USER GUIDE

Wireless Production Test Reference

Atmel AT-WPTRB

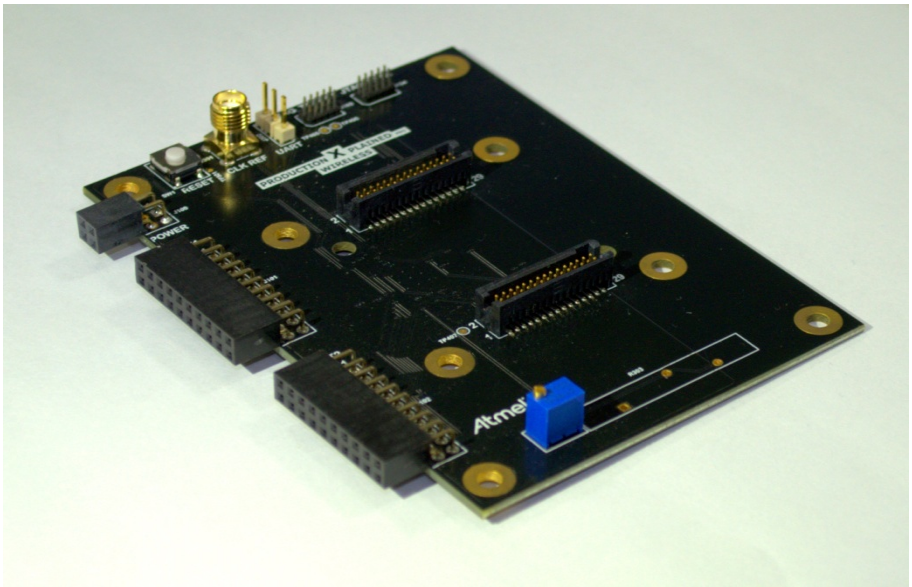


Table of Contents

1. Overview 3
2. Features 3
3. Abbreviations 3
4. Production Test Setup 4
   - 4.1 Hardware Overview 5
     - 4.1.1 Guidelines for Designing a Customer Adapter Board 8
       - 4.1.1.1 Hardware Guidelines 8
       - 4.1.1.2 Mechanical Guidelines 9
     - 4.1.2 Configure the Hardware for Production Test 10
       - 4.1.2.1 Prepare the Production Xplained Pro for Crystal Calibration 10
       - 4.1.2.2 Characterize Golden DUT's used for RF Path Verification of Test Setup: 10
   - 4.2 Firmware Overview 11
     - 4.2.1 Firmware Architecture 11
     - 4.2.2 Production Tests Supported in the WPTR Setup 12
       - 4.2.2.1 Hardware Test 12
       - 4.2.2.2 Current Measurement 12
       - 4.2.2.3 Crystal Calibration 12
       - 4.2.2.4 RF Test 13
       - 4.2.2.5 GPIO Test 13
     - 4.2.3 Porting Application Firmware to Customer Board 13
   - 4.3 Software Overview 14
   - 4.4 Running Production Tests 14
     - 4.4.1 Configuration 15
       - 4.4.1.1 Configuration of ZigBit USB and Xplained Pro 15
       - 4.4.1.2 Configuration of Tests 17
     - 4.4.2 Running Command-line Tool 17
5. Add a Custom Test 20
   - 5.1 Modifying Firmware 21
   - 5.2 Modifying XML Files 23
   - 5.3 Modifying Extension Class 25
   - 5.4 Writing Test Case 25
6. Customize Test Logger 27

Appendix A. Reference Links 28

Appendix B. Revision History 29

1. Overview

Wireless Production Test Reference (WPTR) is a complete production test setup solution offered for customers using the Atmel® wireless solutions including ZigBee® Transceivers and SoC in their product. Main purpose of this reference solution is to ensure faster ramp up of production test for customer's products in high volume manufacturing.

The tests that can be executed in the production site using this framework ensure to cover most of the RF circuitry functionality. With minimal efforts customers would be able to deploy production testing for their products. This test setup reduces the need of test equipment and also the test time required in high volume production.

2. Features

WPTR provides production ready setup consisting of hardware, firmware and software. The main features of the test setup are:

- Supports Atmel Wireless Transceivers and Single Chip Solutions
- Windows® command line tool for running tests
- Extensible framework
- Test cases that the test setup can support
  - Customer Product (or here on referred to as DUT) - Current measurement
  - Crystal calibration
  - Peripheral test
  - RSSI test
  - GPIO test – Detects shorted pins
  - 32kHz crystal test (In SoC devices only)

3. Abbreviations

WPTR - Wireless Production Test Reference

PDT - Production Test

DUT - Device Under Test

MCU - Microcontroller

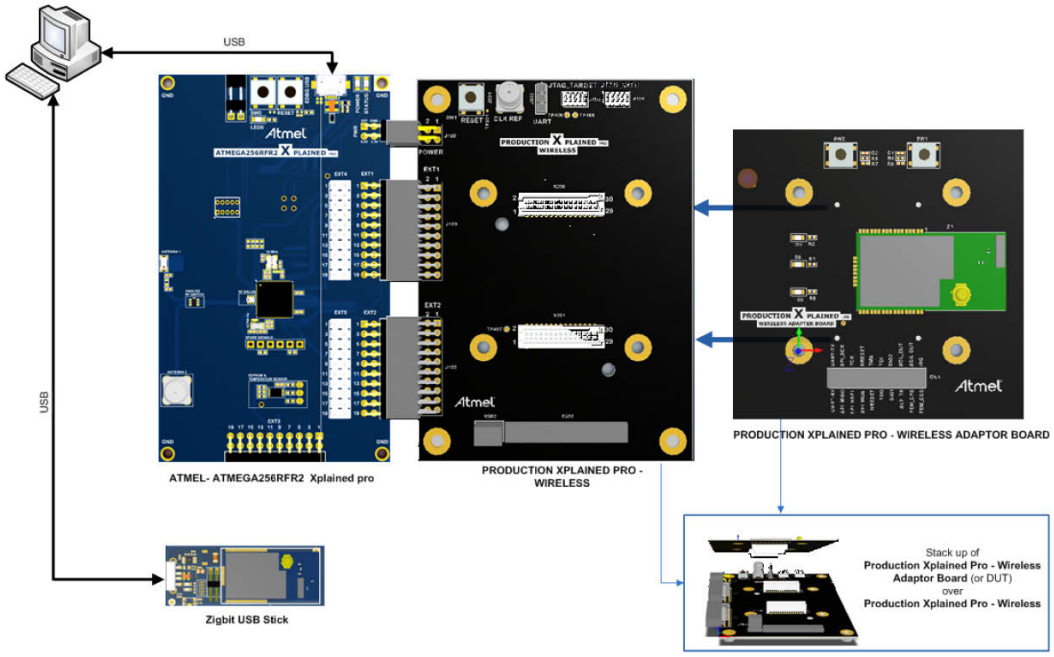
XPRO - Xplained Pro

4. Production Test Setup

A typical production test setup would comprise of the following elements:

- Hardware:
  - Production Xplained Pro board
  - ZigBit® USB Stick: ATZB-X-233-USB and ATZB-X-212B-USB
  - ATmega256RFR2 Xplained Pro Board: ATMEGA256RFR2-XPRO
  - Customer's board or DUT (in this document we will refer to Production Xplained Pro - Adapter board)
- Software:
  - Windows command line software: "pdt_runner.zip"
- Firmware:
  - WPTR Application Firmware: "wptr_firmware.zip"

Figure 4-1. An Overview of the Hardware Setup for Production Test of the DUT



ATmega256RFR2 Xplained Pro: The control board that interfaces the computer with the DUT and also the ATmega1284P controller.

In case of a DUT with on-board MCU, this control board would only perform as a controller between the DUT- MCU and the computer. But if the DUT does not have an on-board MCU, the ATmega256RFR2 on this control board acts as a baseband controller for the RF only transceiver devices.

Production Xplained Pro: This is a production test board that interfaces the Xplained Pro board and the DUT. The following interfaces are exposed:

1. JTAG interface of the controller used in the DUT.
2. ATmega1284P which is used as the controller to run most of the production tests:
   a. JTAG interface of ATmega1284P.
   b. Reset circuitry.
   c. Debug UART interface of the ATmega1284P controller.

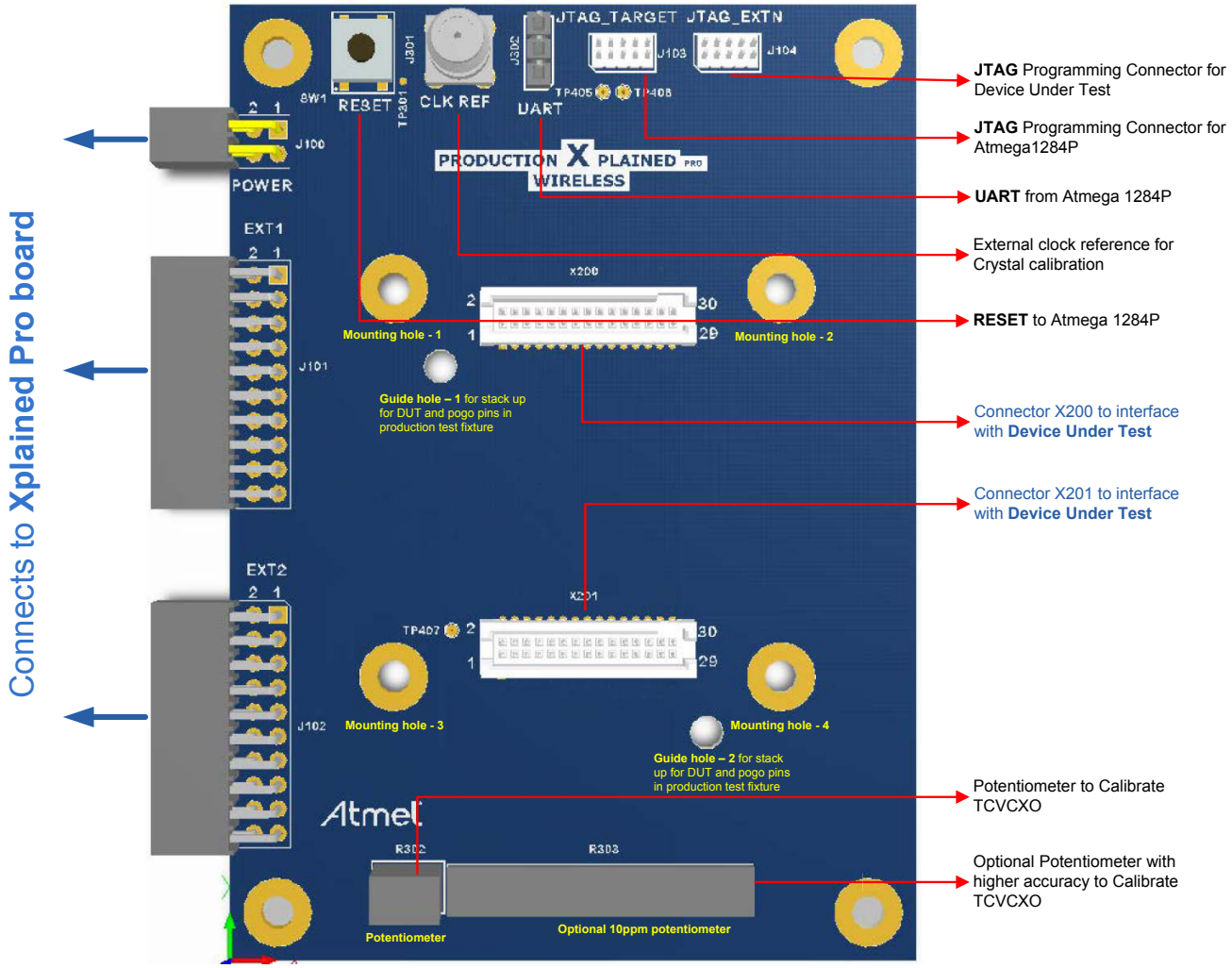
3. CLKREF SMA Connector used for tuning CFPT-126.
   CLKREF SMA Connector is used for measuring the reference clock frequency for crystal calibration and tunes it to the expected value.
4. 10kΩ potentiometer (R303) - Potentiometer R303 can be used to tune the clock output on the CLKREF SMA connector to 1.000065Hz. The accuracy of the DUT crystal calibration depends on this tuning.

Production Xplained Pro - Wireless Adaptor board: This is a reference DUT adaptor board with an ATmega256RFR2-Zigbit mounted to demonstrate the production test setup.

ZigBit USB Sticks: This is the 2.4GHz or SubGHz ZigBit USB Stick used for RF testing of the DUT.

4.1 Hardware Overview

Figure 4-2. Top Side of the Board



X200 and X201 connectors used on the board is Samtec board-to-board connector, part number: TFM-115-02-SM-D-LC (CONNECTOR TFM, 2X15 PINS, SMD 1.27 Pitch).

Refer to Table 4-1 and Table 4-2 for pinout details of the board-to-board connectors X200 and X201.

Table 4-1. Board-to-board Connector X200

Pin number	Pin description	Pin number	Pin description
Pin 1	GPIO pin short test	Pin 2	VCC_TARGET_P3V3
Pin 3	GPIO pin short test	Pin 4	VCC_ZB_P3V3
Pin 5	GPIO pin short test	Pin 6	VCC_CS_P3V3
Pin 7	GPIO pin short test	Pin 8	No Connection
Pin 9	GPIO pin short test	Pin 10	No Connection
Pin 11	GPIO pin short test	Pin 12	No Connection
Pin 13	GPIO pin short test	Pin 14	No Connection
Pin 15	GPIO pin short test	Pin 16	No Connection
Pin 17	GPIO pin short test	Pin 18	JTAG - NRESET
Pin 19	GPIO pin short test	Pin 20	JTAG –TCK
Pin 21	GPIO pin short test	Pin 22	JTAG –TDI
Pin 23	GPIO pin short test	Pin 24	JTAG –TDO
Pin 25	GPIO pin short test	Pin 26	JTAG –TMS
Pin 27	GPIO pin short test	Pin 28	Ground
Pin 29	GPIO pin short test	Pin 30	GPIO pin short test

Table 4-2. Board-to-board Connector X201

Pin number	Pin description	Pin number	Pin description
Pin 1	VCC_TARGET_3V3	Pin 2	CLKO
Pin 3	SDA_DUT	Pin 4	SCL_DUT
Pin 5	UART_TX	Pin 6	UART_RX
Pin 7	SPI_NSEL	Pin 8	SPI_MISO
Pin 9	SPI_MOSI	Pin 10	SPI_SCK
Pin 11	NRESET	Pin 12	IRQ
Pin 13	SLP_TR	Pin 14	FEM_CSD
Pin 15	FEM_CPS	Pin 16	DIG1
Pin 17	DIG2	Pin 18	DIG3
Pin 19	DIG4	Pin 20	GPIO1
Pin 21	GPIO2	Pin 22	GPIO3
Pin 23	GPIO4	Pin 24	GPIO5
Pin 25	Ground	Pin 26	Ground
Pin 27	Ground	Pin 28	Ground
Pin 29	Ground	Pin 30	Ground

Figure 4-3. Bottom Side of the Board

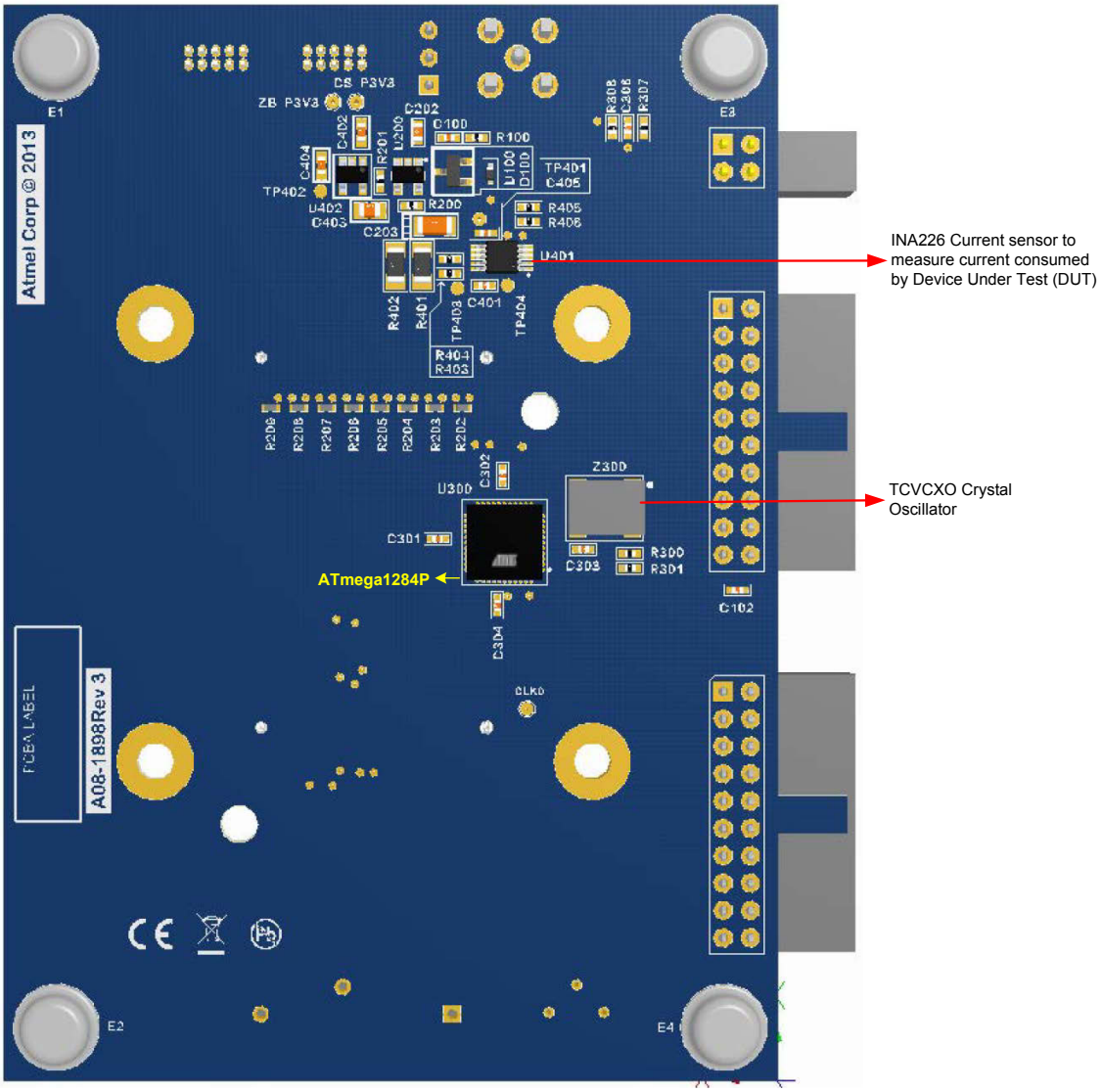
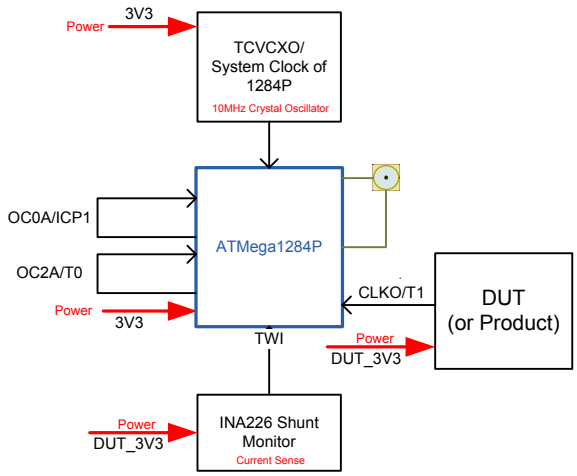


Figure 4-4. Crystal Calibration and Current Measurement using ATmega1284P



DUT 16MHz Crystal Calibrations:

10MHz CFPT-126 Temperature Compensated Voltage Controlled crystal Oscillator (TCVCXO) providing a high degree of frequency stability over a wide temperature range is used to generate the reference clock for crystal calibration using ATmega1284P. 10MHz clock output from CFPT-126 is used as system clock by the ATmega1284P.

A low frequency reference clock of 1.000065Hz is generated by cascading two timers, Timer 2 and Timer 0, of ATmega1284P. Timer 2 module generates a low frequency clock from system clock by using its Output Compare module and outputs it on Output Compare pin OC2A. This signal is used to clock the Timer module Timer 0.

Timer 0 module again uses its Output Compare unit to generate a 1.000065Hz signal on it OC0A pin.

This signal is available on the CLKREF SMA connector for further tuning.

Crystal calibration is done using the Timer 1 module of ATmega1284P, which is clocked from the 4MHz CLKM from the DUT. Low frequency signal 1.000065Hz is given to the Input Capture pin of the Timer 1. Timer 1 counts between the rising edges of this signal, is used to calibrate the DUT 16MHz crystal.

Current Measurement:

INA226 Current/Power monitor with I^2C interface is used to measure the current consumed by the DUT. ATmega1284 can read and configure the INA226 registers using its TWI interface.

4.1.1 Guidelines for Designing a Customer Adapter Board

4.1.1.1 Hardware Guidelines

During the hardware design of the adapter board or DUT, the below mentioned signals, power lines, and GPIOs should be made available to connect with Production XPRO for executing the mentioned tests (Section 4.2.2) in this document.

GPIO Pin Short Test: The pins described as GPIO Pin Short test are used to test the continuity of the GPIOs and short between adjacent pins. Connect the GPIO's that you want to test between adjacent GPIO Pin Short pins.

For example, to test the GPIOs PB2 and PB3; connect PB2 to Pin No. 1 of X200 and PB3 to Pin No. 2 of X200. (Refer to the schematics in Appendix.)

VCC_CS_P3V3: 3.3V power supply to the DUT. In order to measure the current consumed by the DUT, the DUT has to be powered from this power supply on the Production XPRO.

VCC_TARGET_P3V3: 3.3V power supply to ATmega1284P on the Production XPRO.

VCC_ZB_P3V3: 3.3V power supply to the DUT before the current sense resistors. This can be used if current measurement is not required.

JTAG: JTAG lines of the MCU used in the DUT for programming and debugging.

CLKO: Connect the CLKO to the Clock output of the DUT. This clock output should be 4MHz and is used to calibrate the 16MHz crystal oscillator

SDA_DUT and **SCL_DUT**: Used to check the TWI peripheral of the DUT. Connect to the TWI lines of DUT accordingly.

UART_RX: Connect to the UART RX pin of the MCU in the DUT.

UART_TX: Connect to the UART TX pin of the MCU in the DUT.

SPI Bus: Connect to the SPI peripheral of the DUT. The DUT acts as a slave module in this reference test setup.

NRESET: Connect to the RESET pin of the DUT.

IRQ: Connect to the IRQ pin of the DUT. DUT interrupts the ATmega256RFR2 Xplained Pro when the data is to be read.

SLP_TR: Connect to the SLP_TR pin of the RF only Transceivers.

FEM_CSD: Connect to the Chip Select pin of the FEM module if present.

FEM_CPS: Connect to the Mode Control pin of the FEM module if used.

DIG1, DIG2, DIG3, and DIG4: Connect to the DIG line DIG1, DIG2, DIG3, and DIG4 of the DUT.

GPIO signals (GPIO1- GPIO5): Five GPIO pins of the ATmega1284P are available as spare GPIO pins that the customers can use for their requirement.

4.1.1.2 Mechanical Guidelines

Figure 4-5. Reference Stack Up of Boards and Other Mechanicals for Production Test

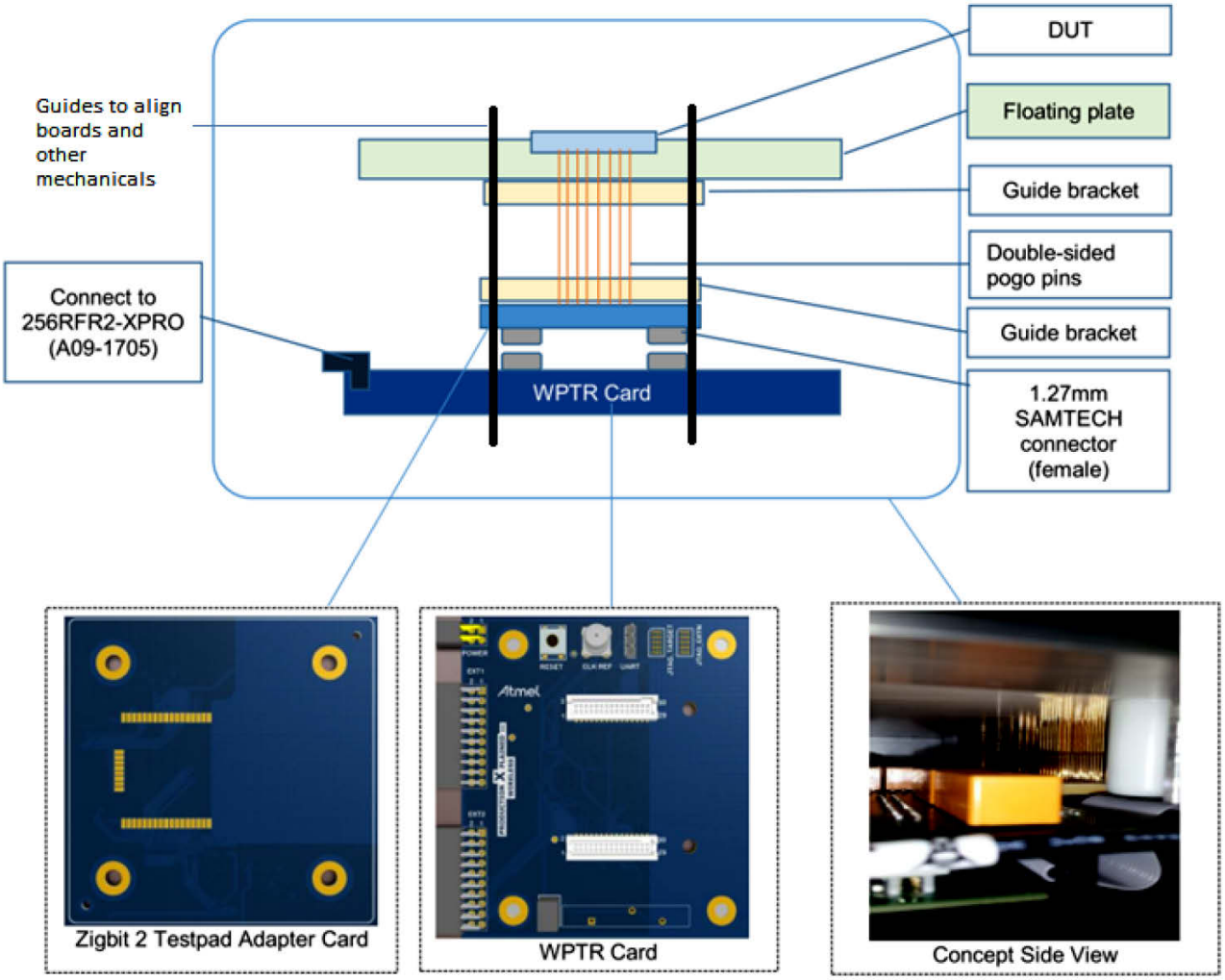


Figure 4-5 shows the example production assembly of boards. An adapter board is mounted onto the WPTR and customer DUT is placed on the floating plate and the test pads on this DUT connect to the adaptor board through the pogo pins.

- Guide holes and mount holes: Provide the Guide and Mount holes in the adapter board in line with the Production XPRO; these are used to ensure the stack-up of boards is aligned for proper contact. For better understanding and more details of how to provide these holes; refer to the layout of Production Xplained Pro-Wireless Adapter board
- Guides: Are used to keep the boards and other mechanicals used in the wireless production test setup aligned with higher accuracy
- Pogo Pins: Double sided pogo pins can be used to interface adaptor board and customer DUT

- Board to board connector: Use Samtec receptacle connector - SFM-115-L2-S-D-LC (Receptacle SFM, 2X15 PINS, SMD 1.27 Pitch)
- Floating plate: A plate which would hold the customer DUT and expose only the test pad beneath the DUT to make contact with the pogo pins

4.1.2 Configure the Hardware for Production Test

4.1.2.1 Prepare the Production Xplained Pro for Crystal Calibration

As explained above the ATmega1284P on the Production Xplained Pro outputs a 1.000065Hz clock output on the CLKREF SMA connector which is used as reference for the DUT crystal calibration. Either the 10kΩ potentiometer (R303) or the optional high precision potentiometer (R304 - not mounted) can be used for tuning the CLKREF clock output to the expected value of 1.000065Hz.

Connect a frequency counter to the CLKREF SMA connector and measure the frequency. Adjust the potentiometer until the measured frequency equals 1.000065Hz.

It is advised that the tuning of CLKREF clock output is done for every batch run as the accuracy of the crystal calibration depends on this tuning.

4.1.2.2 Characterize Golden DUT's used for RF Path Verification of Test Setup:

Follow the below steps to identify and characterize a customer's Golden DUT from the production lot to verify the Test setup.

1. Tx Power Characterization: DUT is configured to transmit data at each defined output power level. Calibrated measurement equipment like Spectrum Analyzer should be used to determine the real output power from the DUT at each defined level.

The measured values should then be compared to the specification and DUT with the closest value should be chosen.

Table 4-3. Example Measurement Table

Power level register value	Power level from spec [dBm]	Measured power level [dBm]
0x0	3.5	
0x1	3.5	
-	-	
-	-	
0xF	-16.5	

2. Rx Sensitivity Characterization: Rx RSSI sensitivity is measured by applying a range of known input power levels, and reading the RSSI values from the DUT. The input power levels must be applied using a calibrated RF generator.

Table 4-4. Example Measurement Table

Power level from generator [dBm]	RSSI level register value
0	
-10	
-20	
-30	

Run the RF Test (refer to Section 4.2.2) using this Golden DUT in this setup at different Tx power level and log the RSSI test results.

Table 4-5. Example Measurement Table

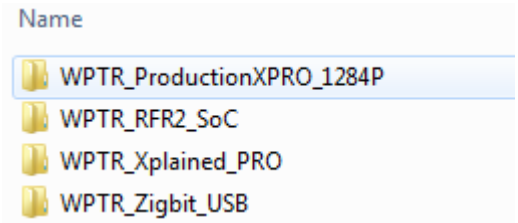
Serial number	Power level register value	Rx RSSI [dBm]	Tx RSSI [dBm]
1	0x0		
2	0x1		

Whenever there is a shift in the RSSI window used for RF Test for a particular test setup, use the Golden DUT to verify the setup using the logged results.

4.2 Firmware Overview

WPTR provides firmware examples based on the new generation ZigBit Wireless modules. Firmware examples are provided as standalone archive containing the applications shown in **Error! Reference source not found.**.

Figure 4-6. Folder Structure



WPTR_Zigbit_USB contains the application firmware for ZigBit USB Sticks. Fuse settings for ATxmega256A3U on ZigBit USB: 0xFFFEFFFF00FF.

WPTR_ProductionXPRO_1284P contains the application firmware running on ATmega1284P controller in the Production Xplained Pro. Fuse settings for ATmega1284P: 0xFF99E0.

WPTR_RFR2_SoC - contains the application firmware demonstrating WPTR on DUT, in this case the ATmega256RFR2 ZigBit Wireless Module. Fuse settings for ATmega256RFR2 on DUT: 0xFF9998.

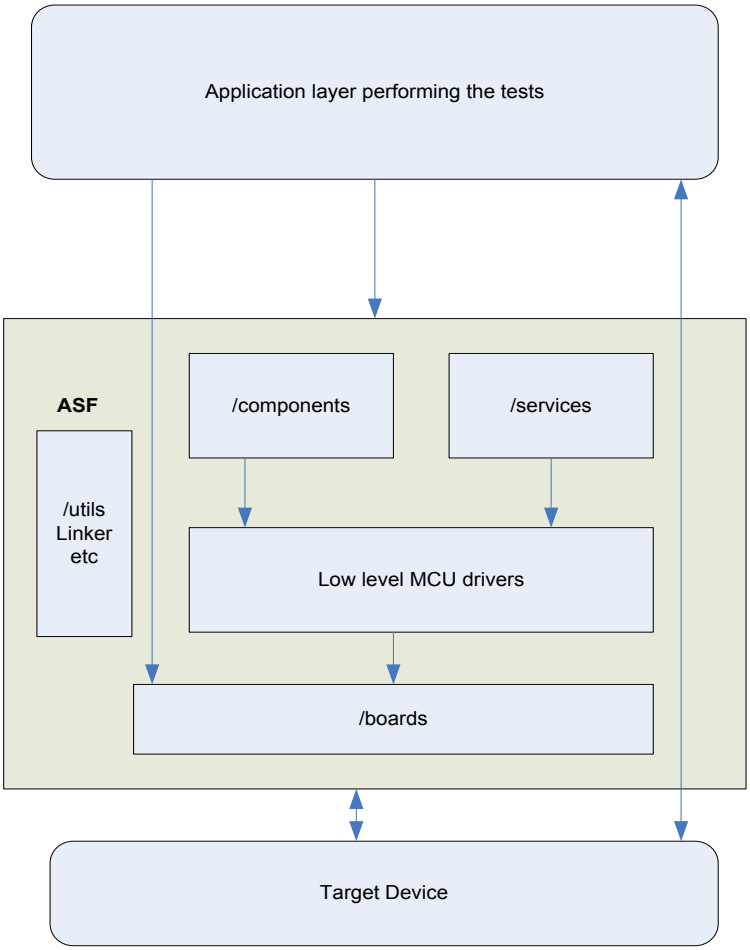
WPTR_Xplained_PRO - contains the application firmware running on the ATmega256RFR2 Xplained Pro. The application also demonstrates the use of WPTR on ZigBit RF only modules such as the AT86RF233 Amplified ZigBit Wireless Module. Fuse settings for ATmega256RFR2 on Xplained PRO: 0xFF99E2.

Atmel Studio Project files are provided for each application along with the archive. The application firmware is tested for Atmel GCC Tool Chain.

4.2.1 Firmware Architecture

WPTR firmware applications are based on the Atmel Software Framework, which has been designed to help develop and glue together the different components of a Software design. Figure 4-7 explains the ASF structure.

Figure 4-7. ASF Architecture



Refer to http://asf.atmel.com/docs/latest/ for further details on Atmel Software Framework.

4.2.2 Production Tests Supported in the WPTR Setup

4.2.2.1 Hardware Test

Peripherals, such as SPI, UART, and TWI of the SoC devices are verified by conducting a simple echo mode test with the master controller on ATmega256RFR2 Xplained Pro. Also the 32kHz crystal oscillator connected to the SoC devices is tested.

In case of RF only transceivers SPI peripheral as well as the GPIOs (RESET, SLP_TR, IRQ) used for transceiver communication are tested.

4.2.2.2 Current Measurement

The current consumed by the DUT in TRX_OFF state is measured using INA226 Current Shunt monitor, which is present on the Production Xplained Pro.

4.2.2.3 Crystal Calibration

All the Atmel Transceivers have an option to control the internal capacitance array connected to XTAL1 and XTAL2 using XTAL_TRIM bits in XOSC_CTRL register. Capacitance of 0pF to 4.5pF with 0.3pF resolution can be attained.

This test gives you the best XTAL_TRIM value with the CLKM frequency measured at that trim value.

The user can store the XTAL TRIM value in a persistent memory on the DUT for future reference.

4.2.2.4 RF Test

The RF performance of the DUT is verified by doing a simple Transmit test with the ZigBit USB Sticks and comparing the RSSI values with the expected range of values.

Steps executed in RF Test:

- DUT transmits a packet of data to the ZigBit USB
- ZigBit USB responds back the same packet along with its RSSI value (TX RSSI)
- DUT receives the packet from ZigBit USB and verifies the received packet against the transmitted packet
- If both the transmitted and received packet in DUT are the same, DUT gets its RSSI value (RX RSSI)
- Finally both the TX RSSI and RX RSSI are reported back to the Test PC

4.2.2.5 GPIO Test

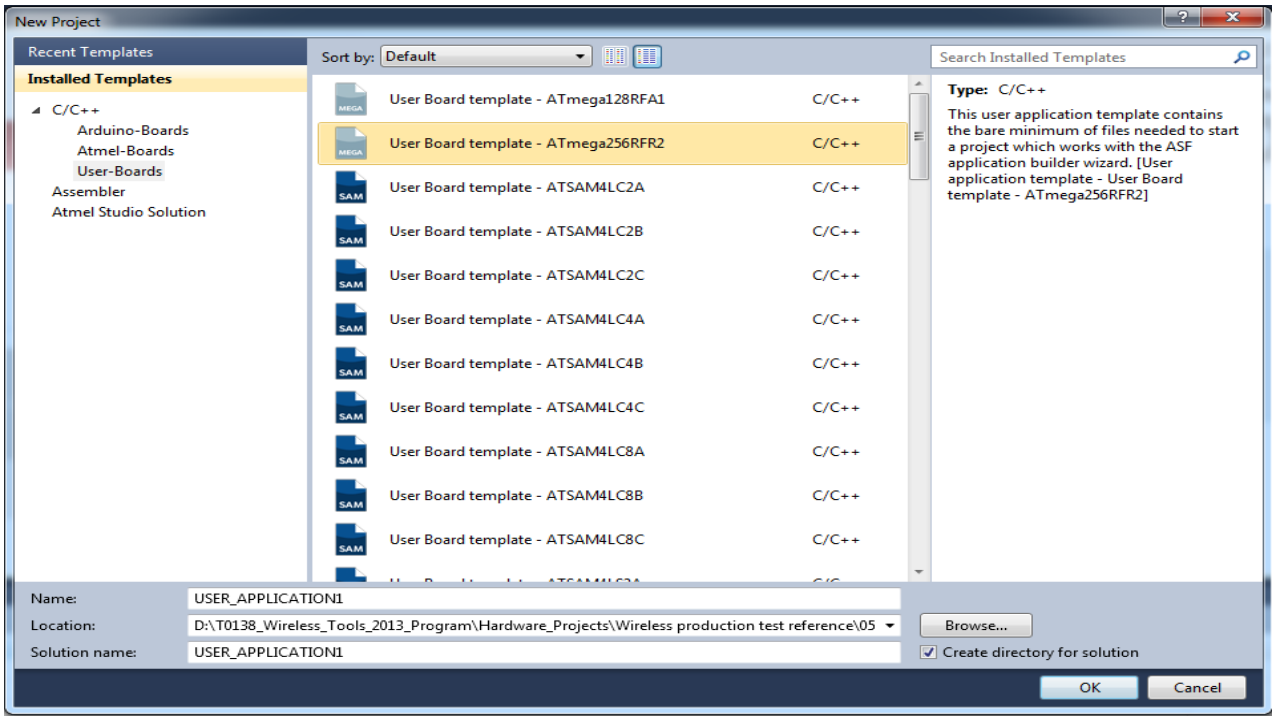
Pin short test and continuity test are done on all the required GPIOs of the SoC devices.

4.2.3 Porting Application Firmware to Customer Board

If the customer has designed his own board containing an Atmel SoC Transceiver, Atmel MCU, and a RF only Transceiver, follow the below steps to port the application firmware to the new hardware:

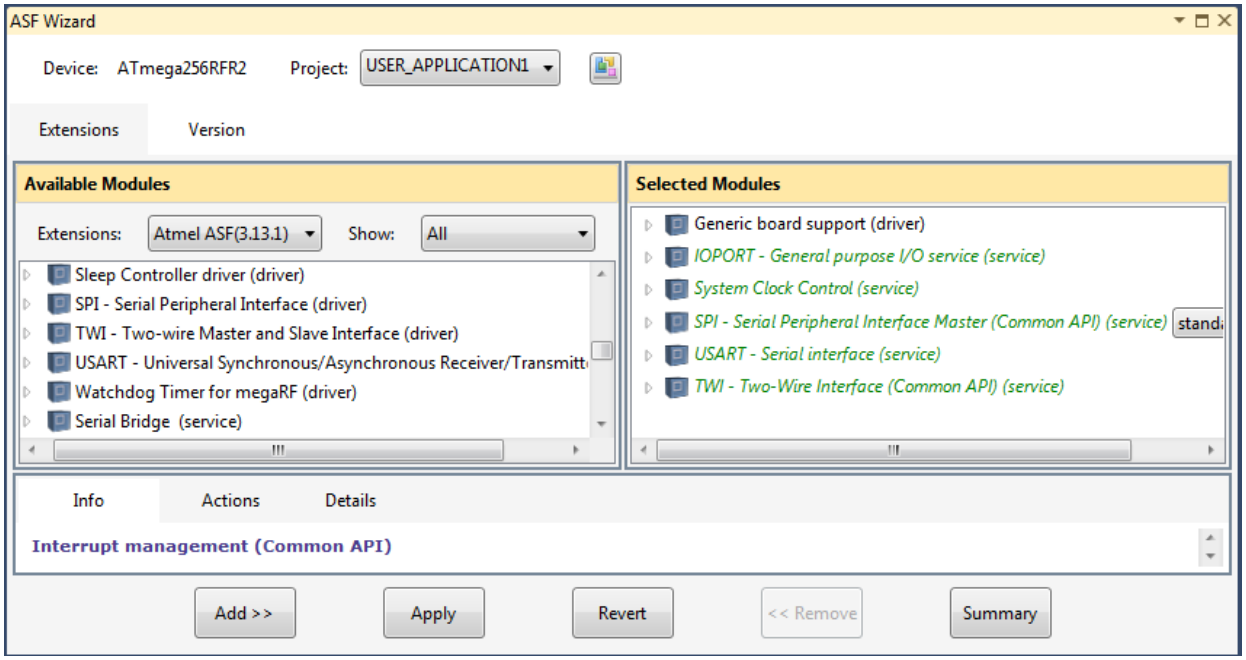
1. Open Atmel Studio and Create a New Project from the File menu.
2. As shown in Figure 4-8, click on User-Boards if the customer board is not supported by Atmel Studio, select the User Board template based on the Atmel MCU, and click 'OK'.

Figure 4-8. New Project



3. This will create a new project in Atmel Studio based on ASF.
4. As shown in Figure 4-9, use the ASF wizard to add the required modules for the application.

Figure 4-9. ASF Wizard



5. Depending on the customer board modify the header files such as “user_board.h” and config. header files under /src/config which helps you configure the System clock and the peripherals under use.
6. Modify the board_init function in “boards/user_board/init.c” to initialize the customer board based on the requirement.
7. Import the application specific source files from “src/src” and “src/inc” of the demo application firmware “WPTR_RFR2_SoC” to the example project.
8. Modify the “app_config.h” header file for customizing the application as per the requirements.
9. Tests mentioned in Section 4.2.2 are implemented in soc_dut_tests.c. The customer can customize the already existing tests based on his requirement.
10. RF functionality is implemented in rf_trx.c. When modifying to any other Atmel transceiver, customize rf_trx.c and underlying transceiver access layer: “trx_access.h” and “trx_access.c”.
11. If the customer wants to add any new test, follow the instructions in Chapter 5 “Add a Custom Test” to add a new custom test to the application.

4.3 Software Overview

WPTR software contains a Windows command line tool, which controls the hardware modules and executes test cases against DUT. Following are the requirements for running the software:

- Windows XP or later (Win 7, Win 8)
- Microsoft® Visual C++® 2008 Redistributable Package (vcredist_x86.exe)

4.4 Running Production Tests

WPTR software is bundled in a zip file called ‘pdt_runner.zip’. Extracting the zip file leads to a folder structure as shown in Figure 4-10.

Figure 4-10. Folder Structure

Name	Date modified	Type	Size
out	29-01-2014 16:29	File folder	
src	29-01-2014 16:29	File folder	
tests	29-01-2014 16:29	File folder	
xml	29-01-2014 16:29	File folder	
config.cfg	17-01-2014 10:58	CFG File	1 KB
pdt_runner.exe	17-01-2014 12:05	Application	10,087 KB

'pdt_runner.exe' is the command-line tool. It depends on 'config.cfg' for all the configurations required for its execution. Folders present inside the zip file are organized with respect to contents, for example the 'tests' folder contains all the test cases written in Python. These folders and their contents will be explained in detailed in the following sections.

4.4.1 Configuration

'config.cfg' has configurations in JSON format and is being used by 'pdt_runner.exe'. It should be provided with required information before executing pdt_runner. The following sections will explain configurations in detail.

4.4.1.1 Configuration of ZigBit USB and Xplained Pro

The configuration details pertaining to ZigBit USB stick and ATmega256RFR2 Xplained Pro boards in the config.cfg file should be updated, for the pdt_runner.exe to identify and connect to them. The configuration is as shown in Figure 4-11.

Figure 4-11. Configuration

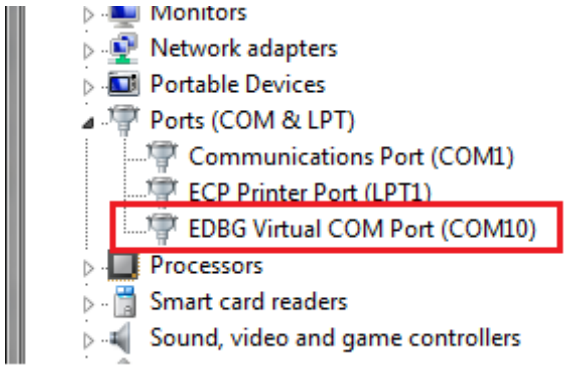
```
xpro:
{
    PortName        :   "COM4"
    BaudRate        :   9600
    TimeOut         :   1
    verbose         :   2
    ExtClass        :   "xpro_controller.XproControllerExt"
}

zigbit_usb:
{
    PortName        :   "COM2"
    BaudRate        :   9600
    TimeOut         :   1
    verbose         :   2
    ExtClass        :   "zigbit_usb.ZigbitUsbExt"
}
```

In Figure 4-11:

- '**xpro**' corresponds to the configuration of ATmega256RFR2 Xplained Pro board
- '**PortName**' corresponds to the EDBG Virtual COM Port enumerated by the embedded debugger present on ATmega256RFR2 Xplained Pro board. This can be obtained from the "Device Manager" in Windows as shown in Figure 4-12

Figure 4-12. Device Manager



- '**BaudRate**' refers to the speed with which the pdt_runner.exe communicates with the Xplained Pro board through the COM port
- '**TimeOut**' corresponds to the read timeout (in seconds) of the serial communication. Internally, the pdt_runner.exe has a retry of 20, so the read timeout is multiplied twenty times resulting in total read timeout of 20 seconds, which is far enough for any test cases to respond properly
- '**Verbose**' controls the verbosity of the logs pertaining to the communication with the ATmega256RFR2 Xplained Pro. It ranges from 0 to 5, with 5 being most verbose. A verbose value of 0 or 1 is recommended for production run. Verbose values greater than 2 can be used during debugging
- '**ExtClass**' an extension class which extends the API available for the 'xpro' object. In order to control the Xplained Pro board, there is an object called 'xpro' provided for each test case. The test case can in turn call the functions/methods available under the xpro object. An extension class should be specified as follows

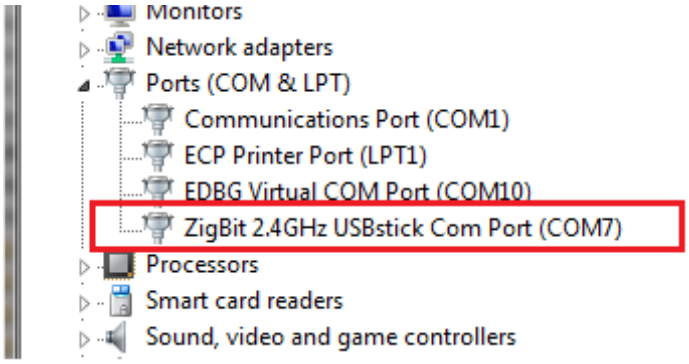
"*<python file name>.<class name>*"

Where 'python file name' is the name of the Python file without the '.py' extension and this file should be placed in the 'src' folder and 'class name' is the name of the class defined in that file. To get an understanding of how an extension class looks like, refer to '*xpro_controller.py*' available in 'src' folder. 'xpro_controller.py' is the default extension Python file provided for Xplained Pro board. More information on extension class will be discussed under the section <Add a Custom Test>

'**zigbit_usb**' corresponds to the configuration of ZigBit USB Stick.

- '**PortName**' – ZigBit USB also enumerates as Virtual COM port on windows. Figure 4-13 shows how it looks on 'device manger'

Figure 4-13. Device Manager



- ‘**ExtClass**’ – Similar to Xplained Pro board, the ZigBit USB is controlled from each test case using the object ‘zigbit_usb’. The extension class specified under ZigBit USB configuration extends the functions available in the ‘zigbit_usb’ object. By default a file called ‘zigbit_dut.py’ is provided as extension file for ZigBit USB Stick under the ‘src’ folder. More information on extension class will be discussed in Chapter 5 ‘Add a Custom Test’

Other properties found under ‘zigbit_usb’ configuration are similar to Xplained Pro boards’ and the explanation provided for ‘xpro’ configuration applies to it.

4.4.1.2 Configuration of Tests

Configurations pertaining to tests are ‘test path’ and ‘test suite’ information. Figure 4-14 shows a typical example.

Figure 4-14. Typical Example of Test Configuration

```

testpath:   "./tests/"

testsuite:
{
    Suite1:
    {
        SetUp       :   "zigbit_setup.py"
        CleanUp     :   "zigbit_cleanup.py"

        Tests       :   [ "zigbit_selftest.py"
                          "zigbit_demo_fw_test.py"
                        ]
    }

}

```

- ‘**testpath**’ – Folder path where all test cases (*.py files) exists. By default it refers to the ‘tests’ folder which exists in the same path as pdt_runner.exe
- ‘**testsuite**’ – Test suite option describes the tests to be executed against the DUT. A test suite is defined as a group of test cases along with setup and cleanup functions:
  - ‘Setup’ test file is executed once before running any of the test cases, if the setup encounters any error the whole test suite will be aborted
  - ‘Cleanup’ test file is executed after all the test cases are run, it will be run even if any of the test cases encounters error thereby cleaning up the test session
  - ‘Tests’ corresponds to list of test cases and each test case is a Python file existing under ‘testpath’

As shown in Figure 4-14, the ‘Suite1’ is the name of the test suite, under which ‘SetUp’, ‘CleanUp’, and ‘Tests’ options are available. ‘SetUp’ and ‘CleanUp’ are optional, so if no setup or cleanup function is available, they can be left blank like “” (empty double quotes). Each test case is a Python file and has to be written by following a specific format. This is discussed in detail under Section 5.4 ‘Writing Test Case’.

4.4.2 Running Command-line Tool

pdt_runner.exe is the command line tool that interacts with ZigBit USB Stick and ATmega256RFR2 Xplained Pro board and executes all test cases (Python files) against the DUT. Invoking pdt_runner.exe with ‘-h’ options prints out all the options supported by it. Figure 4-15 shows the invocation of help (‘-h’) option.

Figure 4-15. pdt_runner command line options

```
Administrator: C:\windows\system32\cmd.exe

D:\pdt_runner>pdt_runner.exe -h

Usage: pdt_runner.exe [OPTIONS]

Options:
  -h, --help            show this help message and exit
  -c <config file>, --config-file=<config file>
                        Configuration info in a file; Default is 'config.cfg'
  -p <tests path>, --test-path=<tests path>
                        Directory containing test files; Default is './tests/'
  -g, --generate-api    Generates API files from Msgset XML data
  -t <test>, --test=<test>
                        Single test to be executed
  -s <serial number>, --serial-number=<serial number>
                        Serial number/Unique ID of the DUT
  -v VERBOSE, --verbose=VERBOSE
                        Verbosity, (0-3) 3 being more verbose

Example: pdt_runner -c config.cfg -v 3

D:\pdt_runner>_
```

When invoked without options pdt_runner.exe uses ‘config.cfg’ as the default configuration file found in the same path and runs with verbosity of 1. A custom configuration file can be provided using the ‘-c’ option. The configuration file should have all the properties discussed in Section 4.4.1 ‘Configuration’.

Option ‘-s’ can be used to provide serial number of the DUT. It is used to log test output against a particular DUT.

Figure 4-16. Typical Example of Running Test

```
Administrator: C:\windows\system32\cmd.exe
 >
 > Running test - 'zigbit_power_test'
 >          Name: ZigBit Power Test
 >   Description: Checks DUT's current consumption
 >
Measuring DUT's current
Current consumption in limits
Test Ok
 > Time taken: 0.01 seconds
'zigbit_power_test' ... pass
----------------------------------------------------------------------
 > Running test - 'zigbit_selftest'
 >          Name: ZigBit Selftest
 >   Description: Tests peripheral, GPIO, crystal and RF
 >
HW Test...
Ok
GPIO Test...
Ok
Crystal calibration...
Ok
RSSI Test...
RSSI withing limits
Ok
 > Time taken: 0.01 seconds
'zigbit_selftest' ... pass
----------------------------------------------------------------------
 > Running test - 'zigbit_demo_fw_test'
 >          Name: ZigBit Demo Fw Test
 >   Description: Programs the demo firmware and tests it.
 >
Programming demo firmware...
Ok
Testing demo firmware...
Ok
 > Time taken: 0.01 seconds
'zigbit_demo_fw_test' ... pass
----------------------------------------------------------------------
 > Running cleanup - 'zigbit_cleanup'
Cleaning up...
Ok
----------------------------------------------------------------------
 > Test Result:
 > Suite - 'ZigBitTestSuite'
 > Tests:
    'zigbit_power_test' ... pass
    'zigbit_selftest' ... pass
    'zigbit_demo_fw_test' ... pass
 > Cleanup:
    'zigbit_cleanup' ... pass

-------------------- TEST PASSED -------------------------------------

D:\pdt_runner>
```

The standard console output shows execution steps and the debug prints from the test cases. At the end the consolidated test report is displayed.

Instead of running the whole test suite provided in the configuration file, the user can choose to run a single test case (test file) by using the '-t' option. This test case file must be present in the test path, which is provided in the configuration file or in the command line option '–p' <test path>.

A typical example of running single test case is as follows:

pdt_runner.exe –t "zigbit_selftest.py"

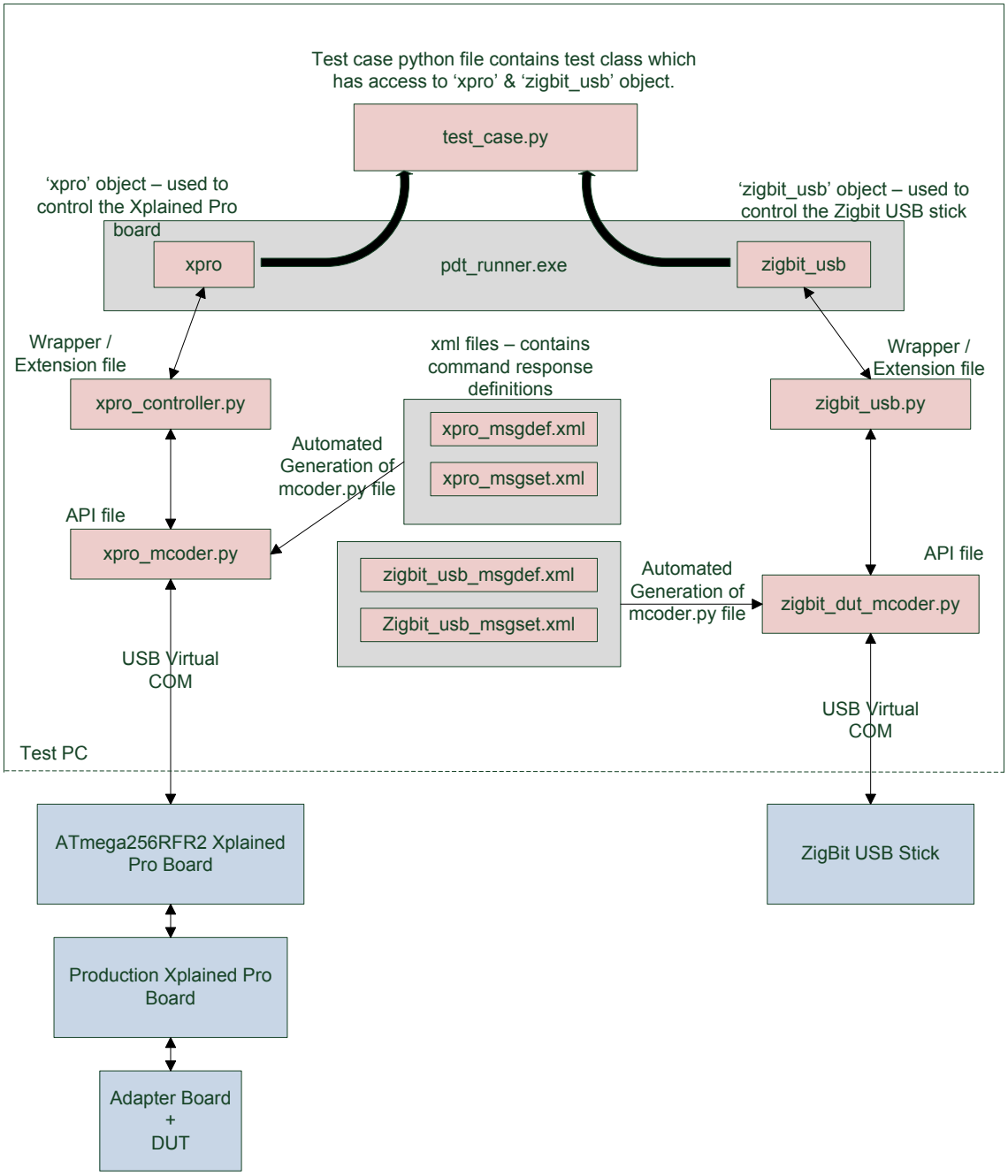
5. Add a Custom Test

Adding a custom test to production test reference involves the following steps:

- Modifying/appending firmware for ATmega256RFR2 Xplained Pro and DUT
- Modifying Xplained Pro specific XML file
- Writing test cases following a specific format

Figure 5-1 gives an insight of the interaction between 'pdt_runner.exe', test cases, and the WPTR hardware modules for adding a custom test case.

Figure 5-1. Interaction Between 'pdt_runner.exe', Test Cases, and the WPTR Hardware Modules



Here pdt_runner communicates with ATmega256RFR2 Xplained Pro and ZigBit USB stick using commands and responses called messages. These messages follow a simple protocol structure, which is discussed in detail in a separate document “WPTR Protocol Specification Document”.

There are two types of messages called ‘request’ and ‘confirm’:

- ‘Request’ refers to commands sent to hardware
- ‘Confirm’ refers to responses received from hardware

Adding new test requires new messages to be defined both in Xplained Pro firmware and in the XML files of software framework. The following sections explain the sequence of steps in detail.

5.1 Modifying Firmware

- “wptr_msg_const.h” under “src/inc” in WPTR_Xplained_PRO_RFR2 and WPTR_RFR2_SoC should be appended with new message ID used for communication between Test PC and Xplained Pro, and between Xplained Pro and the DUT.

Figure 5-2. Wptr_msg_const.h Containing Hardware Test Request and Confirm Messages

```
enum msg_code {
    /* Command requests */

    /* From Test PC to Xplained PRO */
    HWTEST_REQ                          =     (0x59),
    /* Xplained PRO & DUT*/
    RUN_HWTEST_REQ                      =     (0x03),

    /* Confirms and Indications */

    /* Xplained PRO to Test PC */
    HWTEST_CONFIRM                      =     (0x79),
    /* DUT to Xplained PRO*/
    RUN_HWTEST_CONFIRM                  =     (0x13),
}
SHORTENUM;
```

- Add support for this new message ID in “serial_handler.c” under function “handle_incoming_msg()” for both WPTR_Xplained_PRO_RFR2 and WPTR_RFR2_SoC

Figure 5-3. handle_incoming_msg() in serial_handler.c

```c
/**
 * \brief Parses the Received Data in the Buffer and Process the Commands
 * accordingly.
 */
static inline void handle_incoming_msg(void)
{
    /* Check for protocol id is Perofomance Analyzer */
    if (PROTOCOL_ID != sio_rx_buf[1])
    { /* protocol id */
        return;
    }
    /* Process the commands */
    switch (sio_rx_buf[MESSAGE_ID_POS])
    {   /* message id */

        case HWTEST_REQ:
        {
            /* Run the HW test including UART, TWI, 32kHz crystal test */
            test_functions.func_hwtest();
        }
        break;

        default:
        {

        }
    }
}
```

- These below files in the WPTR_Xplained_Pro_RFR2 are to be modified accordingly:

mcudut_tests.c: Include functions related to the tests executed in MCU of the DUT. **production_xpro_1284p_tests.c**: Include functions related to the tests executed in ATmega1284P of Production XPRO.

rfonly_dut_tests.c: Include functions related to the RF only-DUT tests.

- Depending on the test that is being executed, the relevant functions should be added in the files mentioned above

Figure 5-4. Example Test Case for MCU DUT

```c
/**
* \brief Write the RF Register power and channel
* \param rf_channel - Channel
* \param  rf_power - Power value of the register
*
*/

void mcu_config_rf_param(uint8_t channel, uint8_t power)
{
    /* Create the command format with msg id & payload*/
    cmd_dut.msg_id = SET_RFPARAM_REQ;
    *cmd_dut.ptr_payload++ = power;
    *cmd_dut.ptr_payload++ = channel;

    cmd_dut.ptr_payload -= 2;
    /* Update the length in the protocol format */
    cmd_dut.length = PROTOCOL_ID_LEN + MSG_ID_LEN + 2;
    /* Send the command via SPI */
    spi_send_cmd(&cmd_dut, &spi_dut_device_conf);
    /* Get the response back from SPI */
    get_response(&response_dut, &spi_dut_device_conf);
    /* Send the data via serial interface */
    send_response(RF_PARAM_CONFIRM, response_dut.ptr_payload, response_dut.length - PROTOCOL_ID_LEN - MSG_ID_LEN );

}
```

5.2 Modifying XML Files

Python API file, ‘xpro_mcoder.py’ in ‘out’ folder is used to send or receive messages to and from ATmega256RFR2 Xplained Pro board. This file, which is generated automatically from message set XML files (present in ‘xml’ folder) has the definition of messages (commands and responses). ‘xpro_msgdef.xml’ and ‘xpro_msgset.xml’ are the message set files pertaining to ATmega256RFR2 Xplained Pro.

- The ‘xpro_msgset.xml’ should be appended with new message name and its ID (offset)

Figure 5-5. A Typical Example of the HW_TEST Request and Confirm Messages in xpro_msgset.xml

```
<code>
  <name>HWTEST_REQ</name>
  <offset>89</offset>
</code>
<code>
  <name>HWTEST_CONFIRM</name>
  <offset>121</offset>
</code>
<code>
```

Where,

<name> - wraps the name of the request/response message and

<offset> - wraps the unique ID (0-255) assigned for that message.

- The ‘xpro_msgdef.xml’ file should be appended with message definition, which involves providing the parameters of the message

Figure 5-6. Example of HW_TEST Messages Definitions in xpro_msgdef.xml

```
<message>
  <name>HWTEST_REQ</name>
  <description>This is HWTEST.request message structure</description>
  <type>req</type>
  <items>
    <item>
      <name>startup_param</name>
      <type>uint8_t</type>
      <description>Start up parameter to identify the request.</description>
    </item>
  </items>
</message>
<message>
  <name>HWTEST_CONFIRM</name>
  <description>This is HWTEST.confirm message structure</description>
  <type>cnf</type>
  <items>
    <item>
      <name>status</name>
      <type>uint8_t</type>
      <description>
      * Status of the request; This can take the following values,
      * 0x00 - SUCCESS
      * 0x01 - FAILURE
      </description>
    </item>
  </items>
</message>
```

Where,

<message> - the whole message definition.

<name> - the name of the message which is provided in 'xpro_msgset.xml'. This name in small letters is regarded as the API function name.

<description> - brief description of the message.

<type> - defines the type of message (request or response).

'req'- represents request messages.

'cnf' – represents confirm messages.

<items> - group of parameters sent or received in the message.

<item> - definition of single parameter.

<name> - name of the parameter. This name is used as argument for the API function.

<description> - brief description of the parameter.

<type> - type of the parameter. It also defines the number of bytes to be considered as the parameter in the message byte stream. Following are the possible values of the <type> tag.

uint8_t, uint16_t, uint32_t, uint64_t, int8_t, int16_t, int32_t, int64_t, float, octetstr_t.

XML Schema files 'msgdef.xsd' and 'msgset.xsd' present in 'xml' folder provides more information on structure of the message set XML files.

Once the new message is added in the message set files, the API file can be auto generated using the pdt_runner.exe. Invoking pdt_runner.exe with '-g' option transforms the message set XML files of ATmega256RFR2 Xplained Pro and ZigBit USB into API files called 'xpro_mcoder.py' and 'zigbit_usb_mcoder.py'.

pdt_runner.exe –g

5.3 Modifying Extension Class

Extension classes are defined to provide more API functions to control hardware boards. Default extension classes for ATmega256RFR2 Xplained Pro and ZigBit USB are defined in 'xpro_controller.py' and 'zigbit_usb.py' files. The functions created under extension classes can consume the API functions exposed by the auto generated mcoder files thereby acting as wrappers. Figure 5-7 shows the default extension class containing a wrapper function for the HW TEST message discussed in above sections.

Figure 5-7. Default Extension Class

```
class XproControllerExt:

    def test_hw(self):
        self.hwtest_req(DEFAULT_STARTUP_PARAM)
        val = self.hwtest_confirm()
        if val.status != self.SUCCESS:
            raise Exception, ERROR_CODE.get(val.status, val.status)
        return val
```

'XproControllerExt' is the extension class name and 'test_hw' is the function. 'test_hw' calls the 'hwtest_req' and 'hwtest_confirm' functions, which are found in xpro_mcoder.py API file. This provides better abstraction and helps to write test cases very clearly.

5.4 Writing Test Case

A test case here refers to a Python source file placed in the 'tests' folder. A basic understanding of Python is a pre-requisite for writing a test case. The test case must follow a template, for pdt_runner to read it and execute it properly. '*template.py*' available in 'tests' folder acts as the starting point for custom tests cases. Figure 5-8 shows the template test case file.

Figure 5-8. Template Test Case File

```
# Template Test Case file for Wireless Production Test Reference(WPTR).
# NOTE: Please use this file for writing custom test cases.

# Following variables should not be deleted.
testname = "<Test Case Name>"
testdescription = "<One line description of Test>"


# Template test case class; The class name should not be modified.
class Test:

    # This function can be appended for any intialization.
    def __init__(self, info):
        self.info = info
        self.message = ""
        self.testlogger = info['testlogger']
        self.xpro = info['xpro']
        self.golden_dut = info['golden_dut']

    def getMessage(self):
        return self.message

    def setMessage(self, message):
        self.message = message

    def run(self):
        # Test case code goes here

        # If the test case fails the following should be done.
        self.setMessage("FAIL")
        return False

        # If the test case passes the following should be done.
        self.setMessage("PASS")
        return True
```

The template test case file is self explanatory. All the test code goes inside the function ‘run’ found under ‘Test’ class.

Any initializations have to be appended in the ‘__init__’ function. ‘xpro’ and ‘zigbit_usb’ objects created in the ‘__init__’ function provides access to ATmega256RFR2 Xplained Pro and ZigBit USB Stick respectively. These objects have the functions exposed by the API files ‘xpro_mcoder.py’ and ‘zigbit_usb_mcoder.py’ and also by the extension files ‘xpro_controller.py’ and ‘zigbit_usb.py’. The ‘run’ function must return a Boolean saying the status of test (True – Pass; False – Fail). Users can explore existing test cases files to gain more insight.

After a test case file is added, the config.cfg file should be updated to provide the test case name in the ‘testsuite’ options, so that pdt_runner can find it and execute it.

6. Customize Test Logger

A test logger is created for logging test parameters of DUT, like measured current, RSSI value in RF test, crystal calibration value, etc. The test logger here is not used for logging the standard console output.

'testlogger.py' residing in 'src' folder contains classes of test loggers. Based on the configuration of test logger in 'config.cfg', the pdt_runner creates a test logger object and provides it to every test case. Each test case has access to the logger using the object 'self.testlogger'.

Figure 6-5. Typical Test Logger Configuration

```
testlogger:
{
        LoggerFile  :   "./src/testlogger.py"
        Class       :   "FileParameterLogger"
        Config      :
        {
            Name            :   "Template Logger"
            OutputLogFile   :   "parameter_log.csv"
        }

}
```

'**LoggerFile**' refers to the source that contains the implementation of test logger class.

'**Class**' refers to the name of test logger's class found in the 'LoggerFile'.

'**Config**' refers to the argument passed during creation of test logger object. This enables user defined arguments to be passed on to the class during object creation.

By default, the 'FileParameterLogger' is used and it logs the parameters and their values in a CSV file 'parameter_log.csv' provided in 'OuputLogFile' option.

Figure 6-6. Typical Usage of Test Logger Object Inside a Test Case

```
self.testlogger.add_parameter(parameter = "DUT Current",
                              value = result['current'],
                              status = 'fail',
                              info = "Current consumed is not within limits",
                              minlimit = MIN_DUT_CURRENT,
                              maxlimit = MAX_DUT_CURRENT)
```

Users can log this information not only to a file but also to a database, so that the test information of a DUT can be accessed remotely. In order to achieve this, the test logger class should be extended by following these steps:

- Create a new test logger class by inheriting 'ParameterLogger' or 'FileParameterLogger'
- Override 'new_record' function to do any activity that has to be done before the logging starts
- Override 'close_record' function to do any activity that has to be done after the test session is finished. In this function, the logged parameters are available in the 'param' variable and it can be pushed into a server or can be stored wherever required. 'close_record' function implemented in 'FileParameterLogger' class shows, how the parameters are stored in a CSV file.

If pdt_runner is invoked with DUT serial number option (-s), the test logger can log parameters along with serial_number. This helps to identify test output pertaining to any specific DUT.

- Update config.cfg with the new test logger class

Appendix A. Reference Links

- Atmel ATmega256RFR2 Xplained Pro Evaluation Kit
- WPTR Protocol Specification Document
- Atmel ATxmega256A3 and AT86RF233 ZigBit USB Stick
- SAMTEC - TFM-115-02-L-D-LC – HEADER: https://www.samtec.com/ftppub/pdf/TFM_SM.PDF
- SAMTEC - SFM-115-L2-S-D-LC – SOCKET: https://www.samtec.com/ftppub/pdf/SFML_sm.PDF

Appendix B. Revision History

Doc. Rev.	Date	Comments
42253A	03/2014	Initial document release

Enabling Unlimited Possibilities®

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Building
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370

© 2014 Atmel Corporation. All rights reserved. / Rev.: 42253A–WIRELESS–03/2014

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, ZigBit®, and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Windows® is a registered trademark of Microsoft Corporation in U.S. and or other countries. Other terms and product names may be trademarks of others.

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.