.Time

\ScatterWeb 2.x\System\src\ScatterWeb.Time.h

```
#ifndef ScatterWeb_Time
#define ScatterWeb_Time
  #include "ScatterWeb.System.h"
  #define CURRENT_TIME \
      ({ \
            extern volatile time_t currentTime; \
            time_t time;  \
            dint(); \
            _NOP(); \
            time = currentTime; \
            eint(); \
            time; \
      })
  #define CURRENT_TIME_SECS \
      ({ \
            extern volatile time_t currentTime; \
            UINT32 secs;  \
            dint(); \
            _NOP(); \
            secs = currentTime.secs; \
            eint(); \
            secs; \
      })
  typedef struct {
      UINT8 year;
      UINT8 month;
      UINT8 day;
      UINT8 hour;
      UINT8 min;
      UINT8 sec;
  } btime_t;
  typedef struct {
      UINT32 secs;
      UINT16 millis;
  } time_t;
  time_t* Time_getSystemTime(time_t* time);
  bool Time_setSystemTime(time_t* time);
  bool Time_convertSystemTime(const time_t* src, btime_t* dst);
  bool Time_convertTime(const btime_t* src, time_t* dst);
  bool Time_writeTime(fp_char_t writer, btime_t* time);
  bool Time_readTime(const UINT8* c, btime_t* time);
#endif
```

The ScatterWeb.Time represents an instant in time, typically expressed as a date and time of day. This file keeps the global time of this node in the global var currentTime. It contains methods for writing, reading and converting currentTime. To avoid concurrency problems accessing currentTime must be done with disabled interrupts or in interrupt routines only. This defines some macro for accessing Time_currentTime safely.

```
time_t* Time_getSystemTime(time_t* time);
```

Function: Read currentTime or safely with disabled interrupts. Easier (and also faster) to use is CURRENT_TIME:

time_t time = CURRENT_TIME; <-> time_t time; getCurrentTime(&time);

Parameter: time - current time struct to store system time

Return value: pointer of time

```
bool Time_setSystemTime(time_t* time);
```

Function: Write currentTime or safely with disabled interrupts.

Parameter: time - current time to set

Return value: <code>True</code> always.

```
bool Time_convertSystemTime(const time_t* src, btime_t* dst);
```

Function: Converts time to broken down time

Parameter: src - input current time

dst - output btime

Return value: <code>True</code> always.

```
bool Time_convertTime(const btime_t* src, time_t* dst);
```

Function:	Converts broken down time to time
Parameter:	src - input time
	dst - output current time
Return value:	<code>True</code> always.

```
bool Time_writeTime(fp_char_t writer, btime_t* time);
```

Function:	Writes broken down time as a string in the form yyyy-mm-dd hh:mm:ss
Parameter:	writer - function to write
	time - time to write
Return value:	<code>True</code> always.

```
bool Time_readTime(const UINT8* c, btime_t* time);
```

Function:	Reads broken down time as a string.
	The format is the same as in Time_writeTime, but the year must be written without 2000. The number of digits is not fixed like in Time_writeTime, so it is ok to write 03-09-07 08:31:02 or 3-9-7 8:31:2.
Parameter:	c - array to store time
	time - time to read
Return value:	<code>True</code> always.

3.12 ScatterWeb.Timers

```
\ScatterWeb 2.x\System\src\ScatterWeb.Timers.h
#ifndef ScatterWeb_Timers
#define ScatterWeb_Timers
  #include "ScatterWeb.System.h"
  typedef void(*fp_timer)(UINT16);
  bool Timers_init();
  bool Timers_add(UINT16 ticks, fp_timer fp, UINT16 data);
  bool Timers_reset(UINT16 t, fp_timer fp, UINT16 data);
  bool Timers_remove(fp_timer fp, UINT16 data);
  bool Timers_eventHandler();
#endif
```

The ScatterWeb.Timers provides the Timer component, which allows you to raise an event on a specified interval. The Timer component is a hardware-based timer, which allows you to specify a recurring interval at which the Elapsed event is raised in your application. You can then handle this event to provide regular processing.

Software timers

With theses function you can implement delays asynchronously. Just give a time and a pointer to a function which should get called when the time has elapsed. The timer entries are kept in a fixed size array (MAX_TIMERS, timers, numberTimers). The entry at position 0 will always contain the next timer to expire. The ticks in the timer entries are not decreased with every system tick but first accumulated into _ticks until one of two things happen:

- _ticks becomes greater than timers[0].ticks, what means that at least timers[0] has elapsed (if no timer is running timers[0].ticks is set to 0xFFFF, so this will never happen then). The interrupt routine sets callTimer to 1 and Timers_eventHandler will execute. It calls _consolidate and all elapsed timers (where ticks==0).
- - An overflow when adding a new timer is detected. For example a timer with 50000 ticks was added. _ticks has run up to 30000 and then a new timer with 40000 ticks comes in. To get the correct offset for the timer it has to be added internally with 30000+40000=70000 ticks, but that is greater than maximum integer value 65535. _consolidate is called which decreases first timer to 20000, _ticks to 0, the second timer is then added with 40000 ticks.

_consolidate reads _ticks and subtracts the value from all timers[i].ticks (or sets them to 0 if they are already smaller) and _ticks itself. This is a safe way to synchronize with the interrupt and better than setting _ticks to 0, because if _ticks is just increased by the interrupt in between read and write the single tick will not get lost.

```
bool Timers_init();
```

Function: Initialization of DCO, TimerA, TimerB and software timer table.

This method first calls SetDCO and initializes variables for timers. Timer_A is initialized to run from the divided 32,768 Hz Crystal at 4096 Hz. Timer_B is initialized to run from the self-regulated DCO at 4,505600 MHz.

Timer_A0 is used to generate an interrupt every 4 ticks once a 1024/second near to a millisecond.) Timer_A1(-3) is used by watchdog.c to implement a software watchdog. Timer_B0 is used by recir.c to capture RC5 signals every 890 us. Timer_B1(-6) is used to measure the DCO continously with TB6.

Parameter: None

Return value: <code>True</code> always.

```
bool Timers_add(UINT16 ticks, fp_timer fp, UINT16 data);
```

Function: Strictly adds a timer, also if it exists already. If ticks are passed, the given functions is called.

Parameter: ticks - ticks for elapse

fp - function to call if expired

data - parameter to call function (without: use 0xFFFF)

Return value: <code>True</code> if successful;

<code>False</code> otherwise.

```
bool Timers_reset(UINT16 t, fp_timer fp, UINT16 data);
```

Function:	Adds or (if it already exists) resets a timer.
Parameter:	ticks - ticks for elapse
	fp - function to call if expired
	data - parameter to call function (without: use 0xFFFF)
Return value:	<code>True</code> if successful;
	<code>False</code> otherwise.

```
bool Timers_remove(fp_timer fp, UINT16 data);
```

Function:	Removes timer if data is matching
Parameter:	fp - function to call if expired
	data - parameter to call function (without: use 0xFFFF)
Return value:	<code>True</code> if successful;
	<code>False</code> otherwise.

```
bool Timers_eventHandler();
```

Function:	This member supports the ScatterWeb infrastructure and is not intended to be used directly from your code.
Parameter:	-
Return value:	-

4 Usage

4.1 The [EMPTY] application

The [EMPTY] application presents the body to develop your own applications.

ScatterWeb.Process

The ScatterWeb.Process module enables you to initialize and start your application:

```
\ScatterWeb 3.x\Applications\<application>\<platform>\
  ScatterWeb.Process.c
```

Special commands of ScatterWeb.Process:

- app - read the name of the application

You are able to include all needed files like ScatterWeb.Event.h:

```
#include "ScatterWeb.Event.h"
```

You are in a position to include the core of the ScatterWeb.API:

```
#include "../../../../System/src/ScatterWeb.System.h"
```

The Process_interceptRadioHandler() handles all packets which are addressed to other nodes:

```
void Process_interceptRadioHandler() { }
```

The Process_radioHandler() handles all packets:

```
void Process_radioHandler() {
  Messaging_radioHandler();
  if ( !(rxPacket.to == BROADCAST
      || rxPacket.to == Configuration.id) ) {
    Process_interceptRadioHandler();
  }
  else {
    switch(rxPacket.type){
      case PING_PACKET: {...} break;
      case PONG_PACKET: {...} break;
      default: break;
    }
  }
}
```

The COMMAND(app, 0) approves other nodes to check which application on this node is running:

```
COMMAND(app, 0) {
  String_write(term_reply, "[app] [EMPTY]");
}
```

First of all the Process_init() will be called by the System when the complete core is initialized:

```
void Process_init(){
  System_registerCallback(C_RADIO, Process_radioHandler);
  Event_init();
  Comm_log(LOW, "| Application %s initialized\r\n\r\n",
      imageData.versionName);
}
```

ScatterWeb.Event

The ScatterWeb.Event module presents methods for handling and forwarding sensor data events.

Special commands of ScatterWeb.Event:

- dsr - set flags to print no sensor data to none
- esr - set flags to print sensor data to radio
- raf - read announce flags
- rsm - read sensor mask

- saf - set announce flags
  - Valid values are:
    - 01 = AF_SERIAL
    - 02 = AF_RADIO
    - or a combination of this values

- ssm - set sensor mask to x
  - Valid values are:
    - 01 = SENSOR_MICROPHONE
    - 02 = SENSOR_TEMPERATURE
    - 04 = SENSOR_RC5
    - 08 = SENSOR_MOVEMENT
    - 10 = SENSOR_VIBRATION
    - 40 = SENSOR_VOLTAGE
    - 80 = SENSOR_BUTTON
    - 20 = SENSOR_LIGHT
    - or a combination of this values

\ScatterWeb 2.x\Applications\<application>\<platform>
ScatterWeb.Event.h

```
#ifndef ScatterWeb_Event
#define ScatterWeb_Event
  #include "../../../../System/src/ScatterWeb.System.h"
  #define AF_SERIAL    (0x01)
  #define AF_RADIO     (0x02)
  typedef struct {
     UINT8 announceFlags;
     UINT8 sensorMask;
  } appconfig_t;
  bool Event_init();
#endif
```

Processes events according to sensorMask and announceFlags:

appconfig_t.sensorMask contains what sensors will be processed (commands "rsm"/"ssm"):

- 0x10 SENSOR_VIBRATION - Vibration sensor
- 0x08 SENSOR_MOVEMENT - PIR sensor
- 0x80 SENSOR_BUTTON - Button
- 0x01 SENSOR_MICROPHONE - Microphone
- 0x02 SENSOR_TEMPERATURE - Temperature
- 0x04 SENSOR_RC5 - RC5 Receiving
- 0x40 SENSOR_VOLTAGE - Battery Voltage
- 0x20 SENSOR_LIGHT - Light

appconfig_t.announceFlags contains what will be done with sensor events (commands "raf"/"saf"):

- 0x01 AF_SERIAL - Prints events to serial port
- 0x02 AF_RADIO - Broadcasts events in a SENSOR_PACKET.

So to get all sensors reported over serial link and radio do:

<code>

ssm FF

saf 03

</code>

```
bool Event_init();
```

Function:	Initializes the event handling
Parameter:	None
Return value:	<code>True</code> always.

4.2 How to install

Directory Tree:

ScatterWeb_Package

|--------- Compiler	installer for GNU C compiler
|--------- Source	current source of ScatterWeb

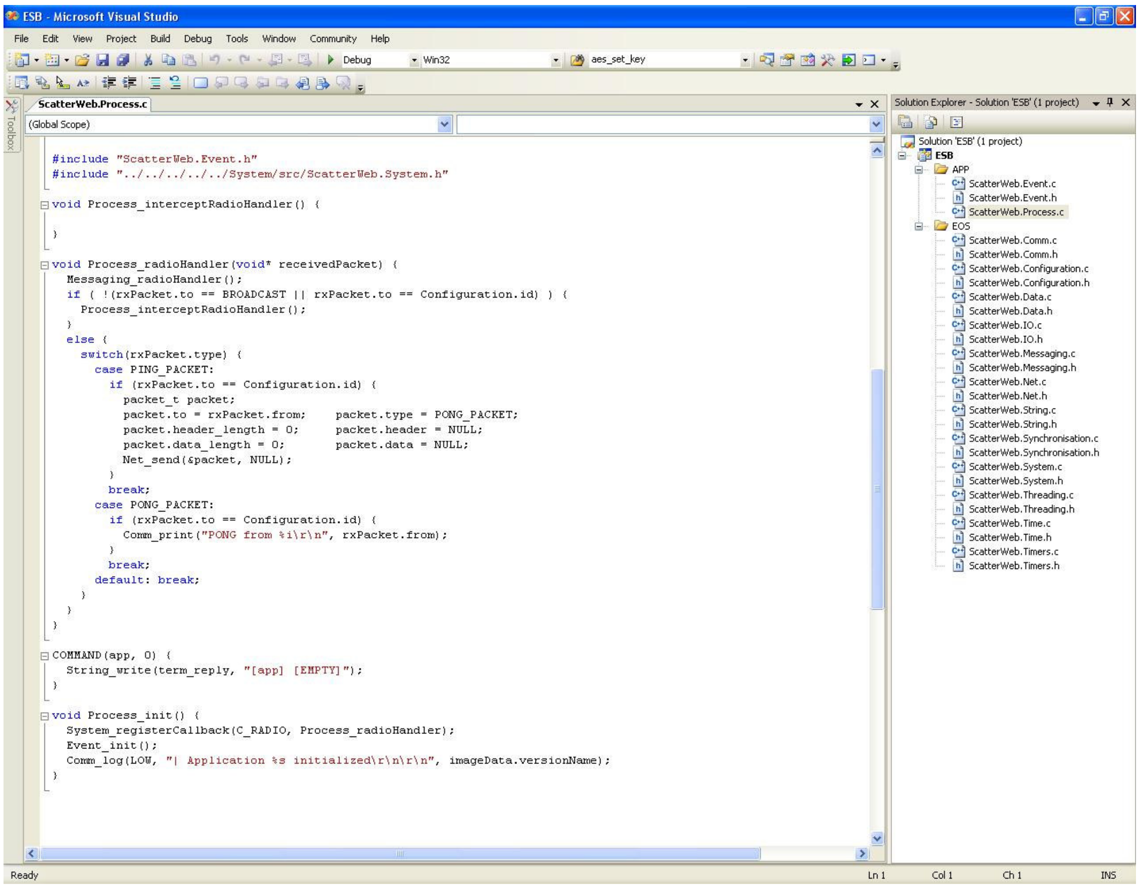
Installation:

1. Execute the installer for GNU C compiler

Usage:

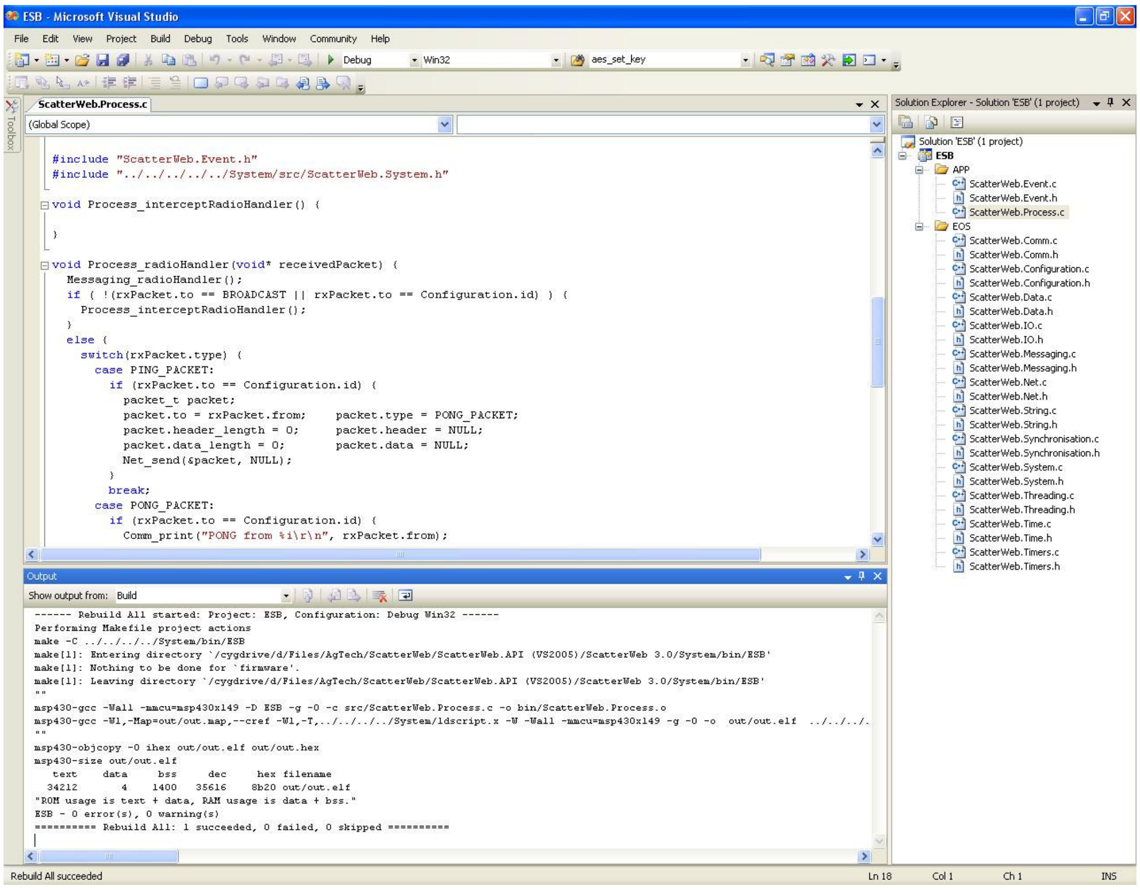
- With Visual Studio 2005, you can open the .sln solution files just by clicking them
- For compilation choose Build/Build Solution in the menu bar
- For flashing execute the flash.bat file or integrate it into the menu bar

4.3 How to use



```c
#include "ScatterWeb.Event.h"
#include "../../../../../System/src/ScatterWeb.System.h"

void Process_interceptRadioHandler() {

}

void Process_radioHandler(void* receivedPacket) {
  Messaging_radioHandler();
  if ( !(rxPacket.to == BROADCAST || rxPacket.to == Configuration.id) ) {
    Process_interceptRadioHandler();
  }
  else {
    switch(rxPacket.type) {
      case PING_PACKET:
        if (rxPacket.to == Configuration.id) {
          packet_t packet;
          packet.to = rxPacket.from;    packet.type = PONG_PACKET;
          packet.header_length = 0;     packet.header = NULL;
          packet.data_length = 0;       packet.data = NULL;
          Net_send(&packet, NULL);
        }
        break;
      case PONG_PACKET:
        if (rxPacket.to == Configuration.id) {
          Comm_print("PONG from %i\r\n", rxPacket.from);
        }
        break;
      default: break;
    }
  }
}

COMMAND(app, 0) {
  String_write(term_reply, "[app] [EMPTY]");
}

void Process_init() {
  System_registerCallback(C_RADIO, Process_radioHandler);
  Event_init();
  Comm_log(LOW, "| Application %s initialized\r\n\r\n", imageData.versionName);
}
```

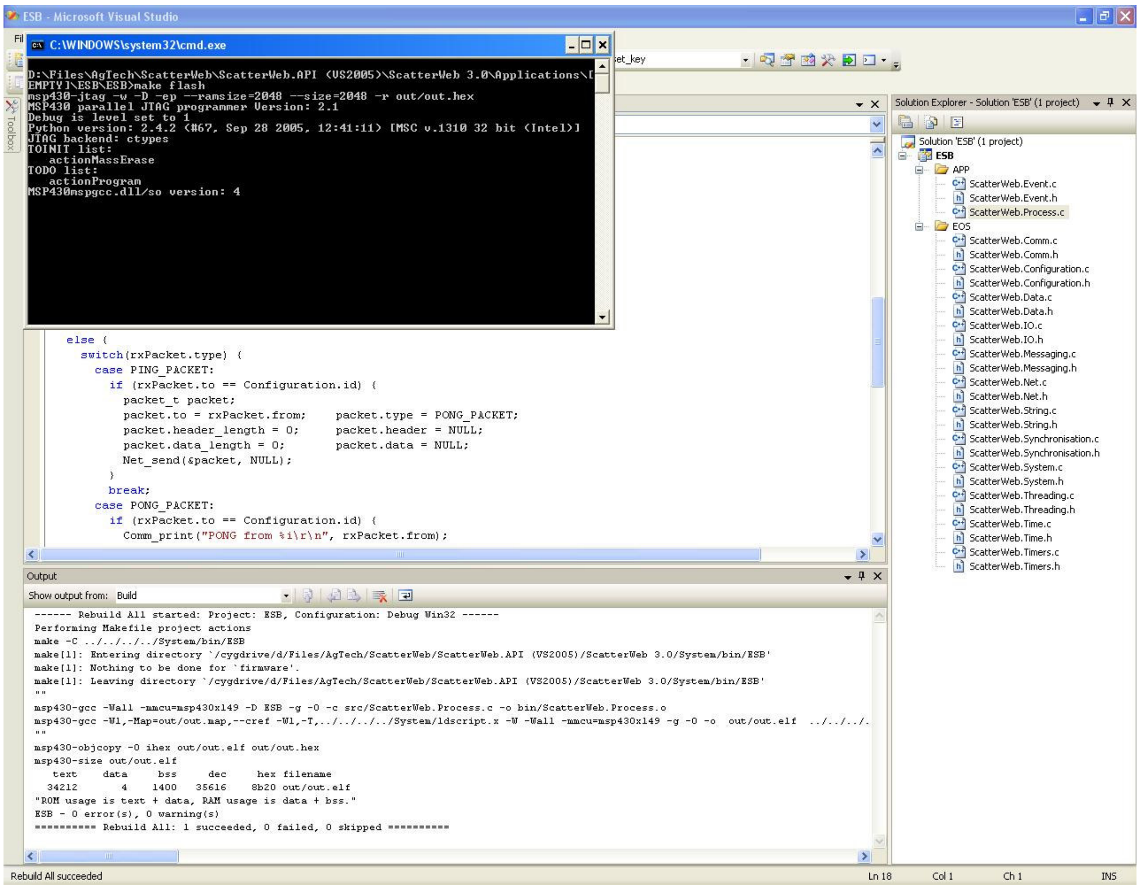
4.4 How to compile



```c
#include "ScatterWeb.Event.h"
#include "../../../../../System/src/ScatterWeb.System.h"

void Process_interceptRadioHandler() {

}

void Process_radioHandler(void* receivedPacket) {
  Messaging_radioHandler();
  if ( !(rxPacket.to == BROADCAST || rxPacket.to == Configuration.id) ) {
    Process_interceptRadioHandler();
  }
  else {
    switch(rxPacket.type) {
      case PING_PACKET:
        if (rxPacket.to == Configuration.id) {
          packet_t packet;
          packet.to = rxPacket.from;    packet.type = PONG_PACKET;
          packet.header_length = 0;     packet.header = NULL;
          packet.data_length = 0;       packet.data = NULL;
          Net_send(&packet, NULL);
        }
        break;
      case PONG_PACKET:
        if (rxPacket.to == Configuration.id) {
          Comm_print("PONG from %i\r\n", rxPacket.from);
```

```
------ Rebuild All started: Project: ESB, Configuration: Debug Win32 ------
Performing Makefile project actions
make -C ../../../../System/bin/ESB
make[1]: Entering directory `/cygdrive/d/Files/AgTech/ScatterWeb/ScatterWeb.API (VS2005)/ScatterWeb 3.0/System/bin/ESB'
make[1]: Nothing to be done for `firmware'.
make[1]: Leaving directory `/cygdrive/d/Files/AgTech/ScatterWeb/ScatterWeb.API (VS2005)/ScatterWeb 3.0/System/bin/ESB'
""
msp430-gcc -Wall -mmcu=msp430x149 -D ESB -g -O -c src/ScatterWeb.Process.c -o bin/ScatterWeb.Process.o
msp430-gcc -Wl,-Map=out/out.map,--cref -Wl,-T,../../../../System/ldscript.x -W -Wall -mmcu=msp430x149 -g -O -o  out/out.elf  ../../../.
""
msp430-objcopy -O ihex out/out.elf out/out.hex
msp430-size out/out.elf
   text    data     bss     dec     hex filename
  34212       4    1400   35616    8b20 out/out.elf
"ROM usage is text + data, RAM usage is data + bss."
ESB - 0 error(s), 0 warning(s)
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

4.5 How to flash



```
D:\Files\AgTech\ScatterWeb\ScatterWeb.API (VS2005)\ScatterWeb 3.0\Applications\[
EMPTY]\ESB\ESB>make flash
msp430-jtag -w -D -ep --ramsize=2048 --size=2048 -r out/out.hex
MSP430 parallel JTAG programmer Version: 2.1
Debug is level set to 1
Python version: 2.4.2 (#67, Sep 28 2005, 12:41:11) [MSC v.1310 32 bit (Intel)]
JTAG backend: ctypes
TOINIT list:
   actionMassErase
TODO list:
   actionProgram
MSP430mspgcc.dll/so version: 4
```

```
    else {
      switch(rxPacket.type) {
        case PING_PACKET:
          if (rxPacket.to == Configuration.id) {
            packet_t packet;
            packet.to = rxPacket.from;     packet.type = PONG_PACKET;
            packet.header_length = 0;      packet.header = NULL;
            packet.data_length = 0;        packet.data = NULL;
            Net_send(&packet, NULL);
          }
          break;
        case PONG_PACKET:
          if (rxPacket.to == Configuration.id) {
            Comm_print("PONG from %i\r\n", rxPacket.from);
```

```
------ Rebuild All started: Project: ESB, Configuration: Debug Win32 ------
Performing Makefile project actions
make -C ../../../../System/bin/ESB
make[1]: Entering directory `/cygdrive/d/Files/AgTech/ScatterWeb/ScatterWeb.API (VS2005)/ScatterWeb 3.0/System/bin/ESB'
make[1]: Nothing to be done for `firmware'.
make[1]: Leaving directory `/cygdrive/d/Files/AgTech/ScatterWeb/ScatterWeb.API (VS2005)/ScatterWeb 3.0/System/bin/ESB'
""
msp430-gcc -Wall -mmcu=msp430x149 -D ESB -g -O -c src/ScatterWeb.Process.c -o bin/ScatterWeb.Process.o
msp430-gcc -Wl,-Map=out/out.map,--cref -Wl,-T,../../../../System/ldscript.x -W -Wall -mmcu=msp430x149 -g -O -o  out/out.elf  ../../../.
""
msp430-objcopy -O ihex out/out.elf out/out.hex
msp430-size out/out.elf
   text    data     bss     dec     hex filename
  34212       4    1400   35616    8b20 out/out.elf
"ROM usage is text + data, RAM usage is data + bss."
ESB - 0 error(s), 0 warning(s)
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

Appendix A – Terminal Command Listing

Command	Parameter	Description	Location
app		read the name of the application	ScatterWeb.Process
brt	x	init Comm with x bps Valid values are: 01 = BPS_2400 02 = BPS_14400 03 = BPS_19200 04 = BPS_38400 05 = BPS_57600 06 = BPS_115200	ScatterWeb.Comm
dco	x	if x==+ set dco++, if x==- set dco—	ScatterWeb.Messaging
dea		erase eeprom	ScatterWeb.IO
est		prints software header of image	ScatterWeb.System
flx	x…y	broadcasting a userapp from x to y	ScatterWeb.System
fst		prints running software header	ScatterWeb.System
lst		list commands	ScatterWeb.Messaging
mem		read free stack bytes	ScatterWeb.Messaging
pcl		does a reset	ScatterWeb.Messaging
png	x	send ping packet to x	ScatterWeb.Messaging
raf		read announce flags	ScatterWeb.Event
rbu		read button state	ScatterWeb.Messaging
rcf		read configuration	ScatterWeb.Configuration
rer	x y	read eeprom from x to y	ScatterWeb.IO
rff		read firmware flags	ScatterWeb.Configuration
rfl	x y	read flash from x to y	ScatterWeb.IO
rfr		read receive limit	ScatterWeb.Messaging
rid		read id	ScatterWeb.Configuration
rlb		read boot log level	ScatterWeb.Configuration
rlg		read green led state	ScatterWeb.Messaging
rlo		read log level	ScatterWeb.Configuration

rlr		read red led state	ScatterWeb.Messaging
rly		read yellow led state	ScatterWeb.Messaging
rmm		read microphone value	ScatterWeb.Messaging
rrp		read temp rxp	ScatterWeb.Messaging
rsm		read sensor mask	ScatterWeb.Event
rsp		read stack pointer	ScatterWeb.Messaging
RST		reset system	ScatterWeb.Messaging
rtp		read transmit power	ScatterWeb.Messaging
rtt		read temperature	ScatterWeb.System
rvb		read battery value	ScatterWeb.Messaging
rve		read extern value	ScatterWeb.Messaging
saf	x	set announce flags Valid values are: 01 = AF_SERIAL 02 = AF_RADIO or a combination of this values	ScatterWeb.Event
sbp	x	if x==1 beeper on, otherwise off	ScatterWeb.Messaging
sco		read tasks	ScatterWeb.Threading
sff	x	set firmware flags Valid values are: 01 = FF_PROGRAMMABLE 02 = FF_DCOCHECKER or a combination of this values	ScatterWeb.Configuration
sfr	x	set receive limit	ScatterWeb.Messaging
sid	x	set id to x	ScatterWeb.Configuration
sir	x	send x over RC5	ScatterWeb.Messaging
slb	x	set boot log level Valid values are: 01 = NO 02 = LOW	ScatterWeb.Configuration

		03 = MEDIUM 04 = HIGH 05 = VERBOSE 20 = DEBUG_SYNC 40 = DEBUG_DCO 80 = DEBUG_RADIO or a combination of this values	
slg	x	if x==1 set green led on, otherwise off	ScatterWeb.Messaging
slo	x	set log level Valid values are: 01 = NO 02 = LOW 03 = MEDIUM 04 = HIGH 05 = VERBOSE 20 = DEBUG_SYNC 40 = DEBUG_DCO 80 = DEBUG_RADIO or a combination of this values	ScatterWeb.Configuration
slr	x	if x==1 set red led on, otherwise off	ScatterWeb.Messaging
sly	x	if x==1 set yellow led on, otherwise off	ScatterWeb.Messaging
ssm	x	set sensor mask to x Valid values are: 01 = SENSOR_MICROPHONE 02 = SENSOR_TEMPERATURE 04 = SENSOR_RC5 08 = SENSOR_MOVEMENT 10 = SENSOR_VIBRATION 40 = SENSOR_VOLTAGE 80 = SENSOR_BUTTON 20 = SENSOR_LIGHT	ScatterWeb.Event

		or a combination of this values	
stp	x	set transmit power to x	ScatterWeb.Messaging
swg		toggle green led	ScatterWeb.Messaging
swr		toggle red led	ScatterWeb.Messaging
swy		toggle yellow led	ScatterWeb.Messaging
tim		read system time	ScatterWeb.Messaging
tis	x	set system time to x seconds	ScatterWeb.Messaging
tiz	x	set system time to x milliseconds	ScatterWeb.Messaging
tmr		read timers	ScatterWeb.Timers
web	x y	write y at x of eeprom	ScatterWeb.IO

Appendix B – Terminal Command Availability

Command	ECR	ESB	EYE	eGate/USB	eGate/WEB
app	x	x	x	x	x
brt	x	x	x	x	x
dco	x	x	x	x	x
dea	x	x	x	x	x
est	x	x	x	x	x
flx	x	x	x	x	x
fst	x	x	x	x	x
lst	x	x	x	x	x
mem	x	x	x	x	x
pcl	x	x	x	x	x
png	x	x	x	x	x
raf	x	x	x	x	x
rbu	-	x	x	-	-
rcf	x	x	x	x	x
rer	x	x	x	x	x
rff	x	x	x	x	x
rfl	x	x	x	x	x
rfr	x	x	x	x	x
rid	x	x	x	x	x
rlb	x	x	x	x	x
rlg	-	x	x	x	x
rlo	x	x	x	x	x
rlr	x	x	x	x	x
rly	-	x	x	x	x
rmm	-	x	x	-	-
rrp	x	x	x	x	x
rsm	x	x	x	x	x
rsp	x	x	x	x	x
RST	x	x	x	x	x

rtp	x	x	x	x	x
rtt	x	x	x	-	-
rvb	-	x	x	-	-
rve	-	x	x	-	-
saf	x	x	x	x	x
sbp	-	x	x	-	-
sco	x	x	x	x	x
sff	x	x	x	x	x
sfr	x	x	x	x	x
sid	x	x	x	x	x
sir	-	x	x	-	-
slb	x	x	x	x	x
slg	-	x	x	x	x
slo	x	x	x	x	x
slr	x	x	x	x	x
sly	-	x	x	x	x
ssm	x	x	x	x	x
stp	x	x	x	x	x
swg	-	x	x	x	x
swr	x	x	x	x	x
swy	-	x	x	x	x
tim	x	x	x	x	x
tis	x	x	x	x	x
tiz	x	x	x	x	x
tmr	x	x	x	x	x
web	x	x	x	x	x