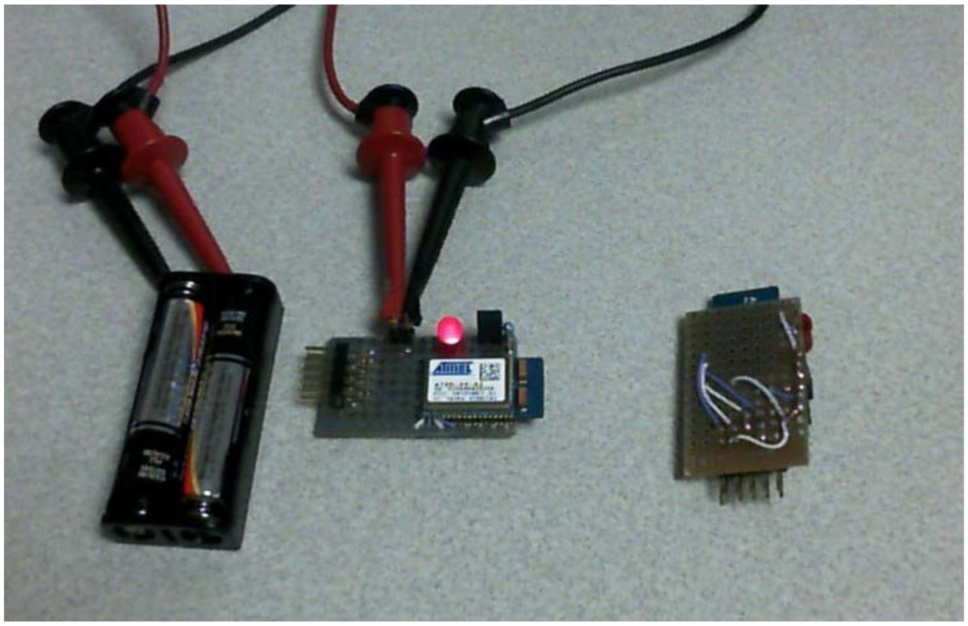
Using Atmel ATZB-24-A2 wireless Module and DIY hardware

The following procedure describes the steps to port the Star_nonbeacon example in the Atmel IEEE802.15.4 MAC to user-defined hardware. This example shows the use of an Atmel ATZB-24-A2 module with user defined hardware. The ATZB-24-A2 module has FCC modular approval. Modular Approval is a huge benefit for rapid development programs. Modular Approval allows developers to skip intentional emissions testing, FCC Part 15c. This reduces Non-Recurring Engineering (NRE) costs and time-to-market significantly. These benefits can easily outweigh the higher cost of a module for modest development programs. The Modular Approval also implies unintentional and conducted emissions are within the legal limits, for the module by itself. Developers still have to certify their final product for unintentional and conducted emissions, FCC Part 15b, because of possible emissions from additional electronics however unintentional emissions from the module will be minor.

The hardware used was about as simple as possible, hand-built with only one LED and one shunt but this process is just the starting point for unlimited projects.

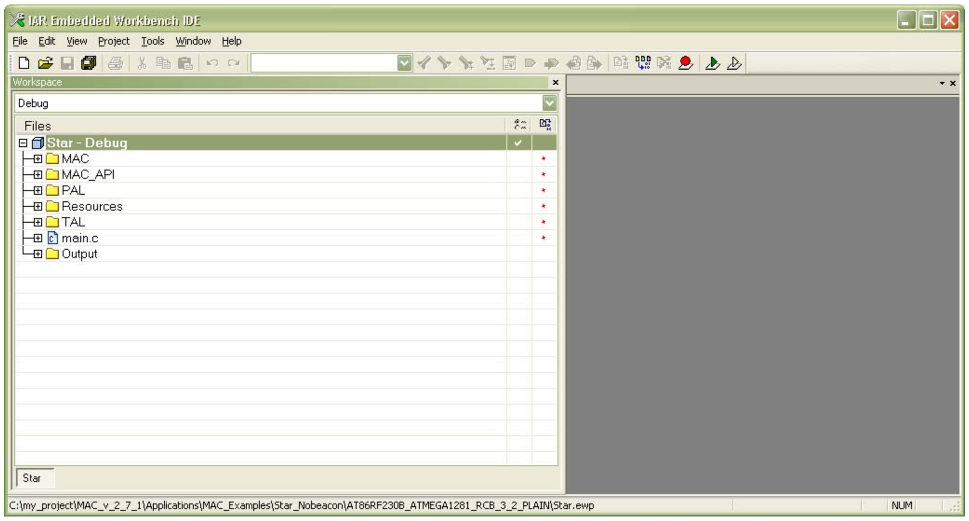


To outline the process we will create our own working directory within the MAC directory structure, configure the IAR tool chain, configure the path and C preprocessor to use our new directory. Then, after verification that the project structure is still sound, we will modify the code to try our new hardware.

- Install the IAR for Atmel AVR. This example uses rev 6.10

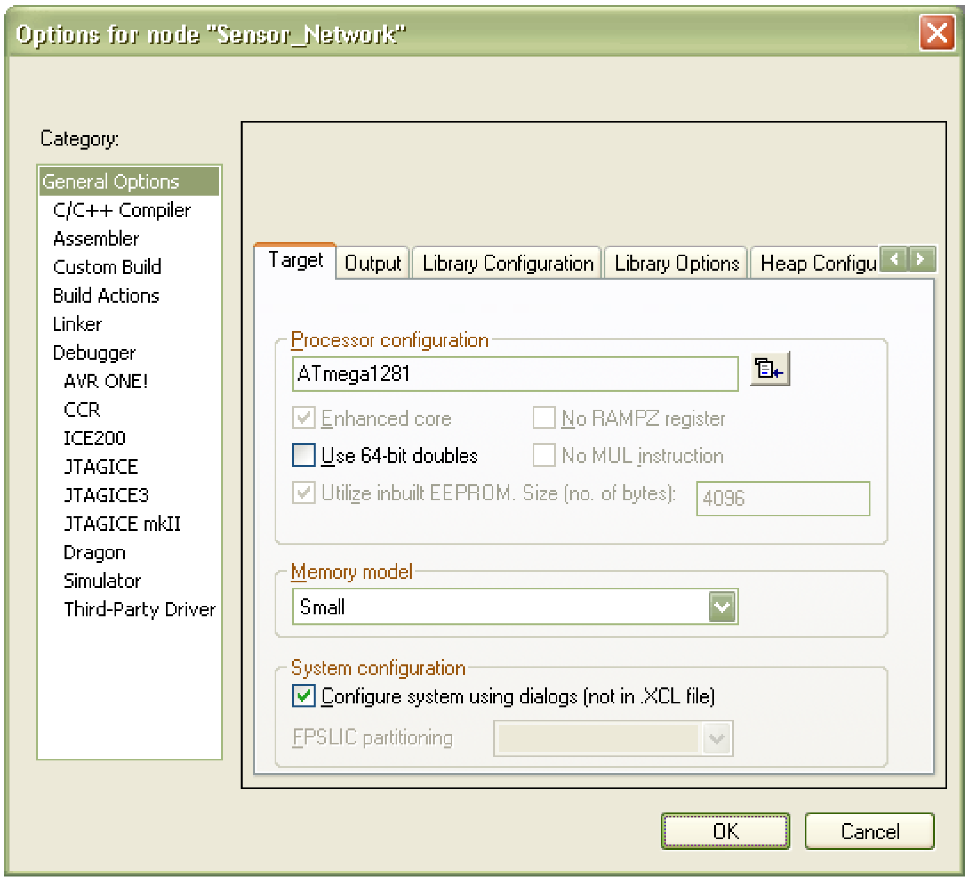
- Get IEEE802.15.4 MAC from Atmel website and run install. This example uses MAC rev 2.7.1

- Copy the entire stack to project directory c:\my_project\
- Copy the folder
  `C:\my_project\MAC_v_2_7_1\Applications\MAC_Examples\Star_Nobeacon\AT86RF230B_ATMEGA1281_RCB_3_2_PLAIN`
  To a new folder
  `C:\my_project\MAC_v_2_7_1\Applications\MAC_Examples\Star_Nobeacon\ZIGBIT_MY_BOARD`
- In IAR Launch
  `C:\my_project\MAC_v_2_7_1\Applications\MAC_Examples\Star_Nobeacon\ZIGBIT_MY_BOARD\Star.eww`
- Set up debug build see Atmel app note AVR2025 section 8.3.3.2
- Select debug workspace
- Highlight Star – Debug in the IAR File Explorer



C:\my_project\MAC_v_2_7_1\Applications\MAC_Examples\Star_Nobeacon\AT86RF230B_ATMEGA1281_RCB_3_2_PLAIN\Star.ewp

Select from the IAR menu Project|Options|General Options
Verify target processor is ATmega1281



Options for node "Sensor_Network"

Category:

- General Options
- C/C++ Compiler
- Assembler
- Custom Build
- Build Actions
- Linker
- Debugger
  - AVR ONE!
  - CCR
  - ICE200
  - JTAGICE
  - JTAGICE3
  - JTAGICE mkII
  - Dragon
  - Simulator
  - Third-Party Driver

Target | Output | Library Configuration | Library Options | Heap Configu

Processor configuration

ATmega1281

- [✓] Enhanced core
- [ ] No RAMPZ register
- [ ] Use 64-bit doubles
- [ ] No MUL instruction
- [✓] Utilize inbuilt EEPROM. Size (no. of bytes): 4096

Memory model

Small

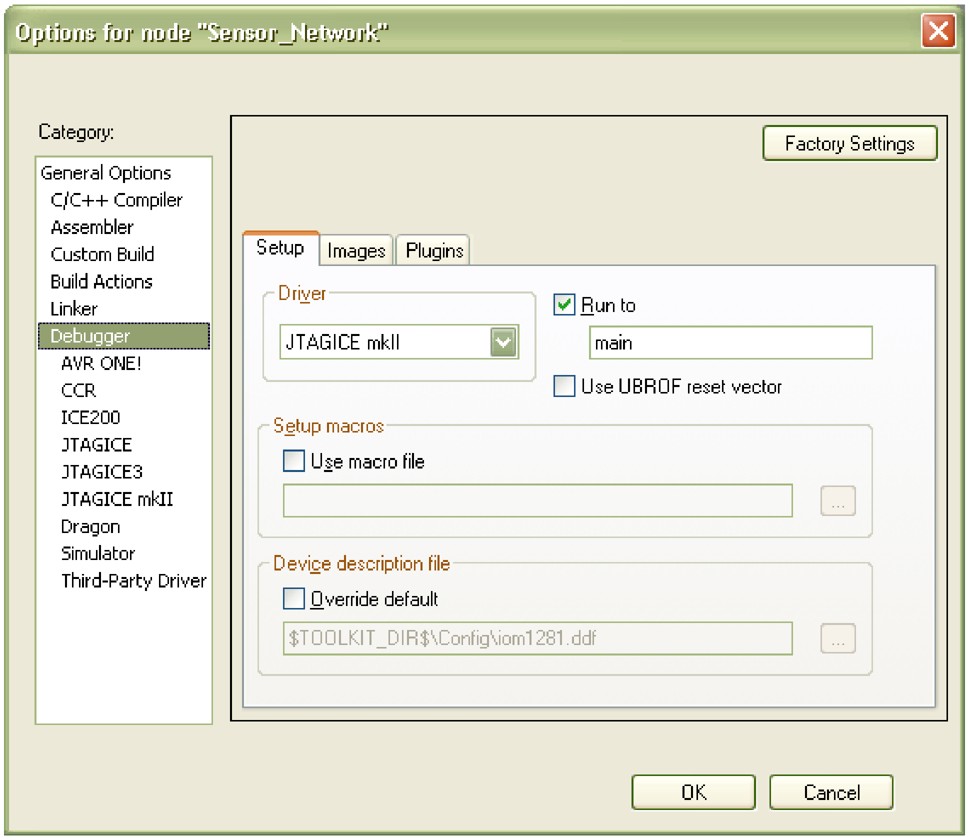
System configuration

- [✓] Configure system using dialogs (not in .XCL file)

FPSLIC partitioning

OK | Cancel

- Select from the sub menu Project|Options|Debugger
  Configure Driver = JTAGICE mkII and check Run to main box


Options for node "Sensor_Network"

Category:

General Options
C/C++ Compiler
Assembler
Custom Build
Build Actions
Linker
Debugger
AVR ONE!
CCR
ICE200
JTAGICE
JTAGICE3
JTAGICE mkII
Dragon
Simulator
Third-Party Driver

Factory Settings

Setup | Images | Plugins

Driver: JTAGICE mkII

[✓] Run to: main

[ ] Use UBROF reset vector

Setup macros

[ ] Use macro file

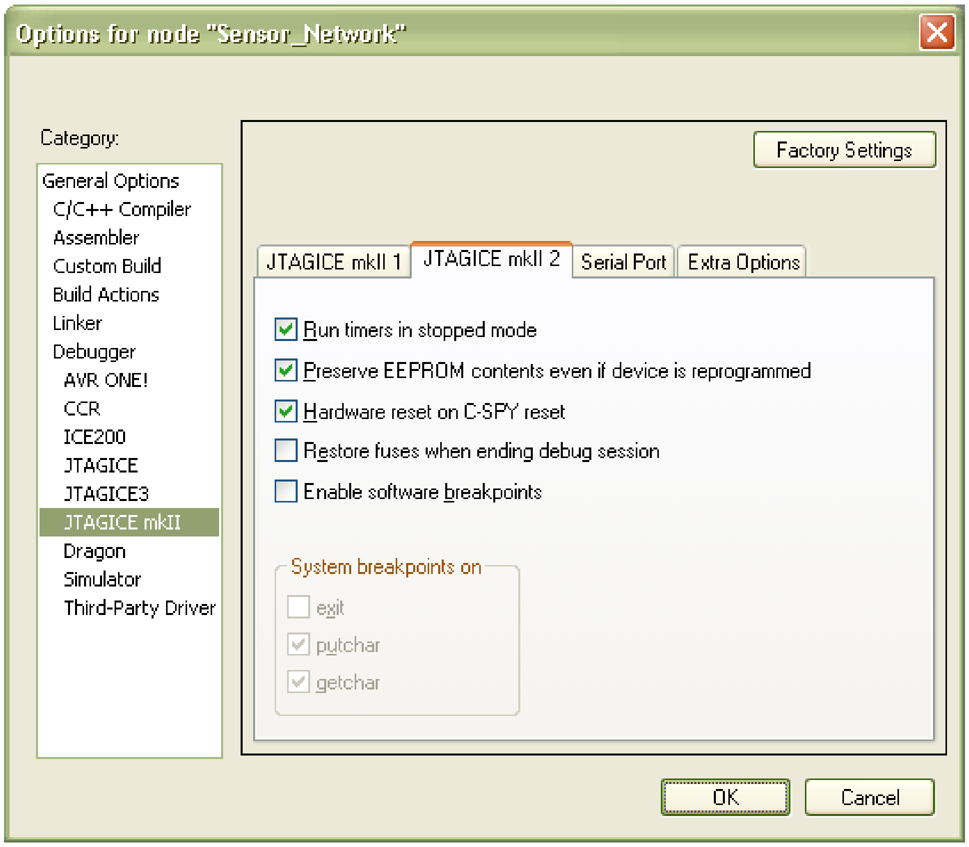
Device description file

[ ] Override default

$TOOLKIT_DIR$\Config\iom1281.ddf

OK | Cancel

- Select from IAR menu Project|Options|Debugger|JTAGICE mkII
  Select JTAGICE mkII 2 tab
  Check Preserve EEPROM contents even if device is reprogrammed
  Check Hardware reset on C-SPY reset


Options for node "Sensor_Network"

Category:

General Options
C/C++ Compiler
Assembler
Custom Build
Build Actions
Linker
Debugger
AVR ONE!
CCR
ICE200
JTAGICE
JTAGICE3
JTAGICE mkII
Dragon
Simulator
Third-Party Driver

Factory Settings

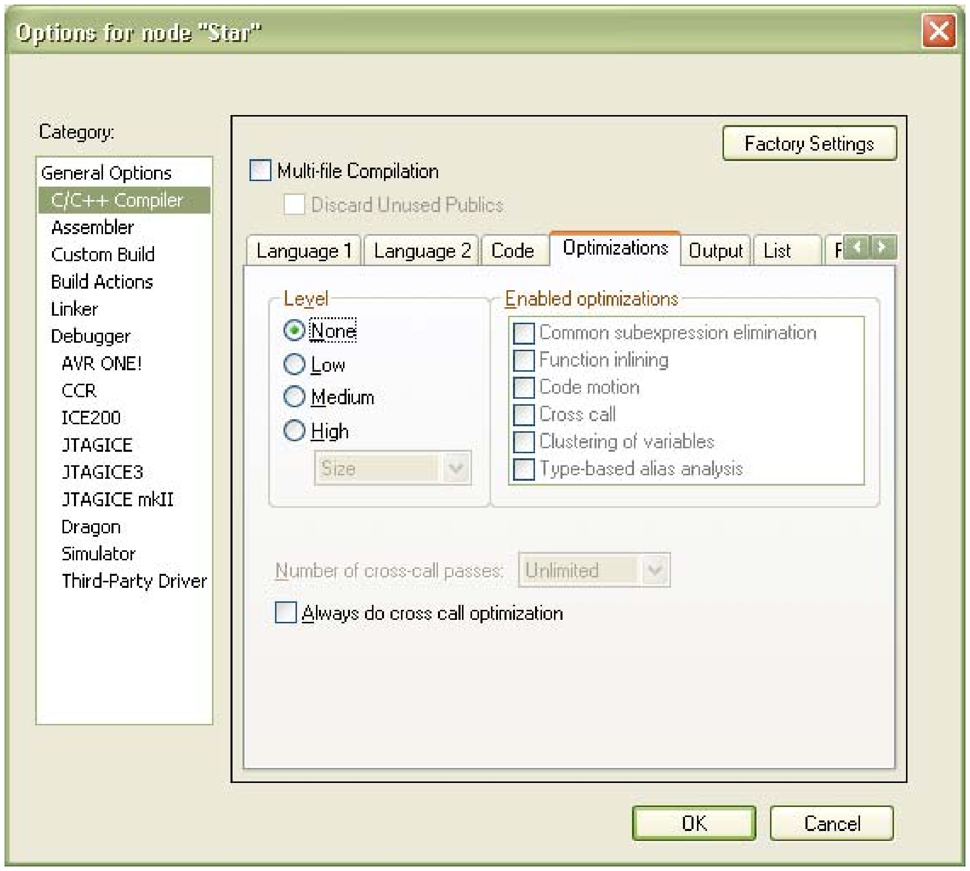
JTAGICE mkII 1 | JTAGICE mkII 2 | Serial Port | Extra Options

- [✓] Run timers in stopped mode
- [✓] Preserve EEPROM contents even if device is reprogrammed
- [✓] Hardware reset on C-SPY reset
- [ ] Restore fuses when ending debug session
- [ ] Enable software breakpoints

System breakpoints on
- [ ] exit
- [✓] putchar
- [✓] getchar

OK | Cancel

- Select from menu Project|Options|C/C++ Compiler
  Select Optimizations tab
  Set level to None
  Select List tab
  Verify Output list file is selected.


Options for node "Star"

Category:
- General Options
- C/C++ Compiler
- Assembler
- Custom Build
- Build Actions
- Linker
- Debugger
  - AVR ONE!
  - CCR
  - ICE200
  - JTAGICE
  - JTAGICE3
  - JTAGICE mkII
  - Dragon
  - Simulator
  - Third-Party Driver

Factory Settings

☐ Multi-file Compilation
☐ Discard Unused Publics

Language 1 | Language 2 | Code | Optimizations | Output | List | F

Level: [✓ None] Low Medium High
Size

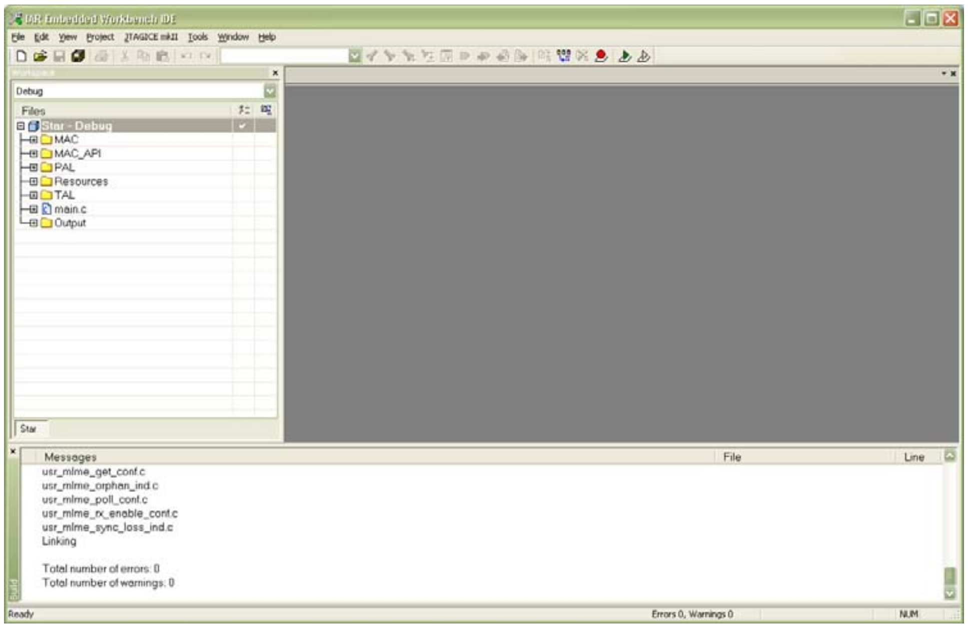
Enabled optimizations:
☐ Common subexpression elimination
☐ Function inlining
☐ Code motion
☐ Cross call
☐ Clustering of variables
☐ Type-based alias analysis

Number of cross-call passes: Unlimited

☐ Always do cross call optimization

OK Cancel

- Close Project|Options control panel
- Run Make to verify integrity of MAC project.
  You should get 0 errors and 0 warnings.


IAR Embedded Workbench IDE
File Edit View Project JTAGICE mkII Tools Window Help
Debug
Files
Star - Debug
MAC
MAC_API
PAL
Resources
TAL
main.c
Output
Star
Build

```
Messages
usr_mlme_get_conf.c
usr_mlme_orphan_ind.c
usr_mlme_poll_conf.c
usr_mlme_rx_enable_conf.c
usr_mlme_sync_loss_ind.c
Linking

Total number of errors: 0
Total number of warnings: 0
```

```
Ready                                        Errors 0, Warnings 0          NUM
```

- Save All and Save Workspace in IAR and Archive stack as baseline. I use .zip or .SVN

Next we will modify the project for the new hardware. We will need to modify the files that define the hardware. To reduce confusion we will endeavor to compartmentalize and contain the modified files in separate directory called "my_toolkit". Use the Windows OS to add a new directory to the project (NB the same location as the .eww file.) Add the modified files included with this app note to the my_toolkit directory. These include

pal_boardtypes.h
pal_config.h
pal_board.c
pal_irq.c
my_code.c
my_header.h

The file pal_boardtypes.h contains a list of board type constants. We need to add a new one for MY_BOARD at the end of the list and assign it the next number in the sequence (0x17). MY_BOARD will also be used as a C Compiler Preprocessor Symbol but that modification is done later in the process.

The file pal_config.h was burrowed from a project containing the ATZB-24-A2 hardware. This file maps thee I/O features to specific ports on of the ATmega1281. We modified the LED and Button features to match our hardware. Also we added the I/O table in the header comments for reference.

The file pal_board.c was also burrowed from a project containing the ATZB-24-A2 hardware and modified to reduce the number of LEDs and buttons. Also ATZB_24_MN2 constant was changed to MY_BOARD.

Likewise the file pal_irq.c was burrowed from a project containing the ATZB-24-A2 hardware and ATZB_24_MN2 constant was changed to MY_BOARD.

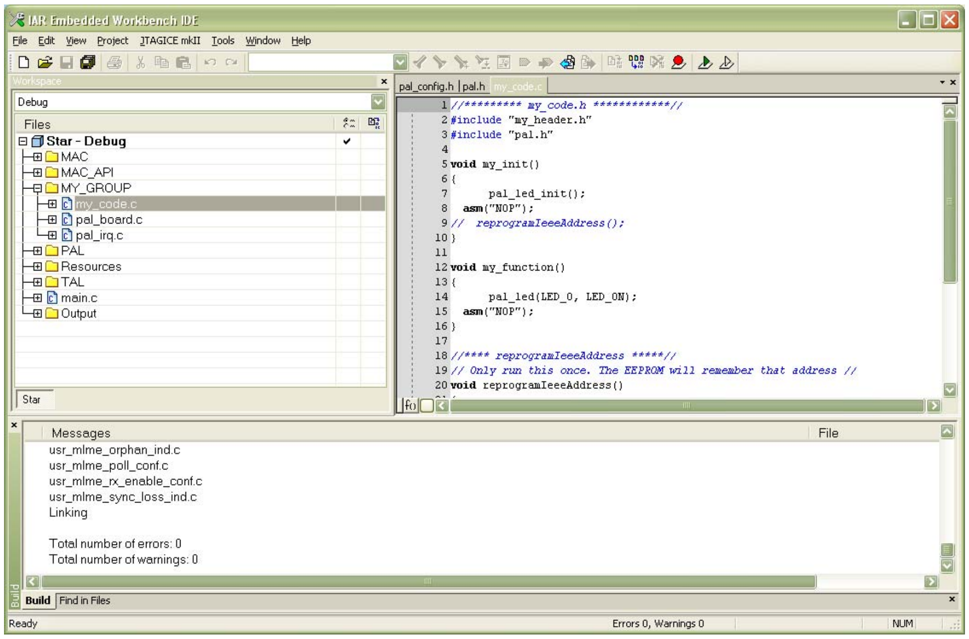
My_code.c and my_header.h contain source code for new functions.

To properly configure the IAR Workbench we need to update the path information in BOTH the IAR Workspace File Manager and the C Compiler. For some reason beyond the authors grasp, IAR retains obsolete path information (or meta-data) in the Workspace File Manager that needs to be flushed. If you are getting errors in compilation, verify the true locations of the .c and .h files. To do this, use the IAR Workspace File Manager and right-click on the file, such as pal_config.h. Select file properties and inspect the path in the control panel.

In the IAR Workspace File Explorer find the Star – Debug\PAL\Board group and delete it.

In the IAR Workspace File Explorer fine the Star – Debug\PAL\pal.c\pal_config.h and pal_boardtypes.h files and remove them from the project. (I'm not sure this does anything, but with meta-data can one really be sure of anything?)

In the IAR Workspace add a new group called MY_GROUP. Add all the .c files from my_toolkit to MY_GROUP.

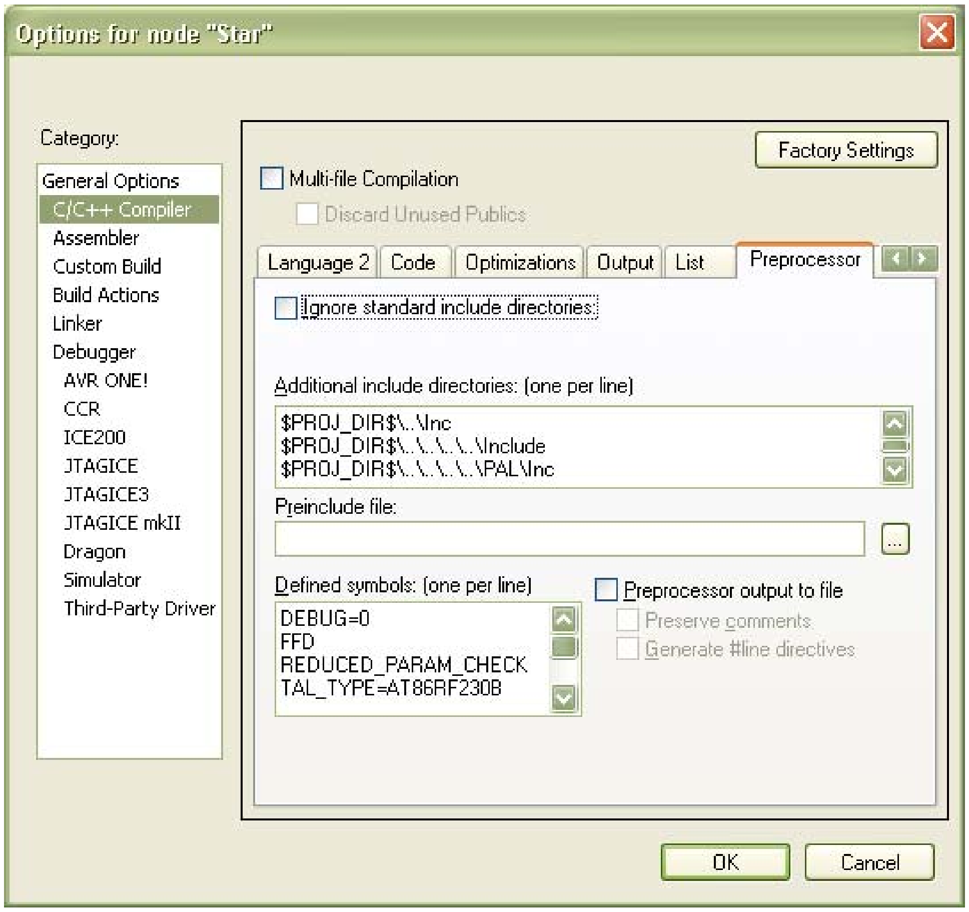


```
//********* my_code.h ************//
#include "my_header.h"
#include "pal.h"

void my_init()
{
      pal_led_init();
  asm("NOP");
//  reprogramIeeeAddress();
}

void my_function()
{
      pal_led(LED_0, LED_ON);
  asm("NOP");
}

//**** reprogramIeeeAddress *****//
// Only run this once. The EEPROM will remember that address //
void reprogramIeeeAddress()
```

```
usr_mlme_orphan_ind.c
usr_mlme_poll_conf.c
usr_mlme_rx_enable_conf.c
usr_mlme_sync_loss_ind.c
Linking

Total number of errors: 0
Total number of warnings: 0
```

Next we need to capture and modify the path and switch settings for the C compiler.

In IAR highlight the Star- Debug then open Projets|Options|C/C++ Compiler control panel and select the Preprocessor Tab. Cut and paste all the Additional include directories and Defined symbols to a text editor for modification



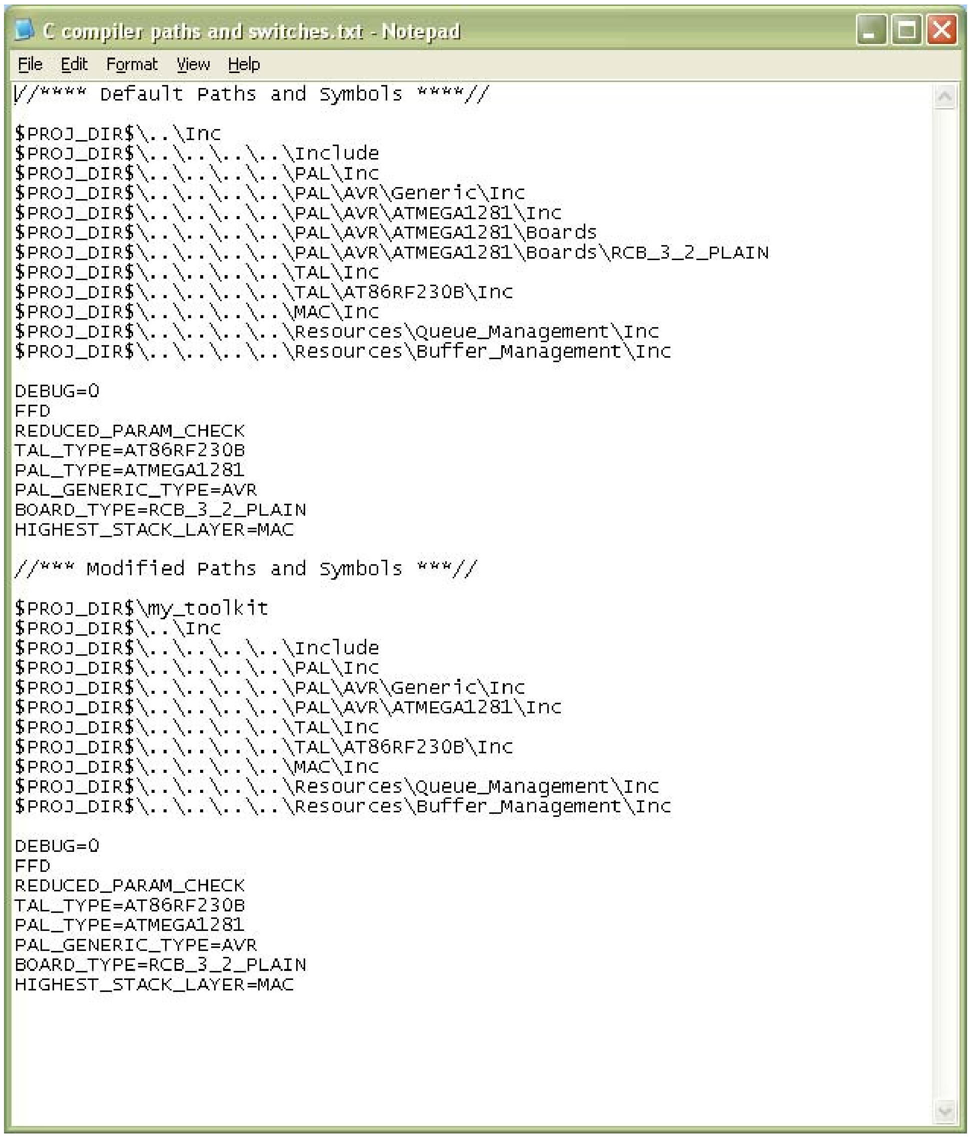
Additional include directories: (one per line)

```
$PROJ_DIR$\..\Inc
$PROJ_DIR$\..\..\..\..\Include
$PROJ_DIR$\..\..\..\..\PAL\Inc
```

Defined symbols: (one per line)

```
DEBUG=0
FFD
REDUCED_PARAM_CHECK
TAL_TYPE=AT86RF230B
```

Modify the path information as shown in the example below. Note the BOARD_TYPE setting. We will modify it a little later in this process after we verify the path information has been harmonized.


C compiler paths and switches.txt - Notepad
File Edit Format View Help

```
//***** Default Paths and Symbols ****//

$PROJ_DIR$\..\Inc
$PROJ_DIR$\..\..\..\..\Include
$PROJ_DIR$\..\..\..\..\PAL\Inc
$PROJ_DIR$\..\..\..\..\PAL\AVR\Generic\Inc
$PROJ_DIR$\..\..\..\..\PAL\AVR\ATMEGA1281\Inc
$PROJ_DIR$\..\..\..\..\PAL\AVR\ATMEGA1281\Boards
$PROJ_DIR$\..\..\..\..\PAL\AVR\ATMEGA1281\Boards\RCB_3_2_PLAIN
$PROJ_DIR$\..\..\..\..\TAL\Inc
$PROJ_DIR$\..\..\..\..\TAL\AT86RF230B\Inc
$PROJ_DIR$\..\..\..\..\MAC\Inc
$PROJ_DIR$\..\..\..\..\Resources\Queue_Management\Inc
$PROJ_DIR$\..\..\..\..\Resources\Buffer_Management\Inc

DEBUG=0
FFD
REDUCED_PARAM_CHECK
TAL_TYPE=AT86RF230B
PAL_TYPE=ATMEGA1281
PAL_GENERIC_TYPE=AVR
BOARD_TYPE=RCB_3_2_PLAIN
HIGHEST_STACK_LAYER=MAC

//*** Modified Paths and Symbols ***//

$PROJ_DIR$\my_toolkit
$PROJ_DIR$\..\Inc
$PROJ_DIR$\..\..\..\..\Include
$PROJ_DIR$\..\..\..\..\PAL\Inc
$PROJ_DIR$\..\..\..\..\PAL\AVR\Generic\Inc
$PROJ_DIR$\..\..\..\..\PAL\AVR\ATMEGA1281\Inc
$PROJ_DIR$\..\..\..\..\TAL\Inc
$PROJ_DIR$\..\..\..\..\TAL\AT86RF230B\Inc
$PROJ_DIR$\..\..\..\..\MAC\Inc
$PROJ_DIR$\..\..\..\..\Resources\Queue_Management\Inc
$PROJ_DIR$\..\..\..\..\Resources\Buffer_Management\Inc

DEBUG=0
FFD
REDUCED_PARAM_CHECK
TAL_TYPE=AT86RF230B
PAL_TYPE=ATMEGA1281
PAL_GENERIC_TYPE=AVR
BOARD_TYPE=RCB_3_2_PLAIN
HIGHEST_STACK_LAYER=MAC
```

- In IAR select Project|Clean

- In IAR select Project|Rebuild All

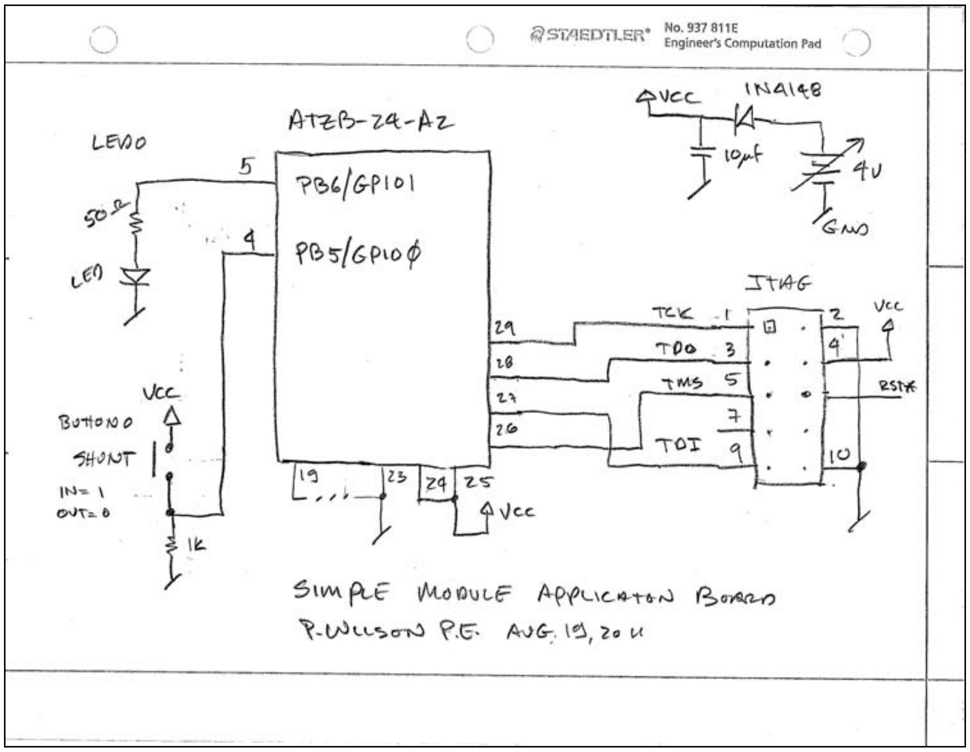
After the rebuild, use the IAR File Explorer verify the properties of the following files. Verify they are all in the my_toolkit directory.

MY_GROUP\pal_board.c
MY_GROUP\pal_irq.c
MY_GROUP\my_code.c
PAL\pal_boardtypes.h
PAL\pal_config.h

And finally change the C Compiler Switch to MY_BOARD. From the IAR Workspace menu select Project|Options|C/C++ Compiler. Go to the Preprocessor tab In the define symbols box change BOARD_TYPE symbol to BOARD_TYPE=MY_BOARD.

Run make again to verify the tool-chain is working with new BOARD_TYPE and using the files in my_toolkit. If things compiler without errors run another archive just for good practice. At this point we have set up the tool-chain and built the framework for user specific hardware. In the next steps we will modify the hardware connections to work with a new board.

In this example we made a really simple base-board for the ZigBit ATZB-24-A2 module. The schematic is shown below. The I/O consisted of 1 LED, 1 shunt and a JTAG connector. Not very a big step, but it is uncharted territory.



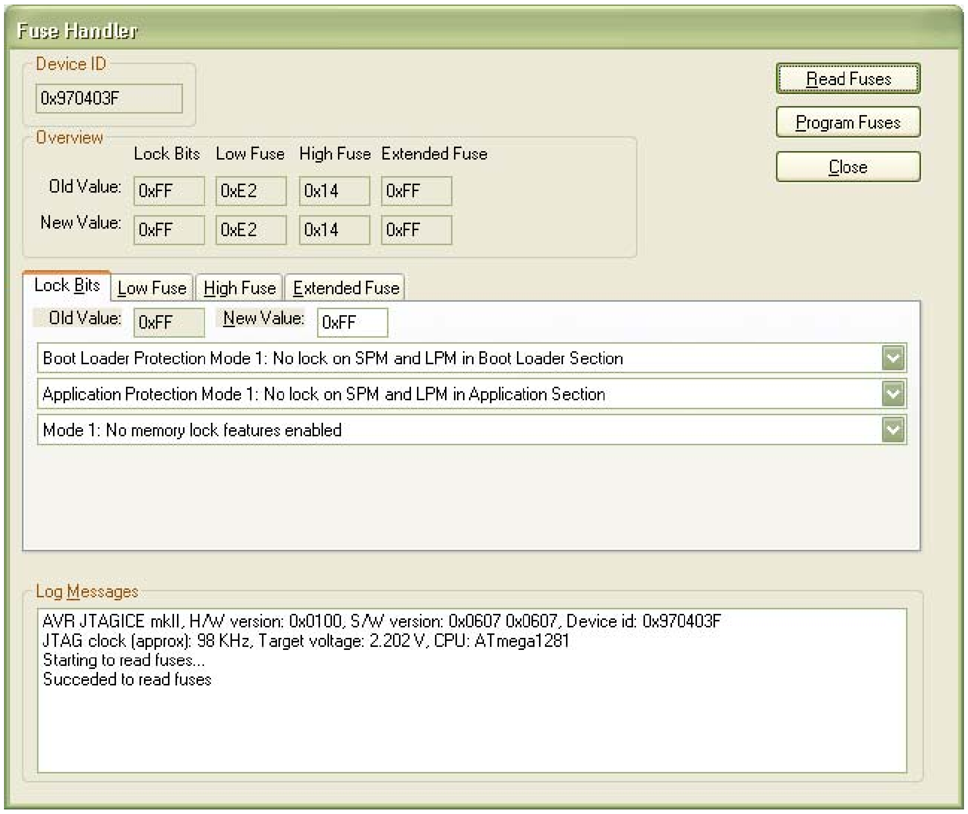
The settings for the Zigbit module are in the new pal_config.h file in my_toolkit. This header also includes a useful map of the ZigBit GPIO to the onboard ATmega1281 microcontroller.

Add #include "my_header.h" to main.c

Add my_init() function to the main loop after the wpan_init() test. My_init() is an assembly NOP command that can used for finite breakpoint.

OK let's program the board. Connect the board to JTAG programmer. I use the JTAGICE mkII. Power up the hardware and verify the green LED on the JTAG programmer is lit. Next check the fuse settings on the ATmega1281. Using IAR workbench menus select JTAGICEmkII. The fuse settings I used for the Zigbit module are

Lock bits 0xFF
Low Fuse 0xE2
High Fuse 0x14
Extended Fuse 0xFF



Device ID: 0x970403F

	Lock Bits	Low Fuse	High Fuse	Extended Fuse
Old Value:	0xFF	0xE2	0x14	0xFF
New Value:	0xFF	0xE2	0x14	0xFF

Old Value: 0xFF New Value: 0xFF

Boot Loader Protection Mode 1: No lock on SPM and LPM in Boot Loader Section
Application Protection Mode 1: No lock on SPM and LPM in Application Section
Mode 1: No memory lock features enabled

Log Messages

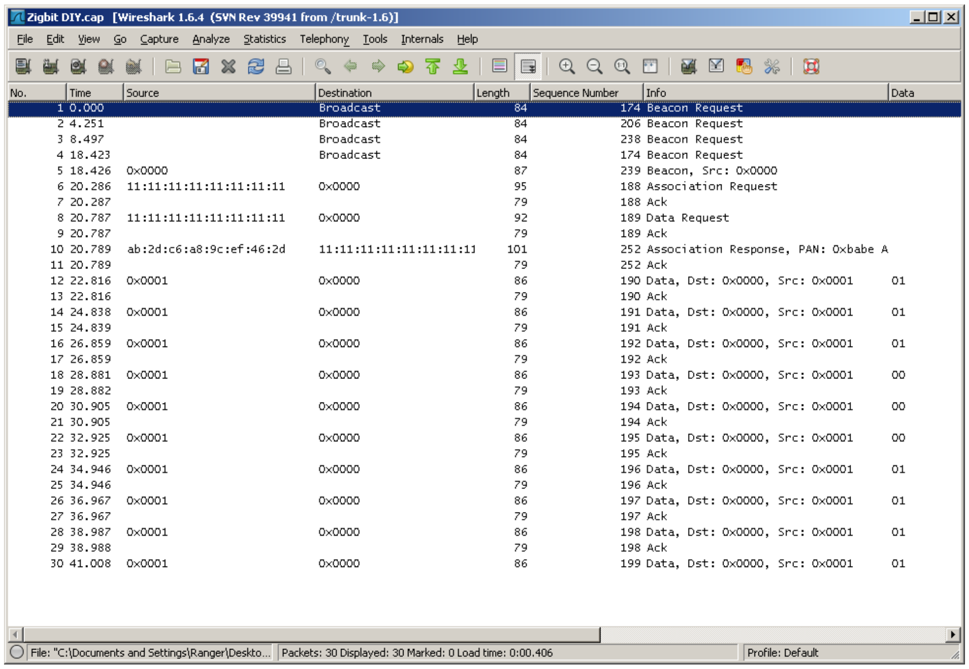
```
AVR JTAGICE mkII, H/W version: 0x0100, S/W version: 0x0607 0x0607, Device id: 0x970403F
JTAG clock (approx): 98 KHz, Target voltage: 2.202 V, CPU: ATmega1281
Starting to read fuses...
Succeded to read fuses
```

In the IAR Workbench open my_code.c and put a breakpoint on the asm("NOP") line inside my_init() function. If all goes well the debugger should land here. Launch the debugger. You should see the download 'zipper' as the program is loaded. If you see warnings about not being able to place breakpoints, this can be caused by inconsistent paths in the IAR file explorer. In other words the true file locations in the meta-data in IAR do not match.

Select the "Go" button in the debugger and if it's all working the debugger should hit the breakpoint in the my_code.c file and stop. Congratulations! This is a significant milestone, you are now able to add your own functions into my_code.c and my_header.h and run them on the ZigBit module.

For the next phase we will verify the baseline software operation and then modify the application layer to use our new hardware. In my_code.c disable the breakpoint compile and reload the firmware on two modules. Then remove the JTAG/ICE from the hardware and power down the hardware. If you have access to an IEEE802.15.4 packet analyzer monitor channel 20. Re-connect the power to one unit. You should see the LED flash to indicate it is scanning for an open channel. Then it will light continuously to show the PAN has been established. This first device has assumed the role of coordinator and is now waiting for other devices to join the pan. Next power-up the second board. The second board will flash a few times as it discovers and joins the PAN. Once the two devices have established the PAN they will synchronously blink about every 2 seconds. And there you have it , the Star Network Demo running on your own ZigBit hardware, yeah!

To top the project off I made a very simple program that demonstrates over-the-air communication between two modules. The two modules will set-up a PAN. The first module to boot will take on the coordinator role. The second module will become the device. These two will send a blink message about every two seconds, just like the default Star_nonbeacon example program. The modification adds the state of the senders shunt to the message payload. The receiver parses the message payload and enables its LED based on the senders shunt state. In other words, if the sender's shunt is in, the receiver's LED flashes. If the shunt is out, the receiver does not flash. Fairly simple proof the messages are getting through. A packet analyzer capture of the demo program is shown below.



Zigbit DIY.cap [Wireshark 1.6.4 (SVN Rev 39941 from /trunk-1.6)]

No.	Time	Source	Destination	Length	Sequence Number	Info	Data
1	0.000		Broadcast	84	174	Beacon Request	
2	4.251		Broadcast	84	206	Beacon Request	
3	8.497		Broadcast	84	238	Beacon Request	
4	18.423		Broadcast	84	174	Beacon Request	
5	18.426	0x0000		87	239	Beacon, Src: 0x0000	
6	20.286	11:11:11:11:11:11:11:11	0x0000	95	188	Association Request	
7	20.287			79	188	Ack	
8	20.787	11:11:11:11:11:11:11:11	0x0000	92	189	Data Request	
9	20.787			79	189	Ack	
10	20.789	ab:2d:c6:a8:9c:ef:46:2d	11:11:11:11:11:11:11:11	101	252	Association Response, PAN: 0xbabe A	
11	20.789			79	252	Ack	
12	22.816	0x0001	0x0000	86	190	Data, Dst: 0x0000, Src: 0x0001	01
13	22.816			79	190	Ack	
14	24.838	0x0001	0x0000	86	191	Data, Dst: 0x0000, Src: 0x0001	01
15	24.839			79	191	Ack	
16	26.859	0x0001	0x0000	86	192	Data, Dst: 0x0000, Src: 0x0001	01
17	26.859			79	192	Ack	
18	28.881	0x0001	0x0000	86	193	Data, Dst: 0x0000, Src: 0x0001	00
19	28.882			79	193	Ack	
20	30.905	0x0001	0x0000	86	194	Data, Dst: 0x0000, Src: 0x0001	00
21	30.905			79	194	Ack	
22	32.925	0x0001	0x0000	86	195	Data, Dst: 0x0000, Src: 0x0001	00
23	32.925			79	195	Ack	
24	34.946	0x0001	0x0000	86	196	Data, Dst: 0x0000, Src: 0x0001	01
25	34.946			79	196	Ack	
26	36.967	0x0001	0x0000	86	197	Data, Dst: 0x0000, Src: 0x0001	01
27	36.967			79	197	Ack	
28	38.987	0x0001	0x0000	86	198	Data, Dst: 0x0000, Src: 0x0001	01
29	38.988			79	198	Ack	
30	41.008	0x0001	0x0000	86	199	Data, Dst: 0x0000, Src: 0x0001	01

File: "C:\Documents and Settings\Ranger\Deskto... | Packets: 30 Displayed: 30 Marked: 0 Load time: 0:00.406 | Profile: Default

The first module broadcasts beacon requests to check for other IEEE802.15.4 networks in the vicinity. Then it establishes a PAN and waits for others to join (No

3). At 18.4 seconds the second device starts and sends out a beacon request. An association request is issued from the coordinator and they hook-up; binding is completed and short addresses are assigned. The PANID is 0xbabe, the coordinator is address 0x0000 and the device is address 0x0001. Once the pan is established the device sends status updates to the coordinator every 2 seconds. Note transactions 18, 20 and 22. This is where the shunt was removed and we can see the payload data has been changed from 01 to 00.

In summary this paper demonstrated how to use the Atmel ATZB-24-A2 module with user designed hardware. The IAR tool-chain configuration and MAC modifications were shown in great detail and all modified files were compartmentalized for illustration and reuse. The hardware and application were trivial examples but this process is not. It is a useful baseline for unlimited projects with ZigBit modules.