IoT-Enablement-Kit 2 Application Guide

Document No.: **L-812e_1**

Kit Prod. No.: **KPW-IOT-002**

phyNODE Product No.: **PBA-D-01**
phyNODE PCB No.: **1426.2**

phyWAVE Product No.: **PWA-A-001**

Edition: December 2015

Copyrighted products are not explicitly indicated in this manual. The absence of the trademark (™, or ®) and copyright (©) symbols does not imply that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this manual.

The information in this document has been carefully checked and is considered to be entirely reliable. However, PHYTEC Messtechnik GmbH assumes no responsibility for any inaccuracies. PHYTEC Messtechnik GmbH neither gives any guarantee nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. PHYTEC Messtechnik GmbH reserves the right to alter the information contained herein without prior notification and accepts no responsibility for any damages that might result.

Additionally, PHYTEC Messtechnik GmbH offers no guarantee nor accepts any liability for damages arising from the improper usage or improper installation of the hardware or software. PHYTEC Messtechnik GmbH further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.

© Copyright 2015 PHYTEC Messtechnik GmbH, D-55129 Mainz.
Rights - including those of translation, reprint, broadcast, photomechanical or similar reproduction and storage or processing in computer systems, in whole or in part - are reserved. No reproduction may occur without the express written consent from PHYTEC Messtechnik GmbH.

	EUROPE	NORTH AMERICA	FRANCE
Address:	PHYTEC Messtechnik GmbH Robert-Koch-Str. 39 D-55129 Mainz GERMANY	PHYTEC America LLC 203 Parfitt Way SW Bainbridge Island, WA 98110 USA	PHYTEC France 17, place Saint-Etienne F-72140 Sillé-le-Guillaume FRANCE
Sales:	+49 6131 9221-32 sales@phytec.de	+1 800 278-9913 sales@phytec.com	+33 2 43 29 22 33 info@phytec.fr
Technical Support:	+49 6131 9221-31 support@phytec.de	+1 206 780-9047 support@phytec.com	support@phytec.fr
Fax:	+49 6131 9221-33	+1 206 780-9135	+33 2 43 29 22 34
Web Site:	http://www.phytec.de http://www.phytec.eu	http://www.phytec.com	http://www.phytec.fr

	INDIA	CHINA
Address:	PHYTEC Embedded Pvt. Ltd. #16/9C, 3rd Main, 3rd Floor, 8th Block, Opp. Police Station Koramangala, Bangalore-560095 INDIA	PHYTEC Information Technology (Shenzhen) Co. Ltd. Suite 2611, Floor 26, Anlian Plaza, 4018 Jin Tian Road Futian District, Shenzhen CHINA 518026
Sales:	+91-80-4086 7046/48 sales@phytec.in	+86-755-3395-5875 sales@phytec.cn
Technical Support:	+91-80-4086 7047 support@phytec.in	support@phytec.cn
Fax:		+86-755-3395-5999
Web Site:	http://www.phytec.in	http://www.phytec.cn

1st Edition December 2015

List of Figures iii
List of Tables iv
Conventions, Abbreviations and Acronyms v
Preface vii
1 Introduction 1
1.1 Hardware Overview........ 1
1.1.1 phyNODE........ 1
1.1.1.1 Features of the phyNODE 1
1.1.1.2 View of the phyNODE with CC2650 phyWAVE Module........ 2
1.1.1.3 Block Diagram of the phyNODE 3
1.1.1.4 Environmental Sensors and I/O Components on the phyNODE........ 4
1.1.2 phyWAVE........ 5
1.1.2.1 Features of the phyWAVE 5
1.1.3 Module Concept........ 6
1.2 Software Overview........ 7
1.2.1 Firmware of the phyWAVE 7
1.2.2 Code Composer Studio (CCS) 7
1.2.3 Architecture of the IoT-Enablement-Kit 2 7
1.2.4 Bluetooth Communication in the Linux OS 8
2 Getting started with the Demonstration Project 9
2.1 Accessing the Sensor Values via gatttool 9
2.1.1 Conversation of GATT Data Fields to true Environmental Sensor Values 12
2.1.1.1 HDC1000 Humidity sensor 12
2.1.1.2 TMP006 IR Temperature Sensor........13
2.1.1.3 MPL3115A2 Pressure Sensor 14
2.1.1.4 MAG3110 Magnetometer Sensor 15
2.1.1.5 TCS3772 Color Sensor 16
2.1.1.6 RGB-LED Control........16
2.2 Access the Sensor Values automatically via *Python* Script 17
2.3 Accessing the Sensor Values manually in Apple iOS 20
2.3.1 Supplementary Information........27
2.3.1.1 UUID explanations........27
2.3.1.2 Helper Functions to Convert the Sensor Data 28
2.3.1.2.1 Calculating the Pressure of MPL3115A2 28
2.3.1.2.2 Calculating the Humidity of HDC1000........ 28
2.3.1.2.3 Calculating the Temperature of TMP006 29
2.3.1.2.4 Calculating the Colors of TCS3772........ 29
2.3.1.2.5 Calculating the Acceleration Axis of MMA8652FC 30
2.3.1.2.6 Calculating the Magnetometer Axis of MAG3110FCR1 ... 30
3 Building and Debugging the Firmware 31
3.1 Setting up a Virtual Linux Machine 31
3.2 Installing the Firmware Development Environment (Windows) 33
3.3 Firmware Build and Debug 36

© PHYTEC Messtechnik GmbH 2015 L-812e_1

4 Introduction to the Application Example 39
4.1 The Structure of the Example Application Firmware 39
4.2 Adding Custom Hardware to the Firmware 40
4.2.1 Adapting the Pin Muxing 40
4.2.2 Adding new Device Drivers 41
4.2.3 Initialization and Environmental Sensor Tasks 43
5 Hardware Description 45
5.1 phyNODE 45
5.1.1 Overview 45
5.1.1.1 Connectors and Pin Header 46
5.1.1.2 Jumpers 47
5.1.1.3 LEDs 48
5.1.1.4 Switches 48
5.1.2 Functional Components on the phyNODE 49
5.1.2.1 Power Supply 49
5.1.2.1.1 Power Connectors (X2)(BAT1,2,3) 49
5.1.2.2 Power LED D5 49
5.1.2.3 Environmental Sensors 50
5.1.2.4 RGB LED (D9) and User Button (S2) 50
5.1.2.5 Arduino Connector (X1) 51
5.1.2.6 JTAG Interface (X4) 52
5.2 phyWAVE CC2650 Module 53
5.2.1 DSC Connector of the phyWAVE 53
5.2.2 Antennas of the phyWAVE 55
6 Troubleshooting 56
6.1 XDS110 Firmware update 56
6.2 Debug Error 56
6.3 Compiler optimization Level 57
6.4 Device does not start without Connection to the XDS110 Debugger 58
7 Revision History 59

© PHYTEC Messtechnik GmbH 2015 L-812e_1

List of Figures

Figure 1: View of the phyNODE with phyWAVE-CC2650 mounted (top view) 2
Figure 2: View of the phyNODE with phyWAVE-CC2650 mounted (bottom view)..................... 3
Figure 3: Block Diagram of the phyNODE with attached phyWAVE-CC2650 SoC...................... 3
Figure 4: Block diagram of the system configuration... 6
Figure 5: Selection of SimpleLink Wireless MCUs during the installation process..................33
Figure 6: Imported projects within Code Composer Studio..34
Figure 7: View of the phyNODE with attached Debugger...37
Figure 8: Overview of the code structure within the phyWAVE_CC2650 project.....................40
Figure 9: Schematic of the RGB LED 41
Figure 10: Adding a new device driver 42
Figure 11: Initialization process at system startup 43
Figure 12: Architecture of Parallel Tasks and their Internal Behavior...................................44
Figure 13: phyNODE Jumpers and Interfaces (top view)..45
Figure 14: phyNODE Jumpers and Interfaces (bottom view) 46
Figure 15: DSC connector of the phyWAVE module..53
Figure 16: Error in case of a wrong setting in CCS 56
Figure 17: Setting up the correct (XDS110) Debugger 57
Figure 18: Optimization Settings in CCS 58

List of Tables

Table 1: Signal Types used in this Manual v
Table 2: Abbreviations and Acronyms used in this Manual vi
Table 3: Environmental Sensor and Actuators on the phyNODE 4
Table 4: GATT Server Sensor Attributes 11
Table 5: Sensor Values read from the Python Script 18
Table 6: UUIDs for Apple iOS Example 27
Table 7: Connectors on the phyNODE 46
Table 8: Jumpers on the phyNODE 47
Table 9: phyNode LEDs Descriptions 48
Table 10: Switches on the phyNODE 48
Table 11: Environmental Sensors on the phyNODE 50
Table 12: Ports used for RGB LED (D9) and User Button (S2) 50
Table 13: Pinout of the Arduino Connector X1 51
Table 14: JTAG Connector X4 52
Table 15: Pin Assignment of the CC2650 phyWAVE Module 54
Table 16: Configuration of the Antenna on the phyWAVE 55

© PHYTEC Messtechnik GmbH 2015

Conventions, Abbreviations and Acronyms

This hardware manual describes the phyWAVE Evaluation Board (PBA-D-01) in the following referred to as phyNODE and the phyWAVE module (PWA-A-001). The manual specifies the design and function of the Phytec modules. Precise specifications for the Texas Instruments CC2650 SimpleLink ™ Multistandard Wireless MCU can be found in the corresponding datasheet and technical reference manual.

Conventions

The conventions used in this manual are as follows:

- Signals that are preceded by an "n", "/", or "#"character (e.g.: nRD, /RD, or #RD), or that have a dash on top of the signal name (e.g.: $\overline{RD}$) are designated as active low signals. That is, their active state is when they are driven low, or are driving low.
- A "0" indicates a logic zero or low-level signal, while a "1" represents a logic one or high-level signal.
- The hex-numbers given for addresses of I^2C devices always represent the 7 MSB of the address byte. The correct value of the LSB which depends on the desired command (read (1), or write (0)) must be added to get the complete address byte. E.g. given address in this manual 0x41 => complete address byte = 0x83 to read from the device and 0x82 to write to the device.
- Tables which describe jumper settings show the default position in **bold, blue text.**
- Text in *blue italic* indicates a hyperlink within, or external to the document. Click these links to quickly jump to the applicable URL, part, chapter, table, or figure.
- Text in ***bold italic*** indicates an interaction by the user, which is defined on the screen.
- Text in `Consolas` indicates an input by the user, without a premade text or button to click on, or a piece of software.
- Text in *italic* indicates proper names of development tools and corresponding controls (windows, tabs, commands, names of variables etc.) used within the development tool, no interaction takes place.
- White Text on black background shows the result of any user interaction (command, program execution, etc.)

Types of Signals

Different types of signals (ST) are brought out at the connectors on the phyNODE. The following table lists the abbreviations used to specify the type of a signal.

Signal Type (ST)	Description	Abbr.
Power	Supply voltage input	PWR_I
Power	Supply voltage output	PWR_O
Ref-Voltage	Reference voltage output	REF_O
Input	Digital input	I
Output	Digital output	O
IO	Bidirectional input/output	I/O

Table 1: Signal Types used in this Manual

Abbreviations and Acronyms

Many acronyms and abbreviations are used throughout this manual. Use the table below to navigate unfamiliar terms used in this document.

Abbreviation	Definition
DSC	Direct Solder Connect
GPIO	General purpose input and output.
J	Solder jumper; these types of jumpers require solder equipment to remove and place.
JP	Solderless jumper; these types of jumpers can be removed and placed by hand with no special tools.
NC	Not Connected
PCB	Printed circuit board.
RTC	Real-time clock.
SBC	Used in reference to Phytec's Single Board Computers
SOC	System on Chip; used in reference to the phyWAVE module (PWA-A-001)
SoM	Used in reference to Phytec's System on Modules
SDR	Software Defined Radio

Table 2: Abbreviations and Acronyms used in this Manual

	At this icon you might leave the path of this Application Guide.
	This is a warning. It helps you to avoid annoying problems.
	You can find useful supplementary information about the topic.
	You have successfully completed an important part of this Application Guide.
	You can find information to solve problems.

© PHYTEC Messtechnik GmbH 2015 L-812e_1

Preface

As a member of PHYTEC's new phyWAVE® product family the IoT-Enablement-Kit 2 is one of a series of PHYTEC kits that offer off-the-shelf solutions to explore the world of IoT and to get started with the development of visionary new IoT products.

The Internet of Things (IoT) is part of Industry 4.0, the fourth industrial revolution after the invention of the steam engine, the production line and the Internet. The vision is to setup a network of interconnected objects that not only harvests information from the environment (sensing) and interacts with the physical world (actuation/command/ control), but also uses existing internet standards to provide services for information transfer, analytics, applications, and communications. The goal is to create opportunities for more direct integration between the physical world and computer-based systems resulting in improved efficiency, accuracy and economic benefit.

The challenges of IoT arise from the need to achieve cheap and stable connections between different device classes such as cheap, small, low energy nodes and powerful Linux-based single board computers.

The new phyWAVE® product family provides one of the industry's first platforms into the newest technologies in the evolving world of IoT. The family consists of cheap modules for wireless communication – so called phyWAVE®s - supporting different protocols and levels of computing power, development boards with different sensors and actuators – so called phyNODE®s, and gateways to the internet – so called phyGATE®s – which are based on our well-proven Single Board Computers.

Our Efforts are Your Benefit

Phytec's phyWAVE® product family deploys industry-leading standards and software solutions to ensure interoperability of our IoT hardware offerings.

We closely watch and track market acceptance of various solutions, including manufacturer support and technical specification. This approach allows us to quickly respond to the changes and ensure optimal use of our products in the rapidly evolving and dynamic IoT realm.

Phytec's development efforts extend to the very beginning of the IoT era. We have observed market trends and devised solutions in accordance to evolving standards and communication formats. The Internet of Things is not unknown territory for Phytec, but rather the natural outcome of over 25 years of engaging in development and providing solutions in the embedded processor, networking and M2M space.

Cost-optimized with Direct Solder Connect (DSC) Technology

At the heart of the phyWAVE® product family are the phyWAVE® SoCs (System on Chip). All phyWAVE® SoCs allow for direct soldering onto a carrier board PCB for routing of signals from the SoC to applicable sensors. This “Direct Solder Connect” (DSC) of the SoC eliminates costly PCB to PCB connectors, thereby further reducing overall system costs, and making the phyWAVE® modules ideally suited for deployment into a wide range of cost-optimized and robust industrial applications.

OEM Implementation

Implementation of an OEM-able SoC subassembly as the "core" of your embedded design allows you to focus on hardware peripherals and firmware without expending resources to "re-invent" microcontroller and wireless communication circuitry.

Furthermore, Phytec IoT solutions offer customers the benefit of an included radio license, compatibility as well as extensive electrical, functional and operating test.

Also, Phytec IoT products assign the key-requirement to integrate things to a complex environment (a cheap, easy and reliable connection to the internet) to concrete technical capabilities:

1. easy: a pre-configured IP-based communication stack for each sensor-node. Unified software interfaces for sensors and chip-peripherals like timer.
2. reliable: IP-based communication, Acknowledged transmissions, advanced antenna design, deeply tested hardware components, long-term availability.
3. cheap : a module price below 10€, No software licensing cost.
4. openness: consistent use of open-source software.

As a consequence , use of Phytec's IoT products allows to implement this strong future-oriented architecture into any custom design.

Software Support

Production-ready firmware and Design Services for our hardware will further reduce your development time and risk and allow you to focus on your product expertise.

Use of open-source software on industry established Phytec Linux-based Single Board Computers (SBC) in harsh environments has been demonstrated successfully. All software components being used within phyWAVE® product family is available under open-source licenses and free.

© PHYTEC Messtechnik GmbH 2015 L-812e_1

Ordering Information

Ordering numbers:
phyWAVE CC2650 Wireless Module: **PWA-A-001**
phyWAVE Evaluation Board (phyNODE): **PBA-D-01**

Product Specific Information and Technical Support

In order to receive product specific information on changes and updates in the best way also in the future, we recommend to register at

http://www.phytec.de/de/support/registrierung.html or
http://www.phytec.eu/europe/support/registration.html

For technical support and additional information concerning your product, please visit the support section of our web site which provides product specific information, such as errata sheets, application notes, FAQs, etc.

http://www.phytec.de/de/support/faq/faq-iot-enablement-kit-2.html or
http://www.phytec.eu/europe/support/faq/faq-iot-enablement-kit-2.html

Other Products and Development Support

Aside of the new product family, PHYTEC supports a variety of 8-/16- and 32-bit controllers in two ways:

(1) as the basis for Rapid Development Kits which serve as a reference and evaluation platform

(2) as insert-ready, fully functional OEM modules, which can be embedded directly into the user's peripheral hardware design.

Take advantage of PHYTEC products to shorten time-to-market, reduce development costs, and avoid substantial design issues and risks. With this new innovative full system solution you will be able to bring your new ideas to market in the most timely and cost-efficient manner.

For more information go to:

http://www.phytec.de/de/leistungen/entwicklungsunterstuetzung.html or
www.phytec.eu/europe/oem-integration/evaluation-start-up.html

Declaration of Electro Magnetic Conformity of the PHYTEC phyNODE(PBA-D-01) and phyWAVE (PWA-A-001)

PHYTEC Single Board Computers (henceforth products) are designed for installation in electrical appliances, or as part of custom applications, or as dedicated Evaluation Boards (i.e.: for use as a test and prototype platform for hardware/software development) in laboratory environments.

Caution!
PHYTEC products lacking protective enclosures are subject to damage by ESD and, hence, may only be unpacked, handled or operated in environments in which sufficient precautionary measures have been taken in respect to ESD-dangers. It is also necessary that only appropriately trained personnel (such as electricians, technicians and engineers) handle and/or operate these products. Moreover, PHYTEC products should not be operated without protection circuitry if connections to the product's pin header rows are longer than 3 m.

PHYTEC products fulfill the norms of the European Union’s Directive for Electro Magnetic Conformity only in accordance to the descriptions and rules of usage indicated in this hardware manual (particularly in respect to the pin header row connectors, power connector and serial interface to a host-PC).

Implementation of PHYTEC products into target devices, as well as user modifications and extensions of PHYTEC products, is subject to renewed establishment of conformity to, and certification of, Electro Magnetic Directives. Users should ensure conformance following any modifications to the products as well as implementation of the products into target systems.

© PHYTEC Messtechnik GmbH 2015 L-812e_1

Product Change Management and information in this manual on parts populated on the SoM / SBC

When buying a PHYTEC SoM / SBC, you will, in addition to our HW and SW offerings, receive a free obsolescence maintenance service for the HW we provide.

Our PCM (Product Change Management) Team of developers, is continuously processing, all incoming PCN's (Product Change Notifications) from vendors and distributors concerning parts which are being used in our products.

Possible impacts to the functionality of our products, due to changes of functionality or obsolesce of a certain part, are being evaluated in order to take the right masseurs in purchasing or within our HW/SW design.

Our general philosophy here is: **We never discontinue a product as long as there is demand for it.**

Therefore we have established a set of methods to fulfill our philosophy:

Avoiding strategies

- Avoid changes by evaluating long-livety of parts during design in phase.
- Ensure availability of equivalent second source parts.
- Stay in close contact with part vendors to be aware of roadmap strategies.

Change management in rare event of an obsolete and non replaceable part

- Ensure long term availability by stocking parts through last time buy management according to product forecasts.
- Offer long term frame contract to customers.

Change management in case of functional changes

- Avoid impacts on product functionality by choosing equivalent replacement parts.
- Avoid impacts on product functionality by compensating changes through HW redesign or backward compatible SW maintenance.
- Provide early change notifications concerning functional relevant changes of our products.

Therefore we refrain from providing detailed part specific information within this manual, which can be subject to continuous changes, due to part maintenance for our products.

In order to receive reliable, up to date and detailed information concerning parts used for our product, please contact our support team through the contact information given within this manual.

© PHYTEC Messtechnik GmbH 2015 L-812e_1

1 Introduction

Phytec's IoT Enablement Kit 2 provides one of the industry's first platforms into the newest technologies in the evolving world of IoT (Internet of Things). The kit enables wireless data acquisition via Bluetooth Low Energie (*BLE*) as well as IoT network and configuration with open source software tools.

1.1 Hardware Overview

The hardware consists of a phyWAVE Evaluation Board, in the following called phyNODE (yellow PCB), and a phyWAVE wireless communication module mounted on the phyNODE.

1.1.1 phyNODE

The phyNODE (*Figure 1*) is equipped with different sensors to perceive environmental data such as temperature and humidity, as well as with user I/Os like LEDs and buttons. It serves as platform for different wireless communication modules which can be mounted via DSC (Direct Solder Connection). Hence, the phyNODE allows to test the communication with it's sensors with different protocols depending on the phyWAVE mounted. *Figure 3* gives a more detailed overview of the internal structure of the phyNODE.

1.1.1.1 Features of the phyNODE

The phyNODE supports the following features :

- DSC (Direct Solder Connect) connector
- Two different power supply options (5 V via micro USB connector; or battery powered with one 2032 coin cell)
- An Arduino-Uno compatible extension connector allowing to add additional functionality
- Reset and User Button
- RGB LED
- Environmental Sensors
  - Temperature
  - Humidity
  - IR-Temperature
  - Pressure
  - Light intensity and color
  - 3-Axis Accelerometer
  - 3-Axis Magnetometer

1.1.1.2 View of the phyNODE with CC2650 phyWAVE Module

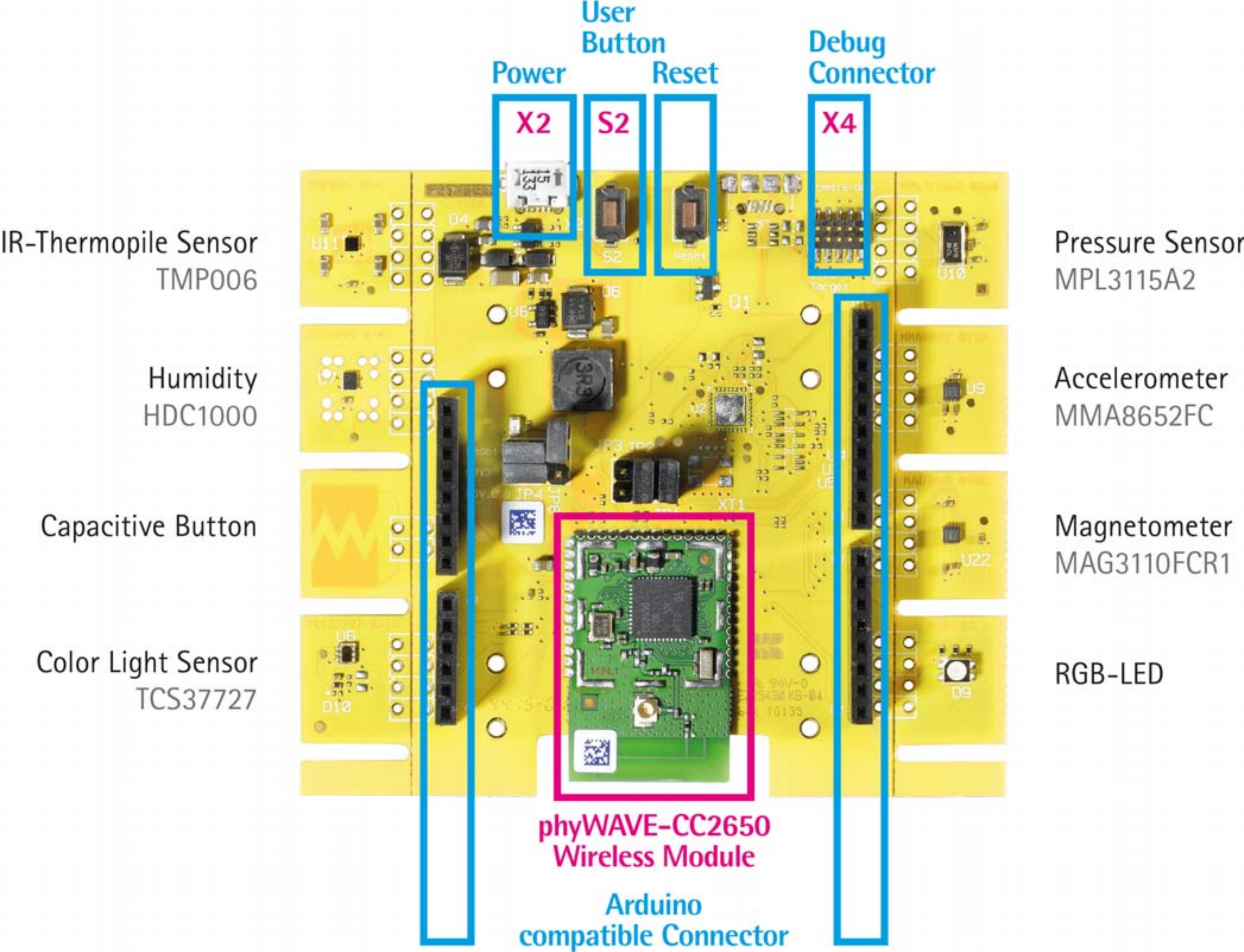

Figure 1: View of the phyNODE with phyWAVE-CC2650 mounted (top view)

© PHYTEC Messtechnik GmbH 2015 L-812e_1

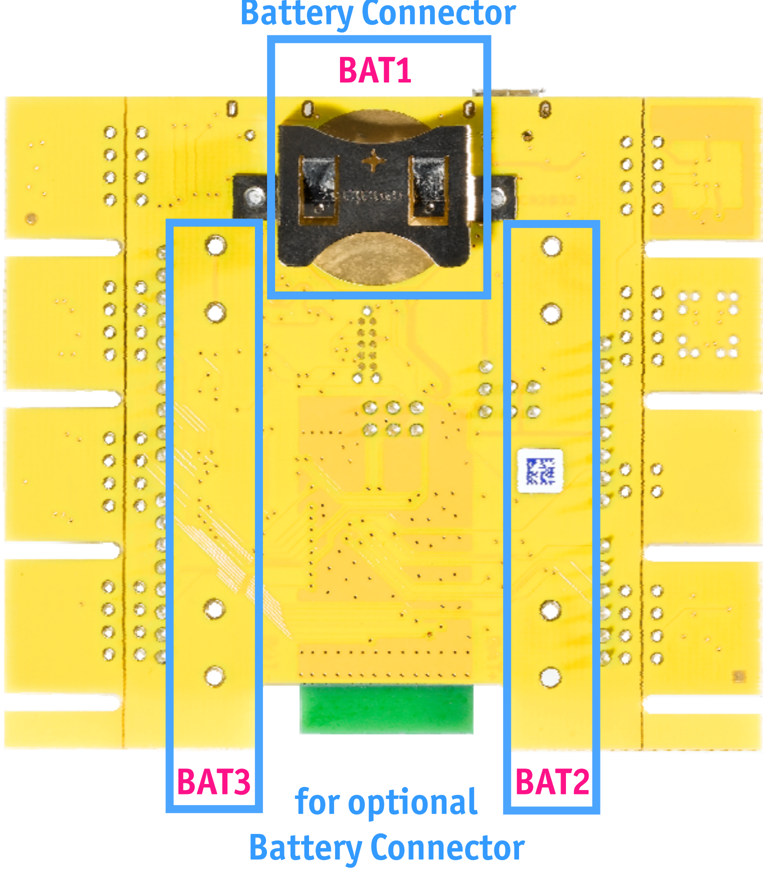

Figure 2: View of the phyNODE with phyWAVE-CC2650 mounted (bottom view)

1.1.1.3 Block Diagram of the phyNODE

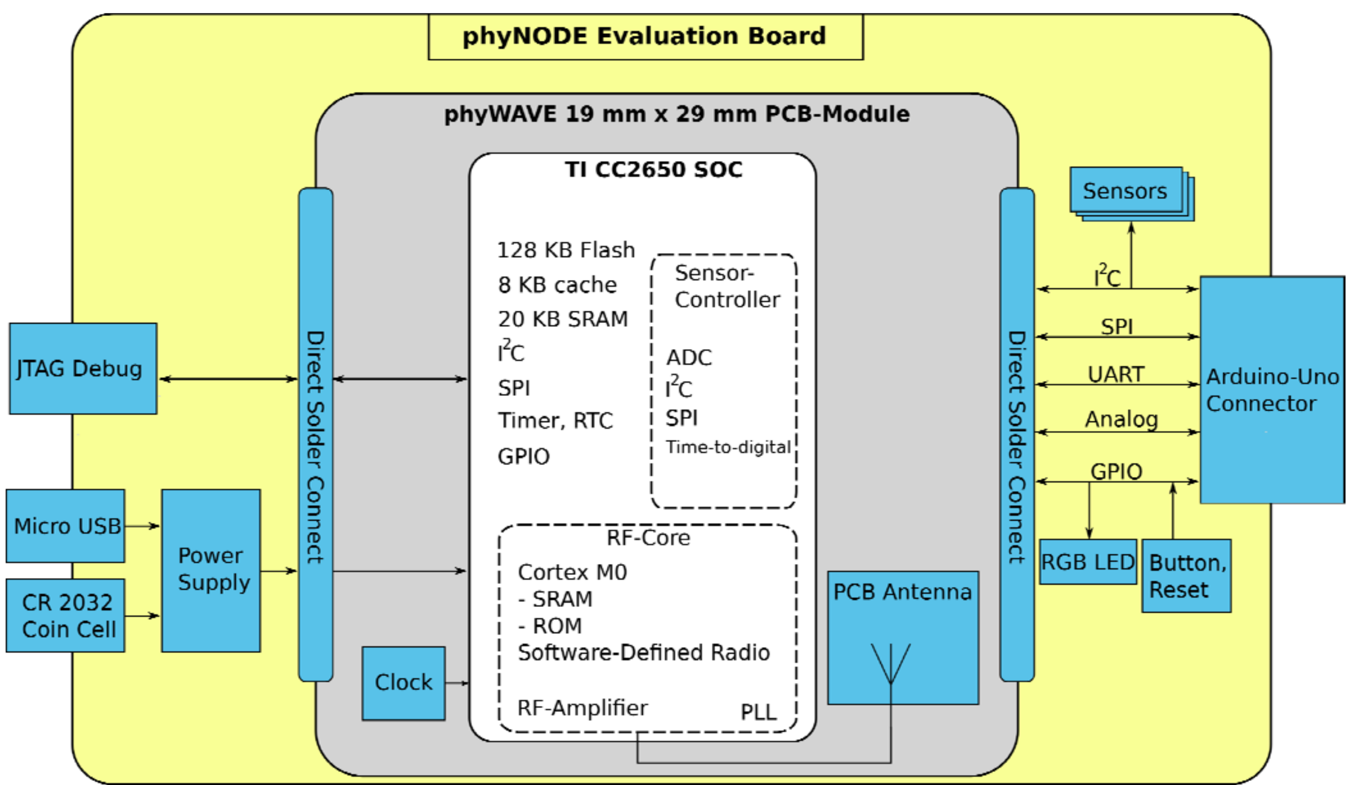

Figure 3: Block Diagram of the phyNODE with attached phyWAVE-CC2650 SoC

1.1.1.4 Environmental Sensors and I/O Components on the phyNODE

The phyNODE is equipped with various environmental sensors and user I/Os listed in the following table:

Sensor Type	Detailed Information
Color Sensor	*TCS3772*
Digital Pressure Sensor	*MPL3115A2*
Three-Axis Accelerometer	*MMA8652FC*
Three-Axis Magnetometer	*MAG3110FCR1*
IR-Thermopile Temperature Sensor	*TMP006*
Digital Humidity Sensor	*HDC1000*
RGB-LED	
PCB capacitive button	

Table 3: Environmental Sensor and Actuators on the phyNODE

All environmental sensors installed on the phyNODE board are connected via I^2C.

Please refer to *section 5.1* for a more detailed description of the phyNODE board.

© PHYTEC Messtechnik GmbH 2015 L-812e_1

1.1.2 phyWAVE

The phyWAVE Wireless Communication Module is a 19 mm x 29 mm PCB which allows to fetch and process sensor values and to serve as wireless link for the sensors of the phyNODE, or a custom board. It provides a DSC (Direct Solder Connector) with 40 connections to mount it to a sensor board. Depending on the controller installed different transmission protocols are possible. The phyWAVE module used in the IoT-Enablement-Kit 2 is equipped with a Texas Instruments (TI) CC2650 SimpleLink ™ Multistandard Wireless MCU, precise clock sources, an integrated PCB antenna, and an U.FL RF connector to attach an external antenna. The CC2650 integrates a Cortex-M3 microprocessor, Flash, RAM, hardware implemented interfaces such as I^2C or SPI and an RF-backend for wireless communication. It can be programmed to operate with either Bluetooth Low Energy (BLE) or IEEE 802.15.4 wireless communication standards within the 2.5 GHz ISM band. This manual focuses on the BLE operation of the SoC. The grey area of *Figure 3* gives a more detailed overview of the structure of the phyWAVE module.

1.1.2.1 Features of the phyWAVE

The phyWAVE comes with a populated TI CC2650 MCU with the following key features. For more information, refer to the *datasheet* or *technical reference manual* of the TI CC2650 SoC.

- ARM Cortex M3 application processor providing
  - 128 KB flash, 115 KB ROM, 20 KB SRAM, 8 KB Cache
  - 48 MHz Clock Speed, 32 kHz for RTC
  - Integrated RF-Section
    - 2.5 GHz ISM band
    - Up to 5 dBm output power
- Software Defined Radio (SDR) allows to run different wireless communication standards
  - Bluetooth Low Energy / Bluetooth Smart
  - IEEE 802.15.4
- Standard Interfaces, i.e. I^2C, I^2S , SPI, UART, ADC
- RTC, Timer
- PCB antenna
- U.FL RF connector to attach an external antenna

	Please refer to *section 5.2* for a more detailed description of the phyNODE board.

1.1.3 Module Concept

The module concept allows evaluating the system's capabilities by using the out of the box features of the evaluation board, i.e. integrated environmental sensors, power management and debugging connection, as a first step. *Figure 4* shows a block diagram of the typical environment of the IoT-Enablement-Kit 2. We expect that most customer applications will use the phyWAVE wireless communication module in combination with a custom hardware platform that features individual sensors or user interaction (e.g. push button, display...). This concept simplifies product design massively, since the time consuming and costly design of the PCB RF antenna do not have to be done each time. The PCB RF antenna on the phyWAVE has already passed simulation, design and validation.

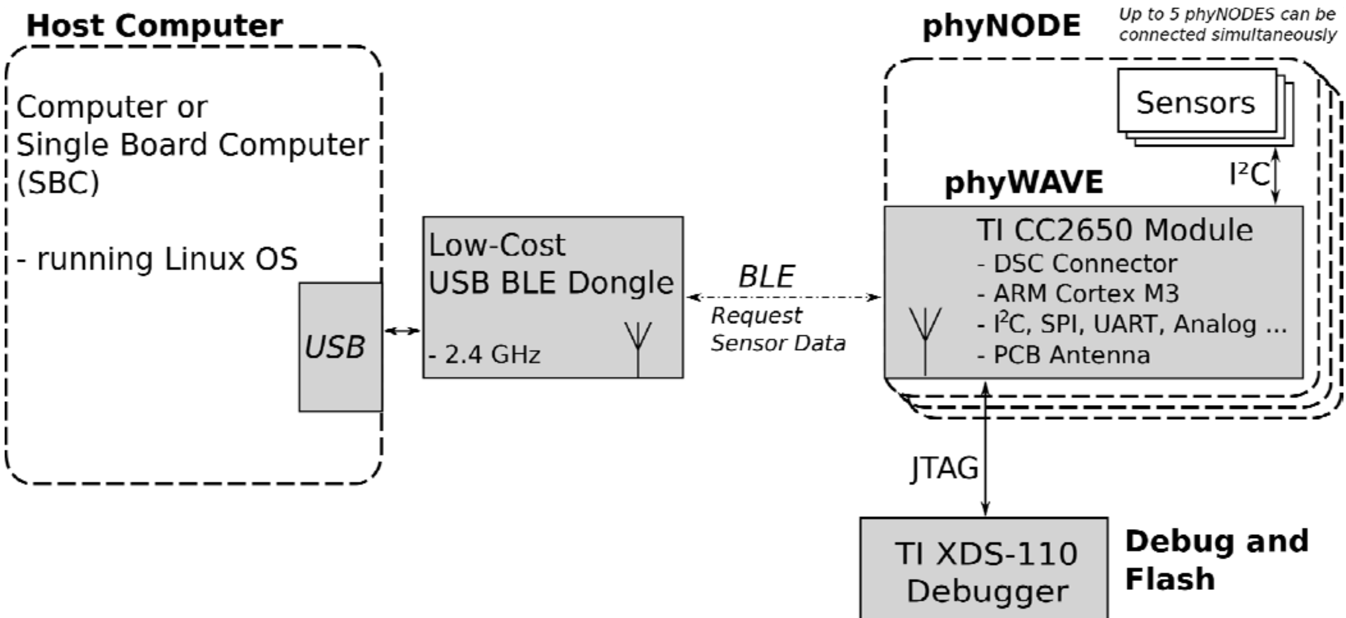

Figure 4: Block diagram of the system configuration

© PHYTEC Messtechnik GmbH 2015 L-812e_1

1.2 Software Overview

The software delivered with the IoT-Enablement-Kit 2 includes:

- Demonstration firmware for the CC2650 MCU including the Bluetooth Low Energy compatible *GATT* server
- Software examples and demonstration application

1.2.1 Firmware of the phyWAVE

With their *Bluetooth stack,* TI provides an extensive collection of firmware examples for the CC2650 SoC. The firmware of phyWAVE module is based on the TI Bluetooth stack 2.1. The phyWAVE ships with a pre-flashed CC2650 SoC. The default application implements a *GATT* server that provides an interface to fetch the sensor values on the phyNODE. Furthermore, all environmental sensors are provided with basic driver support within the example application.

The actual Bluetooth stack on the CC2650 SoC is available as a prebuilt library. The application, driver and *GATT* server setup, however, are available in source code in a BSD-like license. Flashing and debugging of the phyWAVE firmware is supported with the enclosed TI *DevPack XDS-110 debugger* in combination with CCS.

1.2.2 Code Composer Studio (CCS)

Code Composer Studio (CCS) is an IDE provided by TI, based on the Eclipse platform. If the CCS IDE is used in combination with a TI debugger, it can be used full-featured without charge for private and commercial applications.

The firmware provided as example project that ships with the IoT-Enablement-Kit 2 uses the CCS IDE for programming, flashing and debugging on the phyNODE.

	This manual describes the installation and setup of CCS in Windows. We recommend installing a Virtual Machine on your host computer to have instances of both operating systems, *Linux* and *Windows*. The setup of both environments will be explained later in *chapter 3*.

1.2.3 Architecture of the IoT-Enablement-Kit 2

The IoT-Enablement-Kit 2 is being used in a star network configuration with a host computer as central device. A standard *Linux* host computer with an attached BLE USB-dongle serves as access point. Alternatively, other *Linux* supported Bluetooth 4.0/4.1 hardware e.g. a Notebook or a Single Board Computer (SBC) with an integrated Bluetooth module can be used as access point.

1.2.4 Bluetooth Communication in the Linux OS

The Linux OS implements a Bluetooth Stack that features the established regular Bluetooth communication (Bluetooth 2.x-3.x) but also the Bluetooth Low Energy extension (Bluetooth 4.x). The Linux Bluetooth stack is called *BlueZ.* Beside the *BlueZ* stack there are tools like the *hcitool* to analyze and maintain connection to Bluetooth devices. The use of these tools in combination with the phyNODE board is explained later in *section 2*.

© PHYTEC Messtechnik GmbH 2015 L-812e_1

2 Getting started with the Demonstration Project

During this chapter you will learn how to access the sensor values via BLE.

We assume that you have first completed our IoT-Enablement-Kit 2 QuickStart Guide successfully.

There are several methods to request the sensor values:

- via *Python* script (as explained in the QuickStart Guide)
- via *Linux gatttool*
- with an *Apple iOS* device and the provided minimal iOS-App

2.1 Accessing the Sensor Values via gatttool

This section describes how to fetch sensor data with *gatttool*. *Gatttool* is a *Linux* tool that can be used to manipulate attributes on a *GATT* server via *BLE* interface. Before a data set of a specific sensor can be fetched, the sensor has to be activated by writing 1 to the corresponding configuration handle. The configuration handle is represented as one byte on the *GATT* server (the phyWAVE module). This configuration byte is written via Bluetooth using *gatttool*. The actual functionality of the phyWAVE is implemented by means of attributes which can be read or written to, depending on the attribute. The phyWAVE *GATT* attributes are described in *Table 4*.

	The IoT-Enablement-Kit 2 comes with a *Python* script that generally uses the same mechanism to gather all sensor data. Using the *Python* script should be preferred, since it is easier to use (see QuickStart Guide and *section 2.1.1*). However, this manual method shows the principle of the BLE data communication.

The following procedure exemplary describes how to access temperature and humidity values:

- Plug in the USB Bluetooth dongle and check if a *hci* interface is available to the system:
  `:~$ hcitool dev`

```
> Devices:
     hci0   00:1B:DC:xx:xx:xx
```

- Find out the device address of your phyNODE board:
  `:~$ sudo hcitool lescan`

```
> 68:C9:0B:05:59:8C CC2650 PhyWAVE
```

- Start *gatttool* to interface with the desired board:
  `:~$ gatttool -b 68:C9:0B:05:59:8C -I`

```
> [  ][68:C9:0B:05:59:8C][LE]>
```

- Type `connect` which will result in:

```
> [CON][68:C9:0B:05:59:8C][LE]>
```

- To activate e.g. the IR-temperature sensor (HDC1000 Humidity Conf) write 1 to the corresponding configuration handle by typing:

  `:~$ [68:C9:0B:05:59:8C][LE]> char-write-cmd 0x2c 01`

  The configuration and data fields of all sensors are described in *Table 4*.

- Read the raw value of the humidity sensor (HDC1000 Humidity Data):
  `:~$ [68:C9:0B:05:59:8C][LE]> char-read-hnd 0x29`

```
> [68:C9:0B:05:59:8C][LE]> fc 5e 00 7e
```

- These values represent the on-chip temperature and relative humidity of the HDC1000 sensor with a 16 bit resolution. To get the real values for temperature and humidity, the data has to be converted. The conversation algorithms are explained in *section 2.1.1*.

© PHYTEC Messtechnik GmbH 2015 L-812e_1

Name	Short UUID	Handle	Read/Write	Format
IR-Thermopile Temperature Sensor				
TMP006 IR-Temp Service	0xAA00	-	-	-
TMP006 IR-Temp Data	0xAA01	0x0021	Read	4 Byte
TMP006 IR-Temp Conf	0xAA02	0x0024	Write	1 Byte
TMP006 IR-Temp Period	0xAA03	0x0026	Write	1 Byte (ms)
Digital Pressure Sensor				
MPL3115A2 Barometer Service	0xAA40	-	-	-
MPL3115A2 Barometer Data	0xAA41	0x0031	Read	6 Byte
MPL3115A2 Barometer Conf	0xAA42	0x0034	Write	1 Byte
MPL3115A2 Barometer Period	0xAA44	0x0036	Write	1 Byte (ms)
Three-Axis Accelerometer				
MMA8652FC Accelerometer Service	0xAA10	-	-	-
MMA8652FC Accelerometer Data	0xAA11	0x0039	Read	6 Byte
MMA8652FC Accelerometer Conf	0xAA12	0x003c	Write	1 Byte
MMA8652FC Accelerometer Period	0xAA13	0x003e	Write	1 Byte (ms)
Color Sensor				
TCS3772 Color Service	0xAA90	-	-	-
TCS3772 Color Data	0xAA91	0x0049	Read	8 Byte
TCS3772 Color Conf	0xAA92	0x004c	Write	1 Byte
TCS3772 Color Period	0xAA94	0x004e	Write	1 Byte (ms)
Digital Humidity Sensor				
HDC1000 Humidity Service	0xAA20	-	-	-
HDC1000 Humidity Data	0xAA21	0x0029	Read	4 Byte
HDC1000 Humidity Conf	0xAA22	0x002c	Write	1 Byte
HDC1000 Humidity Period	0xAA23	0x002e	Write	1 Byte (ms)
Three-Axis Magnetometer				
MAG3110FCR1 Magnetometer Service	0xAA30	-	-	-
MAG3110FCR1 Magnetometer Data	0xAA31	0x0041	Read	6 Byte
MAG3110FCR1 Magnetometer Conf	0xAA32	0x0044	Write	1 Byte
MAG3110FCR1 Magnetometer Period	0xAA33	0x0046	Write	1 Byte (ms)
Push Button				
Key press Service	0xFFE0	-	-	-
Key press Data	0xFFE1	0x0051	Read	1 Byte
RGB-LED				
RGB-LED Service	0xAA64	-	-	-
RGB-LED Conf	0xAA66	0x0058	Write	1 Byte

Table 4: GATT Server Sensor Attributes

2.1.1 Conversation of GATT Data Fields to true Environmental Sensor Values

This chapter points out how to convert the *GATT* data fields' format to true environmental sensor values. This is necessary if you access the data manually in *Linux* via *hcitool*. When using the prepared *Python* script or the *iOS* example this procedure is not necessary as it prints the true sensor data directly to the console. Some sensors, e.g. the TMP006 IR temperature sensor, require some calculation to convert the raw values to true sensor values. The following examples explain these calculations only in brief. Please refer to the corresponding data sheets for detailed information on the necessary conversation calculations.

Alternatively, you can have a look at the conversation done in the *Python* script within the *git* repository discussed in *section 2.1.1*. The *Python* script is located in the repository at:
[..]\ble-cc26xx\ble_host_sw\bluepy\bluepy\phyWaveBLE.py

2.1.1.1 HDC1000 Humidity sensor

- To get the temperature and the humidity execute the following commands:

```
:~$ [68:C9:0B:05:59:8C][LE]> char-write-cmd 0x2c 01
:~$ [68:C9:0B:05:59:8C][LE]> char-read-hnd 0x29
```

The output will be something similar to:

```
> Characteristic value/descriptor: 5e fc 7e 00
```

The return value represents the content of the two HDC1000 16-bit registers 0x00 (raw temperature) and 0x01 (raw humidity) according to the following structure:

temp_MSB	temp_LSB	hum_MSB	hum_LSB

Hence, in this example we receive:
raw_tmp (register 0x00) = 0x5efc
raw_hum (register 0x01) = 0x7e00;

- Use the following formula to calculate the true temperature and humidity:
  temp = -40 + 165 * (raw_tmp / 65536)
  hum = 100* (raw_hum/65536)

  Refer to the *HDC1000 datasheet* (p. 15) for more information.

Your calculated values should match the following values for temperature and humidity:

Temperature = 21.2°C
Humidity = 49.2%

© PHYTEC Messtechnik GmbH 2015

2.1.1.2 TMP006 IR Temperature Sensor

- To get the temperature of an object, first execute the following commands:
```
:~$ [68:C9:0B:05:59:8C][LE]> char-write-cmd 0x24 01
:~$ [68:C9:0B:05:59:8C][LE]> char-read-hnd 0x21
```

The output will be something similar to:

```
> Characteristic value/descriptor: 85 fe e8 0c
```

The return value represents the content of the two TMP006 16-bit registers 0x00 (sensor voltage (V_{obj})) and 0x01 (die temperature (t_{Amb}) in °C) according to the following structure:

Vobj_LSB	Vobj_MSB	tAmb_raw_LSB	tAmb_raw_MSB

Both raw data values are reported in 16 bit binary twos complement signed integer format. In this example we receive:

V_{obj} (register 0x00) = 0xFE85 which converts to -379

t_{Amb_raw} (register 0x01) = 0x0CE8 which converts to 3304

The names of the variables correspond to the names used in the software delivered with the IoT-Enablement-Kit 2. To comprehend the equations used in the documentation from TI use T_{DIE} instead of t_{Amb_K}, and $f(V_{obj})$ instead of f_{Obj}.

Use the following formula to calculate the true ambient and object temperature:

There are some constants necessary for the calculation:

Constant	Value	Constant	Value
S0 - Calibration Factor	6.4×10^{-14}	b0	-2.94×10^{-5}
a1	1.75×10^{-3}	b1	-5.7×10^{-7}
a2	-1.678×10^{-5}	b2	4.63×10^{-9}
T_{REF}	298.15 K	c2	13.4

In the first two equations the integer temperature result stored in the tAmb_raw register is converted into a physical temperature value in Kelvin.

$t_{Amb} = t_{Amb_raw}/128$

$t_{Amb_K} = t_{Amb} + 273.15$

Now t_{diff}, which is required for the later formulas, is calculated.

$t_{diff} = t_{Amb_K} - T_{REF}$

S represents the sensitivity of the thermopile sensor and how it changes over temperature:
$S = S0 * (1 + a1 * t_{diff} + a2 * pow(t_{diff}, 2))$

V_{os} refers to the offset voltage that arises because of the slight self-heating of the TMP006, caused by the non-zero thermal resistance of the package and the small operational power dissipation of 1 mW in the device:
$V_{os} = b0 + b1 * t_{diff} + b2 * pow(t_{diff}, 2)$

V_{obj} and V_{os} are now used in the following equation which models the Seebeck coefficients of the thermopile and how these coefficients change over temperature.
$f_{Obj} = (V_{obj} - V_{os}) + c2 * pow((V_{obj} - V_{os}), 2)$

Finally t_{Obj} representing the object temperature can be calculated and reconverted into °C.
$t_{Obj} = pow(pow(t_{Amb_K}, 4) + (f_{Obj} / S), 0.25)$
$t_{Obj} = t_{Obj} - 273.15$

Refer to the *TMP006 Users Guide* (p.10) for a mathematical form of the equations above or the *TMP006 datasheet* (p. 12, 15) for more information.

Your calculated values should match the following values for ambient and object:

Temperature_ambient = 25.8°C
Temperature_object = 21.4°C

2.1.1.3 MPL3115A2 Pressure Sensor

- Execute the following commands to read the raw temperature and pressure values:

```
:~$ [68:C9:0B:05:59:8C][LE]> char-write-cmd 0x34 01
:~$ [68:C9:0B:05:59:8C][LE]> char-read-hnd 0x31
```

The output will be something similar to:

```
> Characteristic value/descriptor: 61 00 c0 b9 10 15
```

The return value represents the content of five 16-bit registers 0x01 to 0x03 (raw pressure) and 0x04 to 0x05 (raw temperature) according to the following structure:

press_MSB	status	press_CSB	press_LSB	temp_LSB	temp_MSB

In this example we receive:
raw_press (registers 0x01 to 0x03) = 0x61B9C0
raw_temp (registers 0x04 to 0x05) = 0x1510;

The raw pressure data is reported in 20 bit value in Pascal in Q18.2 format. The raw temperature data is reported in degrees C in 12 bit Q12.4 format.

- Use the following formula to calculate the true for pressure and temperature:

© PHYTEC Messtechnik GmbH 2015 L-812e_1

press = raw_press / 64
temp = raw_temp/256

Refer to the *MPL3115A2 datasheet* (p. 19, 21) for more information.

Your calculated values should match the following values for pressure and temperature:

Pressure = 100071 Pa
Temperature = 21.1 °C

2.1.1.4 MAG3110 Magnetometer Sensor

- Read the magnetometer data by executing the following commands:

```
:~$ [68:C9:0B:05:59:8C][LE]> char-write-cmd 0x44 01
:~$ [68:C9:0B:05:59:8C][LE]> char-read-hnd 0x41
```

The output will be something similar to:

```
> Characteristic value/descriptor: d5 ff 78 ff 95 fb
```

The return value represents the raw values of the magnetic field strength for the 3 axis in six 16-bit registers 0x00 to 0x06 according to the following structure:

magX_LSB	magX_MSB	magY_LSB	magY_MSB	magZ_LSB	magZ_MSB

The raw data values are reported in 16 bit binary twos complement signed integer format. Thus, with the example values we receive:
raw_magX (registers 0x01 to 0x02) =0xff5d which converts to -163
raw_magY (registers 0x03 to 0x04) =0xff78 which converts to -136
raw_magZ (registers 0x05 to 0x06) =0xfb95 which converts to -1131

- The full-scale range of the MAG3110 sensor is -1000 µT to 1000 µT. Use the following formula to calculate the true values for the 3 axis:
  mag = raw_mag / (65536 / 2000)

  Refer to the *MAG3110 datasheet* (p. 15) for more information.

Your calculated values should match the following values for the magnetic field strength:

magX = -4.97 µT
magY = -4.15 µT
magZ = -34.51 µT

2.1.1.5 TCS3772 Color Sensor

- Read the TCS3772 color sensor data by executing the following commands:

```
:~$ [68:C9:0B:05:59:8C][LE]> char-write-cmd 0x4c 01
:~$ [68:C9:0B:05:59:8C][LE]> char-read-hnd 0x49
```

The output will be something similar to:

```
> Characteristic value/descriptor: 80 4a f5 19 00 1c dc 19
```

The return value represents the values of the light intensity in eight 16-bit registers 0x14 to 0x1B according to the following structure:

clear_LSB	clear_MSB	red_LSB	red_MSB	green_LSB	green_MSB	blue_LSB	blue_MSB

The TCS3772 color sensor returns values between 0-2^16 depending on the light intensity:
clear (register 0x14 to 0x15) = 0x4A80 which converts to 19072
red (register 0x16 to 0x17) = 0x19F5 which converts to 6645
green (register 0x18 to 0x19) = 0x1C00 which converts to 7168
blue (register 0x1A to 0x1B) = 0x19DC which converts to 6620

The measurement range of the TCS372 color sensor can be highly adjusted by setting individual gain and integration time. Calibration and adjustment of the ambient light sensor is not integrated in the example firmware. You need to adjust the sensor's behaviour in the firmware according to the *TCS37727 datasheet*.

2.1.1.6 RGB-LED Control

Controlling the color of the RGB LED can be done by writing the specific values to the RGB LED configuration byte. Write data according to the scheme outlined below by using the following command:

```
:~$ [68:C9:0B:05:59:8C][LE]> char-write-cmd 0x58 01
```

The following scheme shows the assignment of the bits to control the single LEDs within the RGB using the configuration byte:

Byte ->	RGB LED Conf							
Bits ->	X	X	X	X	X	blue	green	red

Example Values for RGB LED Conf:
0x00 -> All LEDS off
0x01 -> RED LED on
0x02 -> GREEN LED on
0x04 -> BLUE LED on
0x07 -> ALL LEDS on

© PHYTEC Messtechnik GmbH 2015 L-812e_1

2.2 Access the Sensor Values automatically via *Python* Script

The following steps are a short recapitulation of what you already did while working with the QuickStart Guide. However, this section provides some additional useful information on how to access the sensor values via P*ython* script.

- To read the sensor values via *Python* script first remember the unique device address, or execute:
  `:~$ sudo hcitool lescan`
  after resetting of the phyNODE.
- Now call the manpage of the *Phyton* script with
  `:~$ python phyWaveBLE.py <device address>`

You will receive the following:

```
usage: phyWaveBLE.py [-h] [-n COUNT] [-t T] [-T] [-A] [-H] [-O] [-P] [-R] [-G]
                     [-Y] [-B] [-M] [-C] [-W] [-K] [--all]
                     host

positional arguments:
  host                    MAC of BT device

optional arguments:
  -h, --help              show this help message and exit
  -n COUNT                Number of times to loop data
  -t T                    time between polling
  -T, --temperature
  -A, --accelerometer
  -H, --humidity
  -O, --magnetometer
  -P, --barometer
  -R, --REDled
  -G, --GREENled
  -Y, --YELLOWled
  -B, --BLUEled
  -M, --MAGENTAled
  -C, --CYANled
  -W, --WHITEled
  -L, --color
  -K, --keypress
  --all
```

- To test the function type:
  `:~$ python phyWaveBLE.py --all <device address>`

The color of the RGB LED will change and rall values of the sensor available will be read subsequently. You will receive the following output:

```
Connecting to 00:12:4B:00:61:CE
RED led ON
Green led ON
Yellow led ON
Blue led ON
Magenta led ON
Cyan led ON
White led ON
('Temp: ', (24.40625, 17.543296541630))
('Humidity: ', (21.994401855468745, 66.6470947265625))
('Barometer: ', (98376, 23))
('Accelerometer: ', (-7.5, 25.75, 260.0))
('Magnetometer: ', (-0.823974609375, -6.500244140625, -31.1279296875))
('color: ', (15363, 6511, 6518, 6328))
```

All sensor values will be periodically updated. As all colors of the RGB LED where turned ON successively, their status will remain ON until you change the color. The following table explains the output of the different sensors.

Sensor	Value 1	Value 2	Value 3	Value 4
Temp[2]	Sensor Temperatur (°C)	IR-Temperatur (°C)		
Humidity	Sensor Temperatur (°C)	Relative Humidity (%)		
Barometer	Air Pressure (Pa)	Sensor Temperatur (°C)		
Accelerometer	Acceleration X (1 g/256)	Acceleration Y (1 g/256)	Acceleration Z (1 g/256)	
Magnetometer[1]	Field Strength X (µT)	Field Strength Y (µT)	Field Strength Z (µT)	
color[2]	White	Red	Green	Blue

Table 5: Sensor Values read from the Python Script

If you have any problems conduct the following steps:

Tools and Libraries:

- Check if the tools and libraries were installed to your Linux system by executing:

```
sudo apt-get install git python build-essential libglib2.0-dev
                                                libdbus-1-dev
```

1: The sensor is not calibrated and the values are afflicted with different offsets for each direction.
2: The sensor is not calibrated.

© PHYTEC Messtechnik GmbH 2015

Bluetooth:

- Verify that the USB BLE dongle is recognized by the system by executing *lsusb*:
  `:~$ lsusb`

You should receive:

```
Bus 002 Device 004: ID 0a12:0001 Cambridge Silicon Radio, Ltd Bluetooth Dongle (HCI mode)
```

	If the device has not been recognized by the system and you are using Linux as a virtual machine setup, refer to the setup procedure described in *section 3.1*.

If the USB BLE dongle is recognized correctly, start scanning for the phyNODE.

- Type:
  `:~$ sudo hcitool lescan`

At least one device should be recognized:

```
LE Scan ...
00:12:4B:00:61:CE (unknown)
00:12:4B:00:61:CE PBA-D-01
[..]
```

***Git*:**

- Ensure that the PHYTEC *git* repository was cloned properly by typing:
  `:~$ git status`

You should receive:

```
On branch master
 Your branch is up-to-date with 'origin/master'.
```

***Bluepy*:**

- Verify that the build of the *bluepy* project was successful by entering `make` within the ble-*cc26xx/ble_host_sw/bluepy/bluepy/* directory.

If you get

```
make: Nothing to be done for `all'.
```

the build was successful. If not do the following:

- First clean everything with:
  `:~$ make clean`

You should see

```
rm -f *.o bluepy-helper TAGS
```

- Now ensure that you are in the right directory and start the build again.

2.3 Accessing the Sensor Values manually in Apple iOS

To start programming for any *Apple* device you need several prerequisites.

- An *Apple* development computer (*iMAC*, *MacBook*) running *OSX*
- XCode available for free in the *MAC OSX* AppStore
- An *Apple* developer account (for publishing)

We start by creating a new *Xcode* project.

- Startup *Xcode*.
- Create a new *Xcode* project.
- Click "*Single View Application*".
- Enter a product name.
- Leave language to "objective-C" (this how-to ignores *Swift* for the moment).
- Choose a folder to save the newly created project.

For demonstration purposes we strictly stay in *ViewController.m* and *ViewController.h*, of course it is possible to separate functions to different files and/or classes, but we stay simple here.

- Click *ViewController.m* to open a small template file that starts with something similar to this:

```
/*
 * ViewController.m
 * BLE_example
 *
 * Copyright (C) 2015 PHYTEC Messtechnik GmbH
 *
 * This file is subject to the terms and conditions of the GNU Lesser
 * General Public License v2.1. See the file LICENSE in the top level
 * directory for more details.
 */

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController
```

First, before we start to activate Bluetooth, we define the sensors UUIDs to something more readable, e.g:.

```
define TEMP_DATA @"F000AA01-0451-4000-B000-000000000000"
```


© PHYTEC Messtechnik GmbH 2015 L-812e_1

Further information about the UUIDs on the phyNODE and additional examples can be found in *section 2.3.1.1*.

To stay simple we only use the CONF and DATA characteristics. We are not using the PERIOD characteristics which changes the data update interval of a sensor. Therefore by not writing any value to it 1 second will be the default period.

- Now we are adding the Delegate definitions and the variables:

```
/*
 * ViewController.h
 * BLE_example
 *
 * Copyright (C) 2015 PHYTEC Messtechnik GmbH
 *
 * This file is subject to the terms and conditions of the GNU Lesser
 * General Public License v2.1. See the file LICENSE in the top level
 * directory for more details.
 */

#import <UIKit/UIKit.h>

@import CoreBluetooth;
@import QuartzCore;

@interface ViewController : UIViewController
                           <CBCentralManagerDelegate, CBPeripheralDelegate>
{
    IBOutlet UITableView *sensorsTableView;
}
@property (strong, nonatomic) CBCentralManager *centralManager;
@property (strong, nonatomic) CBPeripheral *discoveredPeripheral;
@property (strong, nonatomic) NSMutableData *data;
```

With *CBCentralManagerDelegate* and *CBPeripheralDelegate* added to our *ViewController.h* we tell the OS where to look for the *Delegates* when receiving answers from either the phyWAVE target or from the IOS hardware.

	Most of *Apples iOS* and *Mac OS*'s APIs use some kind of delegate to interact with the developers program. Best practice is always to create all delegates as *Apple* states it in the developer library. Do not just use the delegates you need, completely implement all of them.

The three variables *centralManager*, *discoveredPeripheral* and *data* will be used for scanning for devices, services and characteristics and data storage.

ViewController.h is now complete for our demonstration project and we can go back to *ViewController.m* to start the actual development.

Please ensure that you initialize the *centralManager* and the data variable. Your *viewDidLoad* function should now look like this:

```
//we will save our connected Peripheral here
CBPeripheral *connectedPeripheral;

//we will save the RGB_LEDs characteristic here
CBCharacteristic *RGB_characteristic;
- (void)viewDidLoad {
    [super viewDidLoad];

    _centralManager = [[CBCentralManager alloc] initWithDelegate:self
                                                            queue:nil];
    _data = [[NSMutableData alloc] init];

}
```

	You could look for special services and/or characteristics and ignoring everything else. Just create a *NSArray* of UUIDs and add the services you want services to it, e.g. *services = @[[CBUUID UUIDWithString:TEMP_DATA,* *[CBUUID UUIDWithString:TEMP_CONF]];* You will only receive answers from these two services.

This project is build so you do not need an UI, you can see every output via the *Debug Console* in *XCode*.

- Initiate the *centralManager* with

```
_centralManager = [[CBCentralManager alloc] initWithDelegate:self
                                                        queue:nil];
```

IOS starts with a hardware check to see if Bluetooth is activated and if the right version is present. Like most of IOS's APIs you receive answers to function calls via delegates which you need to integrate into your software. The first delegate you will invoke is *centralManagerDidUpdateState*.

- Add the following function to your project:

```
- (void)centralManagerDidUpdateState:(CBCentralManager *)central {
    if (central.state != CBCentralManagerStatePoweredOn) {
        return;
    }
    if (central.state == CBCentralManagerStatePoweredOn) {
        [_centralManager scanForPeripheralsWithServices:nil options:nil];
        NSLog(@"STARTED looking for devices!");
    }
}
```


© PHYTEC Messtechnik GmbH 2015

Looking at the delegate you will see that we start the *scanForPeripherals* function if the Bluetooth state is "powered on". Otherwise we will leave the function because our *Apple* device lacks the necessary Bluetooth feature or they are not activated in the system settings.

In order to use the *centralManager* to find and connect to our phyNODE we need to add some more delegates:
In case we find a peripheral after scanning we need to add the *didDiscoverPeripheral*-Delegate.

- Add the following function to your project:

```
- (void)centralManager:(CBCentralManager *)central
 didDiscoverPeripheral:(CBPeripheral *)peripheral
 advertisementData:(NSDictionary *)advertisementData RSSI:(NSNumber
 *)RSSI {

    NSLog(@"DISCOVERED a peripheral %@ at %@", peripheral.name, RSSI);

    if (_discoveredPeripheral != peripheral) {
        // Save a local copy of the peripheral, so CoreBluetooth does
        // not get rid of it
        _discoveredPeripheral = peripheral;

        // And connect
        NSLog(@"CONNECTING to peripheral %@", peripheral);
        [_centralManager connectPeripheral:peripheral options:nil];
    }
}
```

As you see in the last line we connect to the peripheral found with *_centralManager connectPeripheral*, which leads us to the following delegate:

- Add the following function to your project:

```
- (void)centralManager:(CBCentralManager *)central
  didConnectPeripheral:(CBPeripheral *)peripheral {
    NSLog(@"CONNECTED to a peripheral : %@", peripheral);
    connectedPeripheral = peripheral;
    [_centralManager stopScan];
    NSLog(@"STOPPED scanning for devices");

    peripheral.delegate = self;

    [peripheral discoverServices:nil];
}
```

- To be able to react if the if the connection fails, add at least the function header to your project:

```
- (void)centralManager:(CBCentralManager *)central
didFailToConnectPeripheral:(CBPeripheral *)peripheral error:(NSError
*)error {
    NSLog(@"FAILED to connect");
    [self cleanup];
}
```

After these steps you will end up being connected to the phyNODE. With *[peripheral discoverServices:nil]*; you will start a scan for all services of the peripheral. Calling that function will bring you to the next delegate:

- Add the following function to your project:

```
- (void)peripheral:(CBPeripheral *)peripheral
    didDiscoverServices:(NSError *)error {
    if (error) {
        [self cleanup];
        NSLog(@"FAILED finding services");
        return;
    }
    NSLog(@"FOUND services!");
    for (CBService *service in peripheral.services) {
        [peripheral discoverCharacteristics:nil forService:service];
        NSLog(@"FOUND services! %@", service.UUID);
    }
}
```

didDiscoverServices returns an array of all services that have been found on our device. We step through all services and tell the IOS API that we want to have all characteristics of these services. The answer to these calls will be in a new delegate, which is called *didDiscoverCharacteristicsForService*. We finally can start and work with the device.

In this function we see all services for the first time so the first step is to activate the sensors and to subscribe to the DATA fields in order to get updates of the sensor data periodically. The following function only shows the activation and subscription of the TMP006 sensor, all other sensors are used exactly in the same way. Just change the UUID and another sensor will be activated.

 © PHYTEC Messtechnik GmbH 2015

- Add the following function to your project:

```
- (void)peripheral:(CBPeripheral *)peripheral
   didDiscoverCharacteristicsForService:(CBService *)service
   error:(NSError *)error {
    if (error) {
        NSLog(@"FAILED to discover characteristics on %@",
              peripheral.name);
        [self cleanup];
        return;
    }
    NSData* activateByte = nil;
    unsigned char bytes[] = {0x01};
    activateByte = [NSData dataWithBytes:bytes length:sizeof(bytes)];

    for (CBCharacteristic *characteristic in service.characteristics)
{
        NSLog(@"DISCOVERED characteristics %@ on %@",
              characteristic.UUID, peripheral.name);

         // Use the CONF UUIDs to activate the specific sensor
         // (we are writing a 0x01 byte)
         // Use the DATA UUIDs to subscribe to a value change
         // (sth. like an interrupt) (BLE standard calls
         // that Notification).
         // We will get fired to: didUpdateValueForCharacteristic
         // when a value changes (since we are using the
         // standard period, this will be every second).
         // You can change the period by writing the period time
         // in miliseconds to the PERIOD register (1000 for 1 second).
         //
         // Temp example : TEMP_CONF: 0xAA02
         //                TEMP_DATA: 0xAA01
         //                TEMP_PERIOD: 0xAA03

        if ([characteristic.UUID isEqual:[CBUUID
                          UUIDWithString:TEMP_CONF]]) {
            [peripheral writeValue:activateByte
             forCharacteristic:characteristic
             type:CBCharacteristicWriteWithResponse];
            NSLog(@"activating Temperature Sensor");
        }
        if ([characteristic.UUID isEqual:[CBUUID
                          UUIDWithString:TEMP_DATA]]) {
            [peripheral setNotifyValue:YES
             forCharacteristic:characteristic];
            NSLog(@"subscribing to Temperature Data Event");
        }
    }
}
```

We have activated our sensors and subscribed to any data changes, that leads us to the most important delegate. The function where we receive our data and can react to is *didUpdateValueForCharacteristic*.

This is again a long function so we only show its behavior for the TMP006 sensor.

- Add the following code to your project:

```
- (void)peripheral:(CBPeripheral *)peripheral
   didUpdateValueForCharacteristic:(CBCharacteristic *)characteristic
   error:(NSError *)error {
    if (error) {
        NSLog(@"Error didUpdateValueForCharacteristic");
        return;
    }
    if ([characteristic.UUID isEqual:[CBUUID
         UUIDWithString:TEMP_DATA]]) {

        float temp = [self calcTAmb:characteristic.value];

        NSLog(@"--------------------- TEMP_DATA --------------------
             -");
        NSLog(@"LONG UUID                   : %@", characteristic.UUID);
        NSLog(@"RAW DATA                    : %@", characteristic.value);
        NSLog(@"READABLE DATA               : %f", temp);
        NSLog(@"");
    }

}
```

We simply call the appropriate helper function depending on the sensor. You might want to save the return value of these functions to an array or create a debug output with *NSLog*. From here on it is up to you what you want to do with the data received.

This programming example has shown how you can communicate with the sensors on the phyNODE included in the Phytec IoT-Enablement-Kit 2. The example concentrates on the BLE functionality, not on the user interface part. It is up to you how you provide the data to your customers.

 © PHYTEC Messtechnik GmbH 2015

2.3.1 Supplementary Information

2.3.1.1 UUID explanations

Only a small part of the UUID depends on the sensor or the characteristic, most of it is creator specific, here Texas Instruments, and always the same.

Example for TEMP_DATA:
Long UUID: 0xF000AA01-0451-4000-B000-000000000000
Short UUID: 0xAA01 (the sensor specific part)

Even though an UUID is a hexadecimal value, we can ignore the 0x, *Apple*s API will still recognize it.

The following table summarizes the UUIDs which you can simply copy.

//**TMP006** (IR-Thermopile Temperature Sensor)
1. define TEMP_DATA @"F000AA01-0451-4000-B000-000000000000"
2. define TEMP_CONF @"F000AA02-0451-4000-B000-000000000000"
//**MPL3115A2** (Digital Pressure Sensor)
1. d efine BAROMETER_DATA @"F000AA41-0451-4000-B000-000000000000"
2. define BAROMETER_CONF @"F000AA42-0451-4000-B000-000000000000"
//**MMA8652FC** (Three-Axis Accelerometer)
1. define ACCELEROMETER_DATA @"F000AA11-0451-4000-B000-000000000000"
2. define ACCELEROMETER_CONF @"F000AA12-0451-4000-B000-000000000000"
//**TCS3772** (Color Sensor)
1. define COLOR_DATA @"F000AA91-0451-4000-B000-000000000000"
2. define COLOR_CONF @"F000AA92-0451-4000-B000-000000000000"
//**HDC1000** (Digital Humidity Sensor)
1. define HUMIDITY_DATA @"F000AA21-0451-4000-B000-000000000000"
2. define HUMIDITY_CONF @"F000AA22-0451-4000-B000-000000000000"
//**MAG3110FCR1** (Three-Axis Magnetometer)
1. define MAGNETOMETER_DATA @"F000AA31-0451-4000-B000-000000000000"
2. define MAGNETOMETER_CONF @"F000AA32-0451-4000-B000-000000000000"
//**RGB-LED**
1. define RGB_LED_CONF @"F000AA66-0451-4000-B000-000000000000"

Table 6: UUIDs for Apple iOS Example

If you want to create a *UITableView* to show the data to the user you might want to have a look at this simple tutorial on how a *UITableView* works (*http://www.appcoda.com/ios-programming-tutorial-create-a-simple-table-view-app/*).

Also an *UICollectionView* might work out for you (*http://sketchytech.blogspot.de/2015/05/uicollectionview-super-simple-example.html*).

The interface of the Bluetooth implementation of the *Apple* iOS is explained in the *iOS developer Library* (*https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/BestPracticesForInteractingWithARemotePeripheralDevice/BestPracticesForInteractingWithARemotePeripheralDevice.html#//apple_ref/doc/uid/TP40013257-CH6-SW1*).

2.3.1.2 Helper Functions to Convert the Sensor Data

In this section you will find the functions helping to convert the data received from the sensors on the phyNODE into physical values. More information on each sensor's data format and the calculations required can be found in the corresponding data sheets (*Table 3*).

2.3.1.2.1 Calculating the Pressure of MPL3115A2

```
-(int) calcPressure:(NSData *)data {
    char scratchVal[6];
    [data getBytes:&scratchVal length:6];
    uint32_t pressure;
    pressure = (scratchVal[3] & 0xff) | ((scratchVal[2] << 8) & 0xff00) |
               ((scratchVal[0] << 16) & 0xff0000 );
    return (int)(pressure / 6400);
}
```

2.3.1.2.2 Calculating the Humidity of HDC1000

```
-(float) calcHumidity:(NSData *)data {
    char scratchVal[data.length];
    [data getBytes:&scratchVal length:data.length];
    int32_t raw_hum = ((scratchVal[3] << 8) & 0xff00) | (scratchVal[2] &
                      0xff);
    float hum;
    hum = 100 * (raw_hum/65536.0);
    return hum;
}
```

 © PHYTEC Messtechnik GmbH 2015

2.3.1.2.3 Calculating the Temperature of TMP006

```
-(float) calcTAmb:(NSData *)data {
    char scratchVal[data.length];
    int16_t objTemp;
    int16_t ambTemp;
    [data getBytes:&scratchVal length:data.length];
    objTemp = (scratchVal[0] & 0xff)| ((scratchVal[1] << 8) & 0xff00);
    ambTemp = ((scratchVal[2] & 0xff)| ((scratchVal[3] << 8) & 0xff00));

    float temp = (float)((float)ambTemp / (float)128);

    return temp;
}
```

2.3.1.2.4 Calculating the Colors of TCS3772

```
-(int) getRedColor:(NSData *)data {
    char scratchVal[data.length];
    [data getBytes:&scratchVal length:data.length];
    int color = ((scratchVal[3] << 8) & 0xff00) | (scratchVal[2] & 0xff);
    return color;
}
-(int) getGreenColor:(NSData *)data {
    char scratchVal[data.length];
    [data getBytes:&scratchVal length:data.length];
    int color = ((scratchVal[5] << 8) & 0xff00) | (scratchVal[4] & 0xff);
    return color;
}
-(int) getBlueColor:(NSData *)data {
    char scratchVal[data.length];
    [data getBytes:&scratchVal length:data.length];
    int color = ((scratchVal[7] << 8) & 0xff00) | (scratchVal[6] & 0xff);
    return color;
}
-(int) getClearColor:(NSData *)data {
    char scratchVal[data.length];
    [data getBytes:&scratchVal length:data.length];
    int color = ((scratchVal[1] << 8) & 0xff00) | (scratchVal[0] & 0xff);
    return color;
}
```

2.3.1.2.5 Calculating the Acceleration Axis of MMA8652FC

```
-(int) getACCX:(NSData *)data{
    char scratchVal[data.length];
    [data getBytes:&scratchVal length:data.length];
    int axis = ((scratchVal[1] << 8) & 0xff00) | (scratchVal[0] & 0xff);
    return axis;
}
-(int) getACCY:(NSData *)data{
    char scratchVal[data.length];
    [data getBytes:&scratchVal length:data.length];
    int axis = ((scratchVal[3] << 8) & 0xff00) | (scratchVal[2] & 0xff);
    return axis;
}
-(int) getACCZ:(NSData *)data{
    char scratchVal[data.length];
    [data getBytes:&scratchVal length:data.length];
    int axis = ((scratchVal[5] << 8) & 0xff00) | (scratchVal[4] & 0xff);
    return axis;
}
```

2.3.1.2.6 Calculating the Magnetometer Axis of MAG3110FCR1

```
-(int) getMagX:(NSData *)data{
    char scratchVal[data.length];
    [data getBytes:&scratchVal length:data.length];
    int axis = ((scratchVal[1] << 8) & 0xff00) | (scratchVal[0] & 0xff);
    return ((axis *1.0) / (65536 / 2000));
}
-(int) getMagY:(NSData *)data{
    char scratchVal[data.length];
    [data getBytes:&scratchVal length:data.length];
    int axis = ((scratchVal[3] << 8) & 0xff00) | (scratchVal[2] & 0xff);
    return ((axis *1.0) / (65536 / 2000));
}
-(int) getMagZ:(NSData *)data{
    char scratchVal[data.length];
    [data getBytes:&scratchVal length:data.length];
    int axis = ((scratchVal[5] << 8) & 0xff00) | (scratchVal[4] & 0xff);
    return ((axis *1.0) / (65536 / 2000));
}
```


© PHYTEC Messtechnik GmbH 2015

3 Building and Debugging the Firmware

This chapter will guide you through setting up all the equipment and software required for building, flashing and debugging the firmware of the TI CC2650 SoC. We will use TI's Code Composer Studio (CCS)-IDE in combination with TI's XDS110 debugger. After setting up the development environment you will learn how to build and debug the firmware of the phyWAVE.

	TI supports only *Windows* based tools for development. Although, there is a tutorial "*how to install CCS within Linux*" and a tutorial that explains "*how to integrate the TI BLE project*", the method requires some manual effort. Following the two mentioned tutorials is not a clean *Linux* solution since tools like *Wine* and *Mono* are used. Because of that we recommend installing this environment in a *Windows* instance (either a Windows-based PC, or a virtual machine).

3.1 Setting up a Virtual Linux Machine

	If you have a Linux-based PC and *Windows* in a virtual machine, proceed to *section 3.2* and perform the installation described therein in the virtual machine.

As already mentioned in the introduction chapter, we recommend installing a Virtual Machine on your host-computer to have instances of both operating systems, *Linux* and *Windows*. This allows for simultaneous firmware development and BLE communication testing. This section explains how to set up a virtual *Ubuntu* machine in a *Windows* environment.

- Download and install Virtualbox from https://www.virtualbox.org/ (or any other virtualization software).
- Download Ubuntu 14.04 64bit from https://wiki.ubuntuusers.de/Downloads/Trusty_Tahr.
- Open Virtualbox and create a new virtual machine.
- Right-click on the created virtual machine and insert the downloaded Ubuntu image in "Storage->Controller:IDE".
- Start the newly created virtual machine and install Ubuntu by using the default settings.

Install Guest Additions:
  - In the virtual machine's settings bar on the top select *Devices->Insert Guest addition*
  - A window will open in the virtual machine. Agree to install guest additions to the virtualized system.
  - Reboot the virtual machine.

© PHYTEC Messtechnik GmbH 2015

- Update the system with:

```
sudo apt-get update && sudo apt-get dist-upgrade
```

- Install tools and libraries:

```
sudo apt-get install git python build-essential libglib2.0-dev
                                                      libdbus-1-dev
```

- Clone the PHYTEC *git*-repository that contains the *Python* script to interface with the phyNODE board via BLE:

```
:~$ git clone git://git.phytec.de/ble-cc26xx
```

- Within the repository navigate to ble-*cc26xx/ble_host_sw/bluepy/bluepy/*

```
:~$ cd ble-cc26xx/ble_host_sw/bluepy/bluepy
```

- Execute *make* to build the *bluepy* project

```
:~$ make
```

- Insert the BLE USB Dongle to your host computer and forward it to the running instance of your virtual machine:
  - In the VirtualBox menu select *Devices->USB Devices->CSR 8510 A10*

	If your host computer has an integrated Bluetooth adapter, the driver installation for the USB Bluetooth dongle may fail. In that case you need to deactivate the integrated Bluetooth adapter.

© PHYTEC Messtechnik GmbH 2015

3.2 Installing the Firmware Development Environment (Windows)

The installation of the TI development environment *CCS* has been tested with *MS Windows 7*. The following instructions explain how to install the environment and how to get started with software development and debugging. When using the TI *CCS* IDE in combination with the enclosed TI XDS110 debugger a free license will automatically be generated. Further this section explains how to open the project for the phyNODE.

- Download the most recent Windows version of the TI Code Composer Studio (CCS) from *www.ti.com/tool/ccstudio-wcs*
- Start the installation and use the default installation folder.
- Make sure to select support for *SimpleLink Wireless MCUs* during the installation process (*Figure 5*).

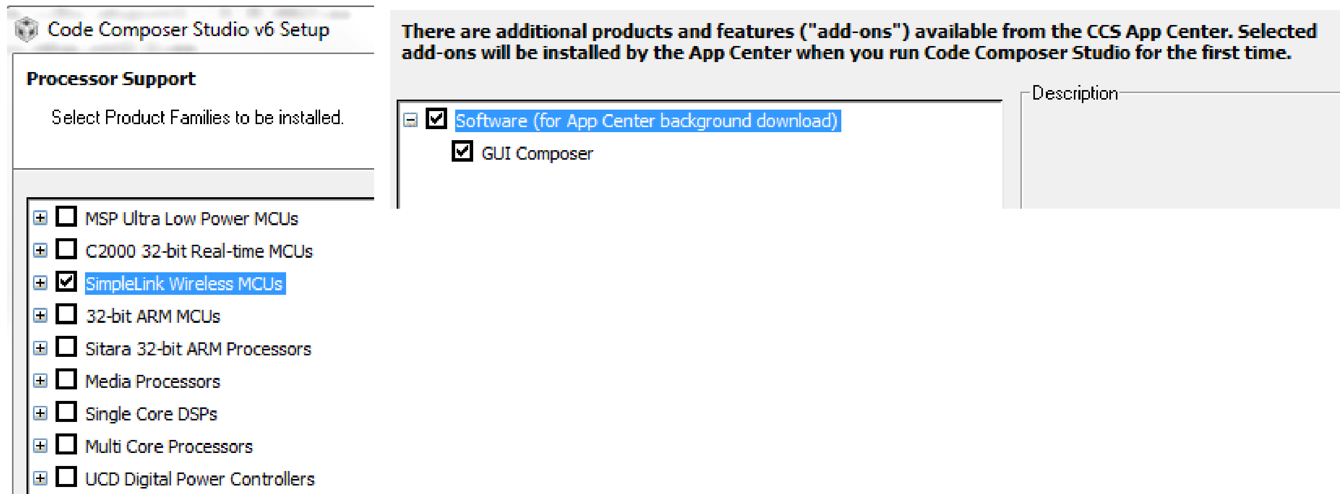

Figure 5: Selection of SimpleLink Wireless MCUs during the installation process

- Open *CCS* after installation and install all available updates (help > check for updates).
- Restart *CCS*.
- Open the App-Center (***help ► CCS App Center***) in *CCS* and install *TIRTOS*.
- Close *CCS*.
- Download the TI BLE-STACK-2.1 from *http://www.ti.com/tool/ble-stack* and install it with the default configuration. The default configuration includes all necessary tools to build the phyWAVE example project.
- Connect the TI XDS110 Debug adapter to the host computer with a micro-USB cable.
- Wait for the system to finish installing device drivers.
- Check if the XDS110 was recognized by the system (two devices should appear in a sub-menu called *Texas Instruments Debug Probes* within the *Windows* Device Manager)
- Download the versioning tool *git* from https://git-scm.com/download/win and install it using the default settings.

- Add the *git* installation path to your system's PATH variable by heading to ***Control Panel* ► *System* ► *Advanced System Settings* ► *Advanced* ► *Environment Variables*** and appending the *git* installation directory (*e.g. C:\Program Files\Git\bin*) to the path variable.
- Open a terminal, navigate to a location of your choice and clone the repository containing the source code of the demonstration project
```
:~$ git clone git://git.phytec.de/ble-cc26xx
```
- Start *CCS*.
- Open the phyNODE example project by clicking ***Project* ► *Import CCS Projects...* ► *Browse...*** and choose the location of the repository you just checked out

CCS should explore two projects within this repository.

- Choose *phyWAVE_CC2650* and *phyWAVE_CC2650Stack* and click ***Finish*** to import both projects.

As depicted in *Figure 6*, the *Project Explorer Tab* in *CCS* should show both projects.

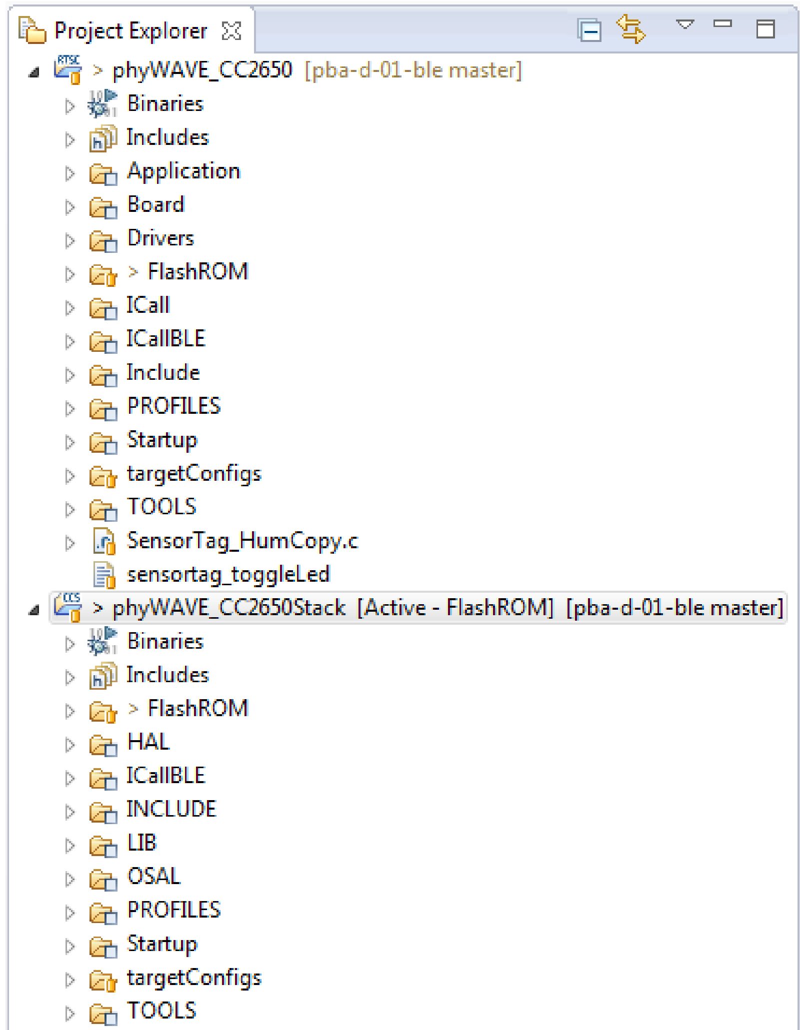

Figure 6: Imported projects within Code Composer Studio

© PHYTEC Messtechnik GmbH 2015 L-812e_1

Contents and Use of the of the default TI BLE-Stack-2.1 installation:

	The unmodified projects of the TI BLE-STACK-2.1 will not fit to the hardware of the phyNODE board without adaption. The Git-repository from PHYTEC, checked out in step 6, contains the adapted version of the TI *Simplelink Sensortag* project which fits the phyNODE board.

In addition to the adapted project for the phyNODE, you can browse the contents of the default TI BLE-Stack-2.1 for reference. In the directory chosen for installing the TI BLE-STACK-2.1 you will find the following folders:

- **Accessories:** installer for *BTool* and *Boundary* tool as well as hex files
- **Components:** source files, headers, and library headers which are shared among the BLE projects
- **Documents:** several important user guides with relevant info for developers
- **Projects:** the projects which are currently supported by the TI Sensortag2

	All these projects can be ported to the phyNODE with manageable effort.

3.3 Firmware Build and Debug

	In order to debug with *CCS*, the firmware of the XDS110 debugger needs to be updated before the first use. The update procedure is explained in *chapter 6.1* "Troubleshooting - XDS110 Firmware Update".

Build the project:

- Right-click on the *phyWAVE_CC2650Stack* project and choose *Build Project*.
- After the build has finished, repeat this step with the *phyWAVE_CC2650* project.
- The console in the bottom of the *CCS* environment should indicate a successful build for both of the two projects.

Start debugging the attached phyNODE board:

- Attach the XDS110 Debug adapter to the phyNODE board as shown in *Figure 7*.
- Connect the TI XDS110 debug adapter with a micro-USB cable to your computer.
- Connect the phyNODE board with a micro-USB cable to your computer.
- Choose the *phyWAVE_CC2650_Stack* project and click ***Debug***.

	It is important to start debugging with the *phyWAVE_CC2650_Stack* as the BLE stack needs to be flashed to the device before the application project *phyWAVE_CC2650*.

Starting the debugging procedure automatically flashes the *phyWAVE_CC2650_Stack* to the phyWAVE.

CCS will open the *Debug* perspective.

- Wait till flashing of *phyWAVE_CC2650_Stack* is finished.

Since the *phyWAVE_CC2650_Stack* is just a part of the example project, debugging is not yet possible.

- Click ***Terminate*** (Red rectangle in the top bar) to return to the edit perspective.
- Repeat the same procedure with the application project. Select the *phyWAVE_CC2650* project and click ***Debug***.
- After *CCS* has changed to the debug perspective and finished flashing, press ***Resume (F8)*** to start the program. The RGB LED on the phyNODE board will blink repeatedly green, signalizing that BLE advertising frames are send out.
- Debugging is now possible in the usual *Eclipse-IDE* manner.

© PHYTEC Messtechnik GmbH 2015 L-812e_1

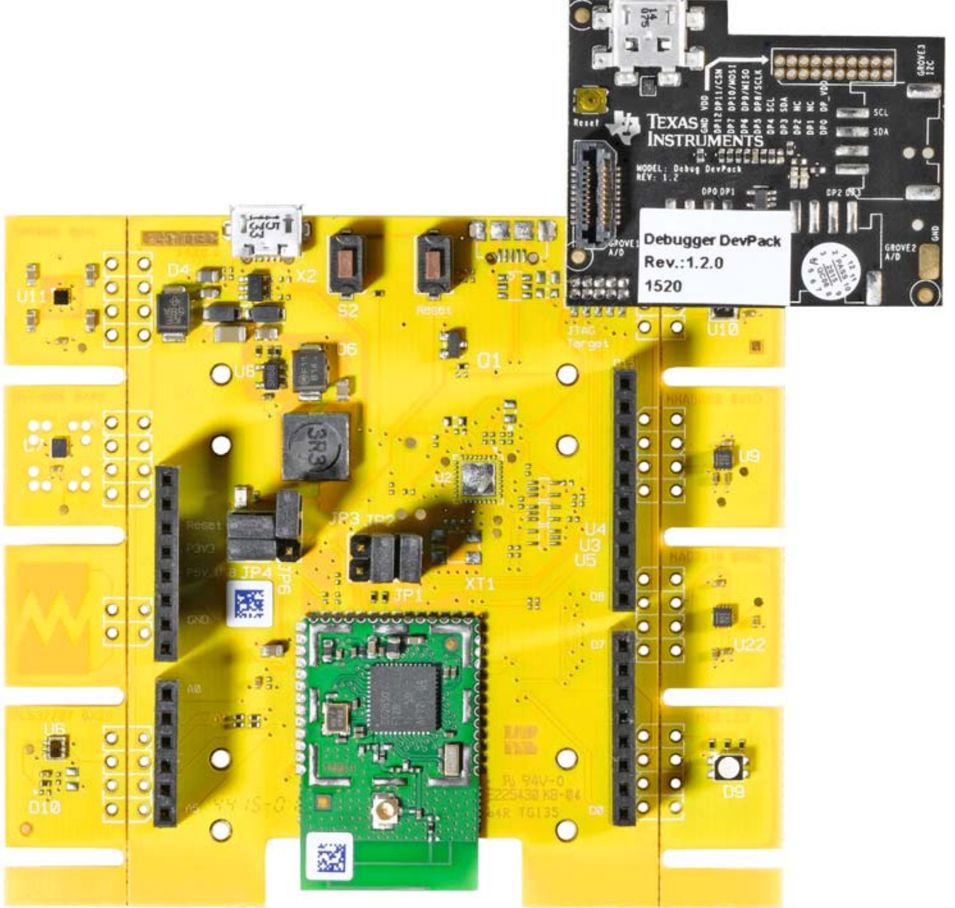

Figure 7: View of the phyNODE with attached Debugger

	You have successfully built and set up your development environment in *CCS*. You have also learned how to flash your application on the target and how to start debugging.

	Now you know how to work with *CCS* and how to develop and execute an application on the phyNODE board, so you are well prepared to start your project. The next chapter gives a short introduction of the driver integration to the CC2650's firmware.

© PHYTEC Messtechnik GmbH 2015 L-812e_1

4 Introduction to the Application Example

This section explains the architecture of the example application firmware. As the software includes many comments, we just show important architectural relationships within the software environment allowing you to find the right parts in the firmware to make your adaptions. Furthermore, it explains the necessary steps to implement a custom application.

	You can find useful supplementary information about the Sensortag Bluetooth firmware in the Texas Instruments *SimpleLink Software Developers Guide.* The SimpleLink Software Developers Guide gives detailed information of the architecture of the firmware with its interfaces that connect the Bluetooth Low Energy stack and the application.

4.1 The Structure of the Example Application Firmware

The demo program consists of two projects, the BLE stack and the main application. *Chapter 3* explains how to set up the IDE with the desired project. You should have an open instance of CCS with two subprojects, *phyWAVE_CC2650_Stack* and *phyWAVE_CC2650*.

phyWAVE_CC2650_Stack

Subproject *phyWAVE_CC2650_Stack* contains the BLE stack that can be used through defined interfaces. The implementation of the stack with its Bluetooth Low Energy specific layers (link layer, L2CAP, Generic Access Profile (GAP) and General Attribute Profile (GATT)) is hidden in a library and not available in source code. The Bluetooth stack is implemented in version 4.1. It is necessary to build the stack and flash it to the device before continuing with the application project.

phyWAVE_CC2650

Subproject *phyWAVE_CC2650* is the application project that includes all sensor drivers. Further, it includes initialization routines and allows for adapting the BLE related characteristics like the sleep interval (BLE link layer duty cycle). This project has to be adapted when new sensors are connected to the phyWAVE module for a custom application. *Figure 8* gives an overview of important source code files within the *phyWAVE_CC2650* project.

```
├── phyWAVE_CC2650
│   ├── Startup
│         ├── main.c      - Create tasks for sensor polling, entering RTOS
│         └── Board.h/c  - Pin configurations
│   ├── Application
│         ├── SensorTag.h/c     - Basic SensorTagTask with
│         │                       IO-Initialization routines
│         ├── SensorTag_Hum.h/c - Application to request humidity and temperature
│         │                       from the HDC1000 humidity sensor
│         └── SensorTag_...h/c  - Tasks to set up measurements on other specific
│         │                       environmental sensors
│   ├── PROFILES
│         ├── humidityservice.h/c - GATT profile definition, sensor-data length
│         ├── ...                 - Other GATT profile definitions.
│   ├── Board
│         ├── Devices
│               ├── sensor_hdc1000.h/c - HDC1000 sensor driver API
│               ├── ...                - Other driver implementations
├── phyWAVE_CC2650Stack
│   └── BLE Stack
```

Figure 8: Overview of the code structure within the phyWAVE_CC2650 project

4.2 Adding Custom Hardware to the Firmware

4.2.1 Adapting the Pin Muxing

If you want to connect custom hardware via a standard interface (I^2C, etc.) you can proceed to *section 4.2.2.*

If you connect proprietary hardware components to the phyWAVE board using GPIOs, you will most certainly need to adapt the pin muxing of the phyWAVE's firmware. The connection of the RGB LED to the phyNODE board will serve as an example to understand the pin muxing. The code listing below shows how the GPIOs for the RGB LED are defined in correspondence to the connection between RGB LED and phyWAVE (*Figure 9*). The definitions within the firmware are made in the file *Board.h*.

Pin muxing for the RGB LED defined in *Board.h*:

```
#define Board_LEDR                IOID_11
#define Board_LEDG                IOID_9
#define Board_LEDB                IOID_27
```

 © PHYTEC Messtechnik GmbH 2015

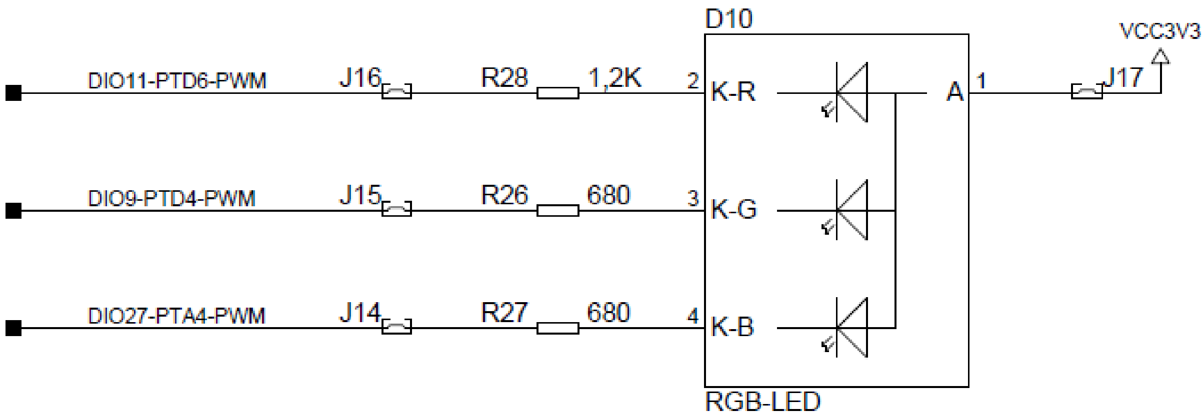

Figure 9: Schematic of the RGB LED

After you defined the new connections you can use already existing functions to set, or read the I/Os. In this example the function *PIN_setOutputValue* can be executed to turn on the LEDs (e.g. *PIN_setOutputValue(hGpioPin, Board_LEDR, Board_LED_ON))*.

If you have to write a new driver, you can use this function as template.

4.2.2 Adding new Device Drivers

Adding new device drivers is another task that will be necessary in case you are using custom hardware in combination with the phyWAVE Evaluation Board. If the new hardware connects to the phyNODE via standard interface (I^2C, etc.) you do not need to adapt the pin muxing as described in the previous section. To add new device driver files, a *C* file must be created along with its header file. This header file has to be included in the task function file (*e.g. SensorTag_Bar_Copy.c*) from which the functions of the newly created device file can be called.

How to create a new Device Driver

- Navigate to *C:\ti\simplelink\ble_cc26xx_xx_xx\Components\Components\ti-rtos\boards\sensortag\Devices*.
- Create a blank .c and .h file in that directory.
- Open the *phyWAVE_CC2650* project in *CCS*.
- Right-click the device folder and click ***Import*** ► ***import***.

An import window appears as shown in *Figure 10.*

- Expand the category *General* within the tree of import sources.
- Double-click ***File System*** **(1)**.
- In the *Import Window* click ***Advanced*** and check ***Create links in workspace*** **(2)**.
- Click ***Browse***.
- Select the directory from step 1 and click ***OK.***
- Select the newly created driver file and click ***Finish***.

- The added files will now appear in the *Device* folder of the *Project Explorer* and can be edited with a double-click.

For more details on how to integrate new device drivers please refer the “Peripherals and Drivers” section of the Texas Instruments *SimpleLink Software Developers Guide*.

Figure 10: Adding a new device driver

You can either use the new driver within an existing task, or you can create a new task for that specific sensor. If you use a new task, ensure that it is created in the initialization process described in the next section.

© PHYTEC Messtechnik GmbH 2015 L-812e_1

4.2.3 Initialization and Environmental Sensor Tasks

The activity diagram in *Figure 11* shows the startup and initialization process. During system-boot, GPIOs and other peripherals like I^{2}C are initialized. The *SensorTag task* implements the control of GPIO peripherals like push button and RGB LED.

Several tasks will be created in order to request different sensor values. The *main* function triggers the creation of tasks and returns the context to the RTOS. *Figure 11* depicts the initialization process at system startup. Any individual task may become runnable at different times in order to trigger reading the sensor value via I^{2}C. As can be seen in the software example for the tmp006 sensor below, the delay functions inside the while loop determine the wake up behavior of the task.

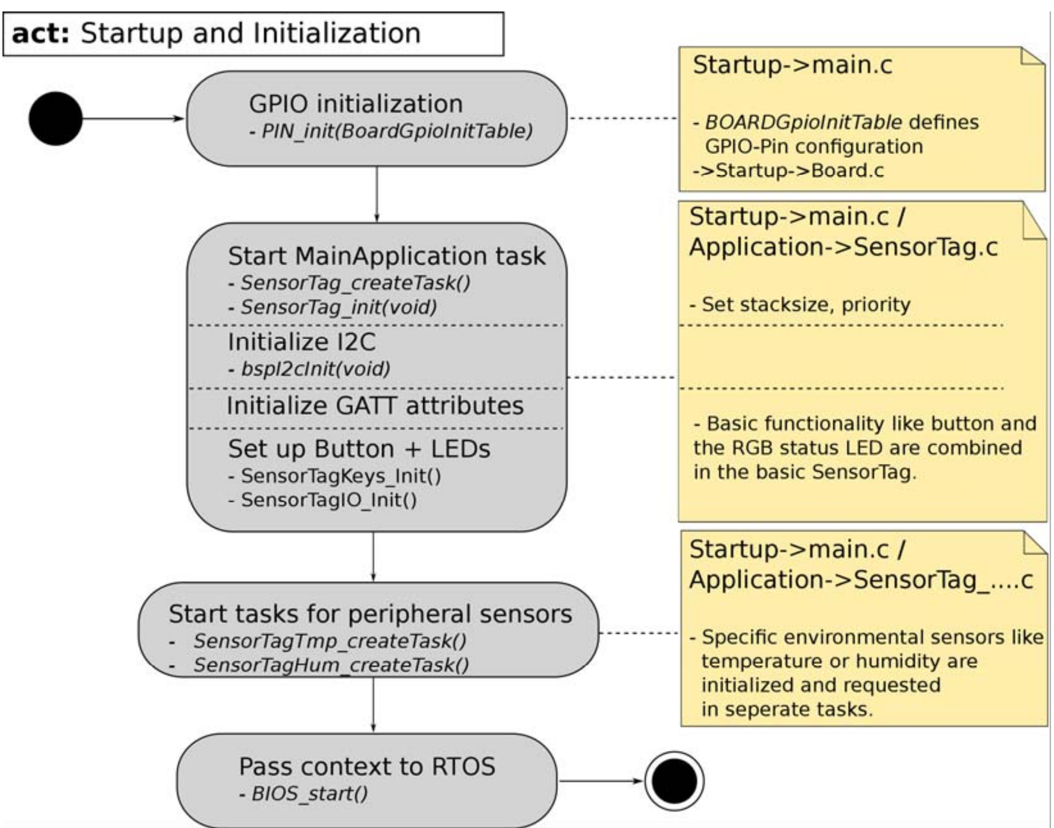

Figure 11: Initialization process at system startup

The task function of the particular task will be declared during task creation. All sensor driver APIs (Enable sensor, Read sensor value, etc.) will be called in their respective sensor task functions (*sensorTaskFxn*). The definition of the driver APIs are found in the *Devices* folder followed by the sensor name, e.g. *tmp_tmp006.c* (temperature sensor). The implementation of the task function method *sensorTaskFxn* of the SensorTag_Tmp task, located in *Application\SensorTag_Tmp.c*, shows an infinite loop inside the temperature measurement task. *Figure 12* shows the architecture of parallel tasks and their internal behavior.

```
// Task loop inside the TMP006 measurement task
while (true)
{
  if (sensorConfig == ST_CFG_SENSOR_ENABLE)
  {
    System_printf("tAp. TEMP SENSOR EANBLED\n");
    Data_t data;
    // Read data
    sensor_tmp_tmp006Enable(true);
    delay_ms(TEMP_MEAS_DELAY);
    sensor_tmp_tmp006Read(&data.v.tempLocal, &data.v.tempTarget);
    sensor_tmp_tmp006Enable(false);

    // Update GATT
    IRTemp_setParameter(SENSOR_DATA, SENSOR_DATA_LEN, data.a);

    // Next cycle
    delay_ms(sensorPeriod - TEMP_MEAS_DELAY);
  }
  else
  {
    delay_ms(SENSOR_DEFAULT_PERIOD);
  }
}
```

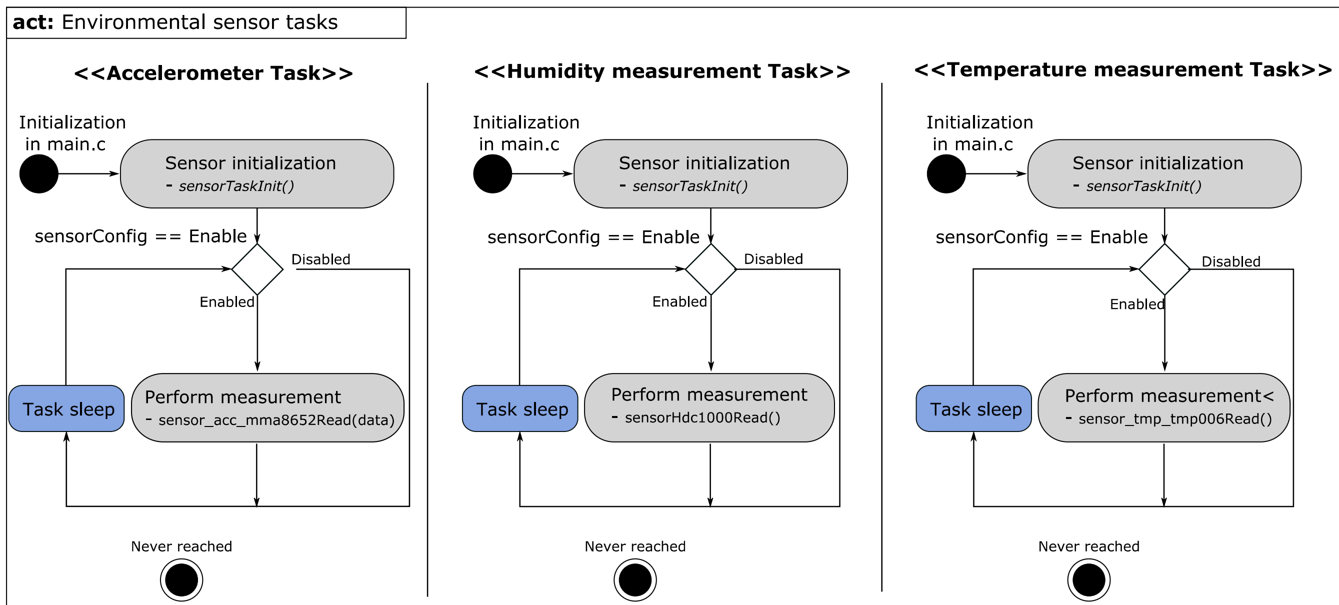

Figure 12: Architecture of Parallel Tasks and their Internal Behavior

5 Hardware Description

This section gives an overview of the peripherals and pin assignment of the phyNODE and phyWAVE module which is mounted to the phyNODE via a 40 pin DSC connector.

5.1 phyNODE

5.1.1 Overview

The phyNODE is depicted in *Figure 13* and *Figure 14*. It features many different interfaces and is equipped with the components as listed in *Table 3*, and *Table 7* to *Table 8*. For a more detailed description of each peripheral refer to the appropriate chapter listed in the applicable table. *Figure 13* highlights the location of each peripheral for easy identification.

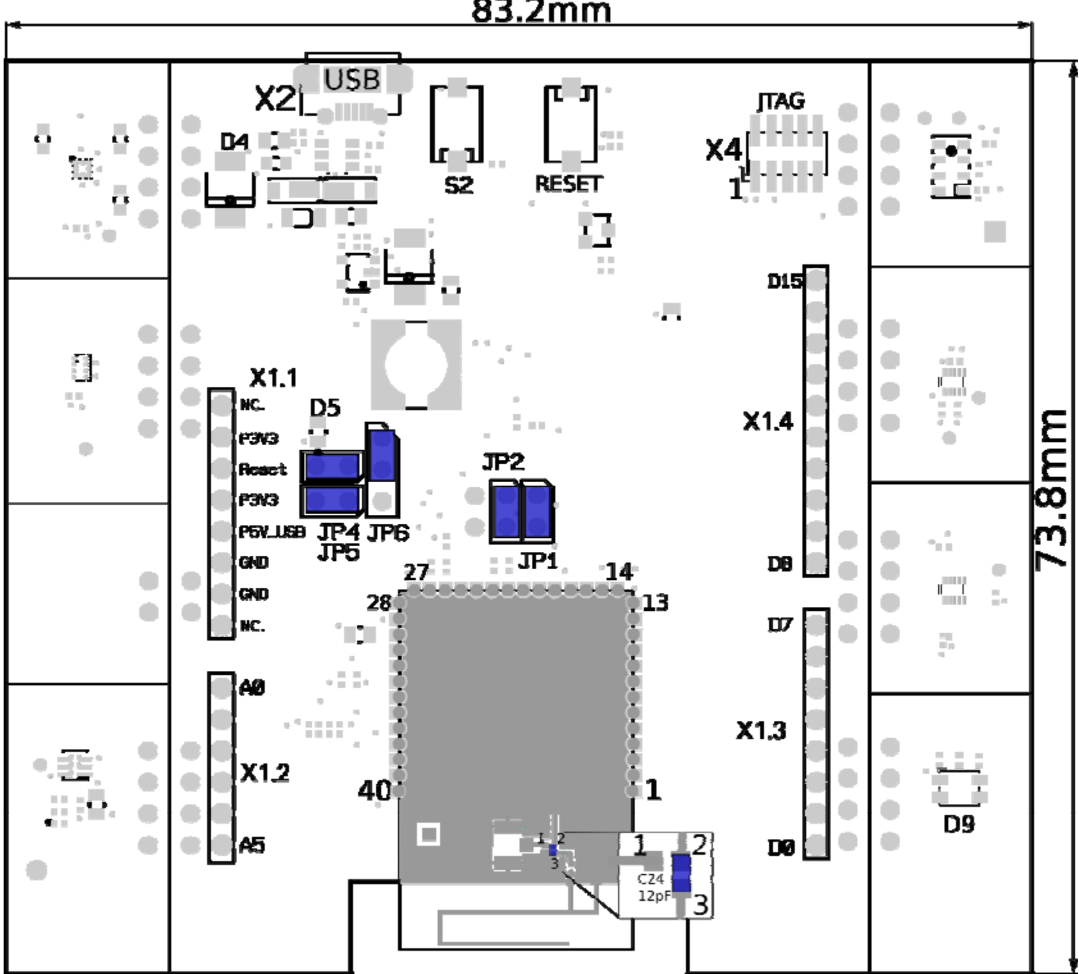

Figure 13: phyNODE Jumpers and Interfaces (top view)

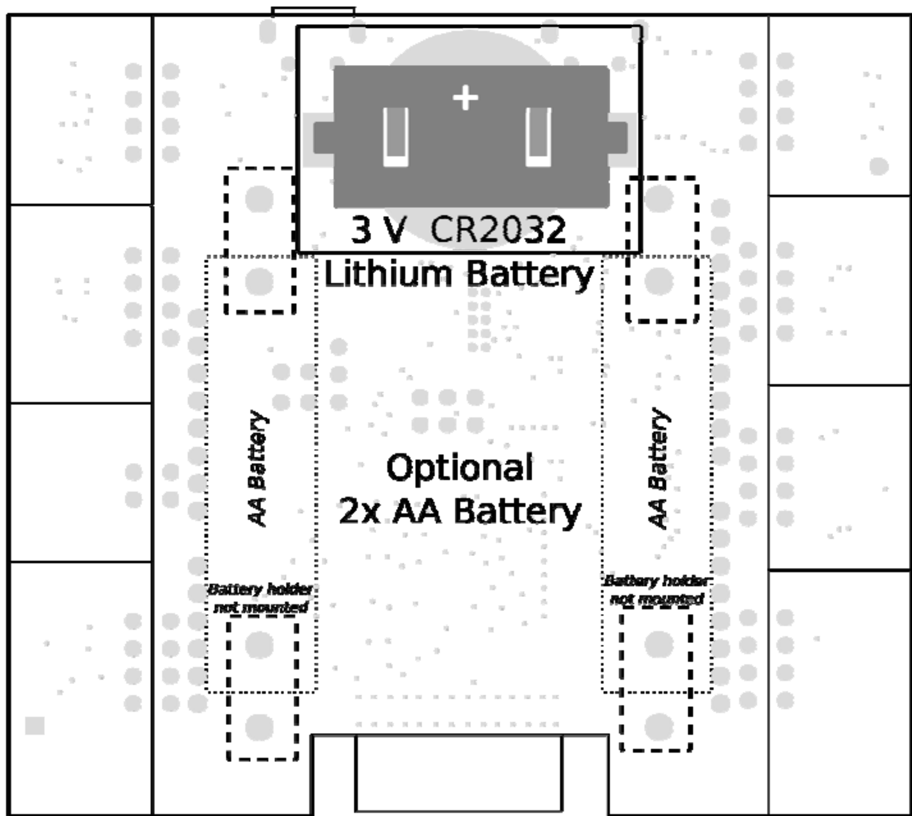

Figure 14: phyNODE Jumpers and Interfaces (bottom view)

5.1.1.1 Connectors and Pin Header

The phyNODE comes with three connectors.

Table 7 gives an overview of all interfaces. *Figure 13* and *Figure 14* highlight the location of the connectors on the phyNODE.

Reference Designator	Description	See Section
X1	*Arduino Uno* compatible connector (4 socket connectors 2.54 mm pitch; 1x6 pin; 2x8 pin; 1x10 pin)	*5.1.2.5*
X2	USB connector (USB Micro-AB)	*5.1.2.1.1*
X4	10 pin JTAG debug connector (2×5 pin header 1.27 mm pitch)	*5.1.2.6*
BAT1, BAT2, BAT3	Battery connector (BAT1 CR2032MFR coin cell; BAT2 + BAT3 2x AA, 3.0 V)	*5.1.2.1.1*

Table 7: Connectors on the phyNODE

© PHYTEC Messtechnik GmbH 2015 L-812e_1

5.1.1.2 Jumpers

The phyNODE comes pre-configured with five removable jumpers (JP) and one configuration capacitor (C24). The jumpers allow flexible configuring of a limited number of features for development purposes to the user. *Figure 12* indicates the location and the default configuration of the removable jumpers and the configuration capacitor on the board. In this figure a beveled edge indicates the location of pin 1. *Table 8* provides a comprehensive list of all carrier board jumpers. The default setting is marked in blue text.

Jumper	Description	See Section
JP1; JP2	Jumpers JP1 and JP2 disconnect the JTAG signals from the JTAG connector X4 JP1 - JTAG_DIO, JP2 JTAG_SCK). These jumpers must be closed to allow flashing and debugging.	*5.1.2.6*
open	JTAG is not connected to the JTAG connector X4	
closed	**JTAG is connected to the JTAG connector X4**	
JP4	Jumper JP4 connects the 3.3 V supply voltage to the environmental sensors. **Note:** The external power source used depends on the setting of JP6.	*5.1.2.1.1*
open	All environmental sensors are unpowered	
closed	**All environmental sensors are powered**	
JP5	Jumper JP5 connects the 3.3 V supply voltage to the phyWAVE module mounted on the DSC connector. **Note:** The external power source used depends on the setting of JP6.	*5.1.2.1.1*
open	The phyWAVE module is unpowered	
closed	**The phyWAVE module is powered**	
JP6	Jumper JP6 selects either a stationary power source via USB connector X2, or a battery installed at BAT1 (or BAT2 and BAT3) as power supply.	*5.1.2.1.1*
open	No power source is selected. The board remains unpowered	
1+2	**Stationary power source via USB connector X2 used as power source**	
2+3	A battery mounted at BAT1 (or BAT2 and BAT3) used as power source.	

Table 8: Jumpers on the phyNODE[3]

[3]: Default settings are in **bold blue** text

5.1.1.3 LEDs

The phyNODE is populated with two LEDs to indicate the status of the power supply voltage, and as user LED.

Figure 13 shows the location of the LEDs. Their function is listed in the table below:

LED	Color	Description	See Section
D5	red	Indicates presence of the supply voltage	*5.1.2.2*
D9	RGB	User LED (ports DIO_9, DIO_11, DIO_27 of phyWAVE)	*5.1.2.4*

Table 9: phyNode LEDs Descriptions

5.1.1.4 Switches

The phyNODE is populated with two switches, one to reset the phyWAVE and another as user button.

*Figure 13*shows the location of the switches. Their function is listed in the table below:

Switch	Description	See Section
S1	Reset Button	-
S2	User Button (ports DIO_6 of phyWAVE)	*5.1.2.4*

Table 10: Switches on the phyNODE

 © PHYTEC Messtechnik GmbH 2015

5.1.2 Functional Components on the phyNODE

This section describes the functional components of the NODE. Each subsection details a particular connector/interface and associated jumpers for configuring that interface.

5.1.2.1 Power Supply

	Do not change modules or jumper settings while the phyNODE is supplied with power!

5.1.2.1.1 Power Connectors (X2) (BAT1,2,3)

The phyNODE allows two different ways to be supplied with power. Depending on your order you will find one, or two of the following connectors on your module :

1. an USB Micro-AB connector (X2) to connect a standard +5 V USB power supply, and/or
2. a battery holder for either a CR2032MFR coin cell (BAT1), or two battery holders for two 3 V AA batteries (BAT2 and BAT3) (*Figure 2* and *Figure 14*)

	The USB connector X2 can only be used to supply the phyNODE. It does not provide an USB interface.

Jumper JP6 connects either the USB Micro-AB connector X2 (JP6: 1+2), or battery holder BAT1, or BAT2 and BAT3 (JP6: 2+3) to supply the phyNODE. Because of that, batteries installed can not be used as backup power supply in case of a power drop at X2.

	For measuring purposes jumpers JP4 and JP5 allow to disconnect the phyWAVE module (JP5) and/or the environmental sensors of the phyNODE (JP4) from the external power source.

5.1.2.2 Power LED D5

The red LED D5 right above jumper JP4 (*Figure 13*) indicates the presence of the 3.3 V supply voltage.

5.1.2.3 Environmental Sensors

The phyNODE is equipped with various environmental sensors. All environmental sensors installed on the phyNODE board are connected via I^2C. The following table lists all sensors and their I^2C address:

Sensor Type	I^2C Address	Detailed Information
Color Sensor (U6)	0x29	*TCS3772*
Digital Pressure Sensor (U10)	0x60	*MPL3115A2*
Three-Axis Accelerometer (U9)	0x1D	*MMA8652FC*
Three-Axis Magnetometer (U22)	0x0E	*MAG3110FCR1*
IR-Thermopile Temperature Sensor (U11)	0x41	*TMP006*
Digital Humidity Sensor (U7)	0x43	*HDC1000*

Table 11: Environmental Sensors on the phyNODE

Sensors and user I/O (capacitive touch field and an RGB-LED) can be broken off and reconnected via ribbon cable. This allows placing individual sensors, or actors to a user-defined location independent from the phyNODE's position. The maximum length depends on the ribbon cable used, shielding and I^2C speed.

5.1.2.4 RGB LED (D9) and User Button (S2)

The phyNODE provides one RGB LED (D9) and a user button (S2). The RGB LED as well as the user button are directly connected to the ports of the phyWAVE module, and thus, to the CC2650.

The following table lists the ports used.

Peripheral component	CC2650 GPIO	Note
RGB_LED RED	DIO_11	active low
RGB_LED GREEN	DIO_9	active low
RGB_LED BLUE	DIO_27	active low
User Button S2	DIO_6	-

Table 12: Ports used for RGB LED (D9) and User Button (S2)

 © PHYTEC Messtechnik GmbH 2015

5.1.2.5 Arduino Connector (X1)

X1 is an *Arduino Uno* compatible connector. It is separated into four different connectors (1x6 pin; 2x8 pin; 1x10 pin) with 2.54 mm pitch and allows the attachment of additional *Arduino* shields, such as a motor-driver, battery or e-paper display shield.

Most pins of X1 connect directly to the ports of the phyWAVE module. *Table 15* shows the relation between the *Arduino* compatible connector X1 and the ports of the phyWAVE.

	In contrast to the standard *Arduino* connector specification only DIO23 to DIO30 can be used as analog input. Also, all DIO-pins on the CC2650 can be used with any special function (UART, I²C, SPI or PWM).

Signal	ST	SL	Description
X1.1 - N.C.	-	N.C.	Not connected
X1.1 – 3V3	PWR_O	3.3 V	
X1.1 - nRESET	I/O	3.3 V	**nRESET**
X1.1 - 3V3	PWR_O	3.3 V	
X1.1 – 5 V	PWR_O	5 V	
X1.1 - GND	-	GND	Ground
X1.1 - GND	-	GND	Ground
X1.1 – N.C.	-	N.C.	Not connected
X1.2 - A0	I/O	3.3 V	DIO15
X1.2 - A1	I/O	3.3 V	DIO16, **JTAG_TDO**[4]
X1.2 - A2	I/O	3.3 V	DIO12
X1.2 - A3	I/O	3.3 V	DIO10
X1.2 - A4	I/O	3.3 V	DIO13, **I²C-SDA**[4]
X1.2 - A5	I/O	3.3 V	DIO14, **I²C-SCL**[4]
X1.3 - D0	I/O	3.3 V	DIO7
X1.3 - D1	I/O	3.3 V	DIO8
X1.3 - D2	I/O	3.3 V	DIO17, **JTAG_TDI**[4]
X1.3 - D3	I/O	3.3 V	DIO27, **RGB-LED Blue Channel**[4]
X1.3 - D4	I/O	3.3 V	DIO29
X1.3 - D5	I/O	3.3 V	DIO11, **RGB-LED RED Channel**[4]
X1.3 - D6	I/O	3.3 V	DIO25
X1.3 - D7	I/O	3.3 V	DIO24

Table 13: Pinout of the Arduino Connector X1

[4]: These pins are used on the phyNODE with the function specified in bold. Hence, special care must be taken if these pins are to be used by a device plugged in the *Arduino* connector.

© PHYTEC Messtechnik GmbH 2015

X1.4 - D8	I/O	3.3 V	DIO_4, **User-Button S2**[5]
X1.4 - D9	I/O	3.3 V	DIO_9, **RGB-LED Green Channel**[5]
X1.4 - D10	I/O	3.3 V	DIO_2
X1.4 - D11	I/O	3.3 V	DIO_4, **Capacitive Sensor**[5]
X1.4 - D12	I/O	3.3 V	DIO_5
X1.4 - D13	I/O	3.3 V	DIO_3
GND	-	GND	Ground
AREF	REF_0	3.3 V	Analog reference voltage
X1.4 - D14	I/O	3.3 V	DIO_20
X1.4 - D15	I/O	3.3 V	DIO_21

Table 13: Pinout of the Arduino Connector X1 (continued)

5.1.2.6 JTAG Interface (X4)

The 4-pin JTAG interface of the phyNODE is accessible at connector X4 (*Figure 13*). It serves to flash and debug the firmware of the TI CC2650 MCU on the phyWAVE module using TI's SimpleLink SensorTag Debugger DevPack (XDS110 debugger) which comes with the kit. This interface is compliant with JTAG specification IEEE 1149.1 or IEEE 1149.7. When referencing contact numbers note that pin 1 located at the angled corner. Pins towards the labeling “JTAG” are even numbered

Pin #	Signal Name	ST	SL	Description
1	VCC3V3	O	3.3 V	JTAG reference voltage
2	TARGET_SWD_DIO	O	3.3 V	JTAG Test Mode Select Signal
3, 5	GND	-		Ground
4	TARGET_SWD_CLK	I	3.3 V	JTAG Test Clock Signal
6	TARGET_TDO	I/O	3.3 V	JTAG Test Data Output
8	TARGET_TDI	I	3.3 V	JTAG Test Data Input
10	TARGET_nRESET	O	3.3 V	JTAG Reset

Table 14: JTAG Connector X4

Jumpers JP1 and JP2 must be closed to allow programming and debugging of the phyWAVE firmware.

[5]: These pins are used on the phyNODE with the function specified in bold. Hence, special care must be taken if these pins are to be used by a device plugged in the *Arduino* connector.

 © PHYTEC Messtechnik GmbH 2015 L-812e_1

5.2 phyWAVE CC2650 Module

The phyWAVE CC2650 module is a 19 mm x 29 mm PCB which integrates the TI CC2650 SoC, two clock domains (a 24 MHz oscillator and a 32.768 kHz oscillator used for the internal RTC and during sleep modes), a PCB antenna, as well as an U.FL RF connector to attach an external antenna. Most GPIOs and interfaces, such as I^2C, or SPI of the SoC are brought out at the modules DSC connector. *Figure 15* shows an abstracted figure of the phyWAVE module. The pin assignment of the DSC connector is described in *Table 15*.

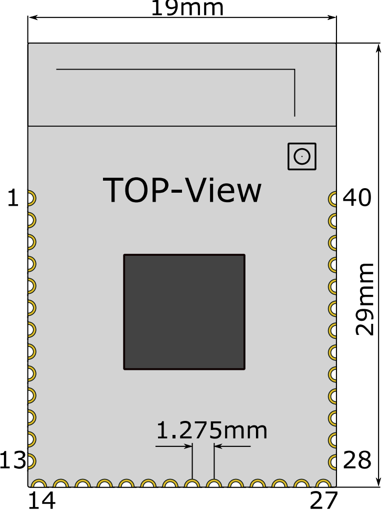

Figure 15: DSC connector of the phyWAVE module

5.2.1 DSC Connector of the phyWAVE

Most of the controller pins have multiple multiplexed functions. As most of these pins are connected directly to the Arduino compatible connector the alternative functions are available by using the CC2650 pin muxing options. Signal names and descriptions in *Table 15* however, are in regard to the specification of the phyNODE and the functions defined therein. Please refer to the *TI CC2650 MCU Technical Reference Manual* to get to know about alternative functions. In order to utilize a specific pin's alternative function the corresponding registers must be configured within the firmware of the CC2650 SoC.

Pin #	phyWAVE CC2650	Function / Signal Name	Arduino Connector on phyNODE
1	GND1	Ground	
2	DIO_0	GPIO	-
3	DIO_1	GPIO	-
4	DIO_2	GPIO	X1.4 - D10
5	DIO_3	GPIO	X1.4 - D13
6	DIO_4	Capacitive-Sensor[6]	X1.4 - D11
7	DIO_5	GPIO	X1.4 - D12
8	DIO_6	User-Button S2[6]	X1.4 - D8
9	DIO_7	GPIO	X1.3 - D0
10	DIO_8	GPIO	X1.3 - D1
11	DIO_9	RGB-LED Green Channel[6]	X1.4 - D9
12	DIO_10	GPIO	X1.2 - A3
13	DIO_11	RGB-LED Red Channel[6]	X1.3 - D5
14	GND2	Ground	X1.1 - GND
15	DIO_12	GPIO	X1.2 - A2
16	DIO_13	I^2C SDA[6]	X1.2 - A4
17	DIO_14	I^2C SCL[6]	X1.2 - A5
18	DIO_15	GPIO	X1.2 - A0
19	JTAG_TMSC	cJTAG Data	-
20	JTAG_TCKC	cJTAG Clock	-
21	DIO_16	JTAG TDO[6]	X1.2 - A1
22	DIO_17	JTAG TDI[6]	X1.3 - D2
23	DIO_18	GPIO	-
24	DIO_19	GPIO	-
25	DIO_20	GPIO	X1.4 - D14
26	DIO_21	GPIO	X1.4 - D15
27	GND3	Ground	
28	DIO_22	GPIO	
29	GND4	Ground	
30	VDD_IN	VCC3V3_Mod	
31	nRESET	Reset	X1.1 - nRESET
32	DIO_23	GPIO	-
33	DIO_24	GPIO	X1.3 - D7
34	DIO_25	GPIO	X1.3 - D6
35	DIO_26	GPIO	-
36	DIO_27	RGB-LED Blue Channel[6]	X1.3 - D3
37	DIO_28	GPIO	-
38	DIO_29	GPIO	X1.3 - D4
39	DIO_30	GPIO	-
40	GND5	Ground	

Table 15: Pin Assignment of the CC2650 phyWAVE Module

[6]: These pins are used on the phyNODE with the function specified. Hence, special care must be taken if these pins are to be used by a device plugged in the *Arduino* connector.

© PHYTEC Messtechnik GmbH 2015 L-812e_1

5.2.2 Antennas of the phyWAVE

The phyWAVE is equipped with a PCB antenna. An additional U.FL. RF connector is available at X1 to allow connecting an external antenna instead. In order to use an external antenna the position of capacitor C24 must be changed. The following table shows the correct settings:

Capasitor	Description
C24	Capacitor 24 allows to chose between the integrated PCB antenna or, the U.FL. connector for an external antenna.
open	No antenna is connected
1+2	External antenna via U.FL. connector selected
2+3	**On board PCB antenna selected**

Table 16: Configuration of the Antenna on the phyWAVE

6 Troubleshooting

6.1 XDS110 Firmware update

In order to be able to debug with *CCS*, the firmware of the XDS110 debugger needs to be updated. Perform the update by opening a console in your *Windows* environment; navigate to the *xdsdfu* utility found in the *CCS* install directory at *C:\ti\ccsv6\ccs_base\common\uscif\xds110*. Update the firmware by using the following two commands within that directory:

```
xdsdfu -m
xdsdfu -f firmware.bin -r
```

6.2 Debug Error

The TI XDS110 debugger is used for debugging the phyWAVE board. If another debugger than the XDS110 is selected in *CCS* the following error may appear when trying to debug or flash the firmware.

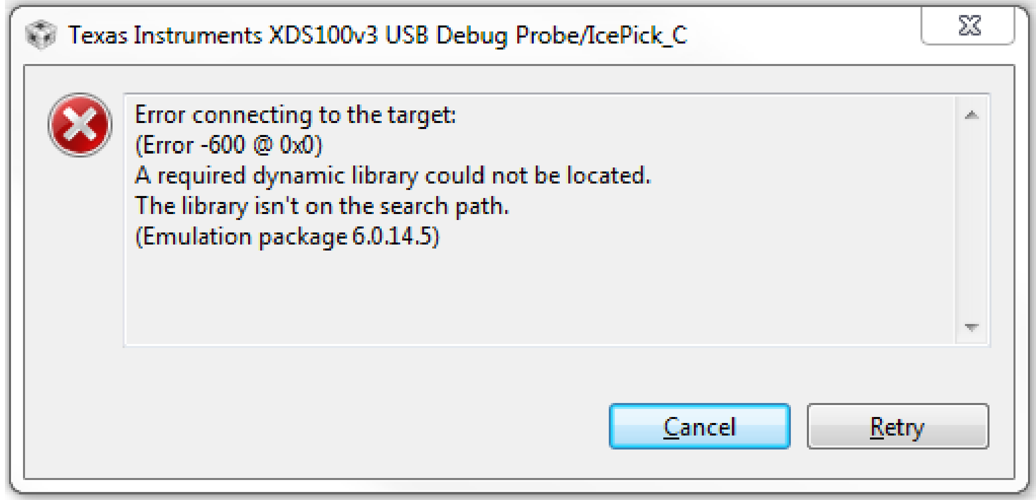

Figure 16: Error in case of a wrong setting in CCS

If you face this error, right-click on the project and choose properties as shown in *Figure 17*. Click on ***General*** and select the ***Texas Instruments XDS110 USB Debug Probe*** in the *Connection* field.

© PHYTEC Messtechnik GmbH 2015 L-812e_1

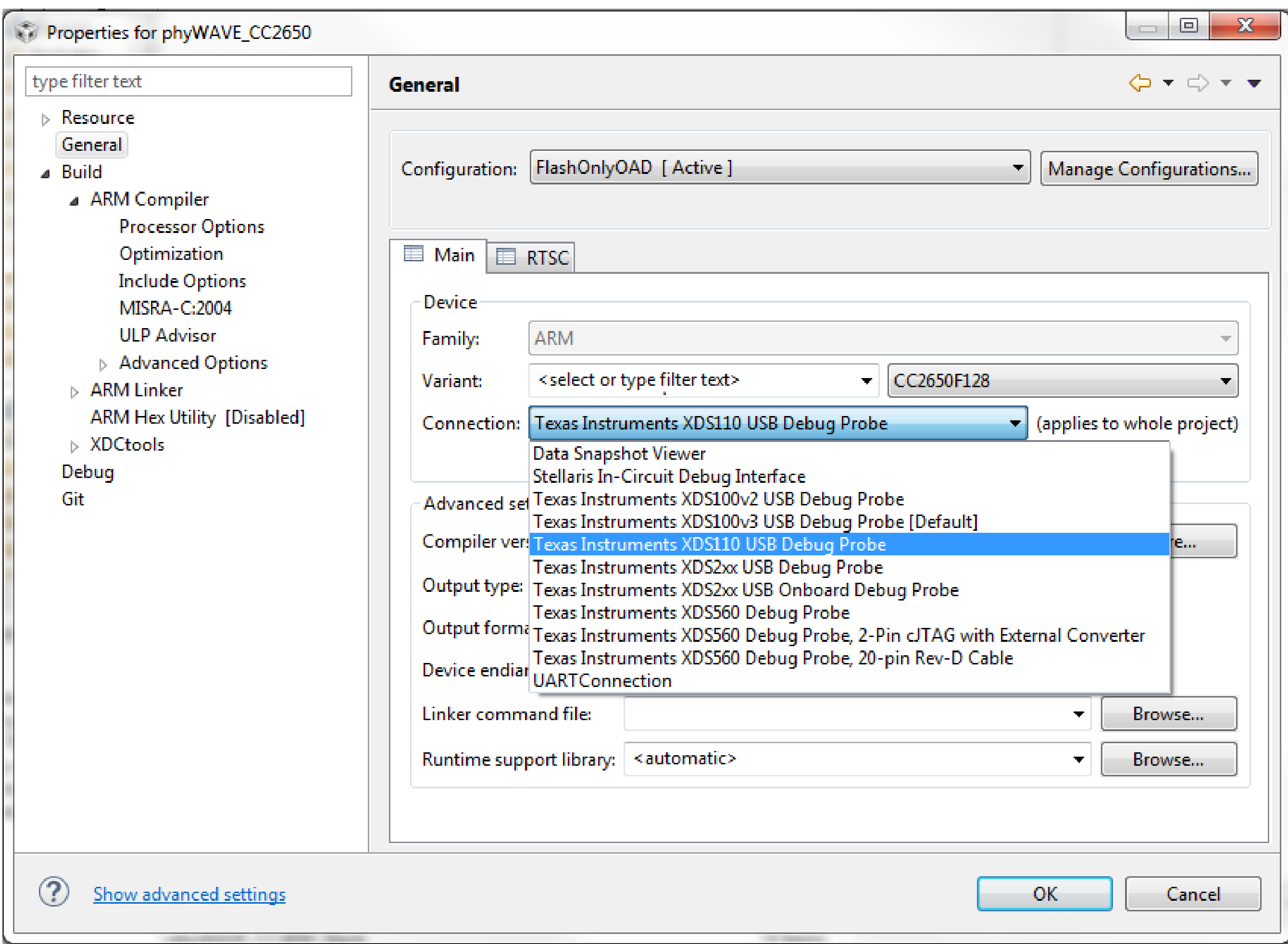

Figure 17: Setting up the correct (XDS110) Debugger

6.3 Compiler optimization Level

Depending on the optimization chosen within *CCS*, code size and RAM usage may vary. *Figure 18* shows an effective optimization level within *CCS*. To change the optimization level right-click on the phyWAVE_CC2650 project and choose ***Build* ▶ *ARM Compiler* ▶ *Optimization***. Depending on your application, you may have to find out the perfect setting.

The resulting code size and RAM usage is stored in a *.map* file in the project directory:
[..]\pba-d-01-ble\ble_fw\Projects\ble\phyWAVE\CC26xx\CCS\phyWAVE_CC2650\FlashOnlyOAD\phyWAVE.map

```
MEMORY CONFIGURATION

    name              origin    length     used      unused    attr    fill
-----------------  --------  ---------  --------  --------  ----  -------
  FLASH              00000000  0000dfff  0000ba10  000025ef  R         X
  SRAM               20000000  0000445f  00003c95  000007ca  RW        X
```

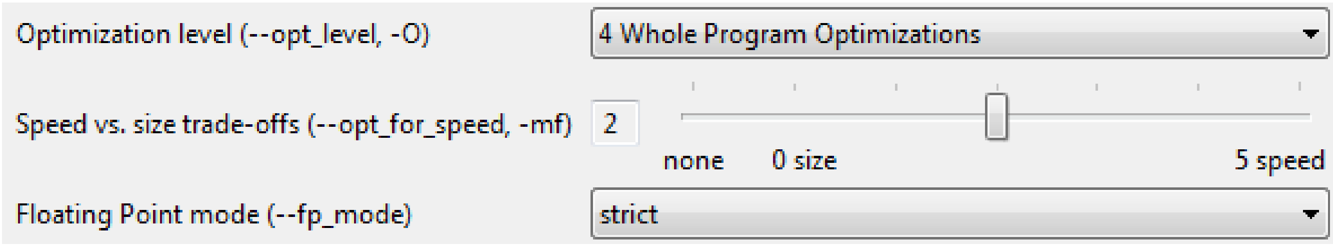

Figure 18: Optimization Settings in CCS

6.4 Device does not start without Connection to the XDS110 Debugger

If the flash section of the CC2650 has been completely erased, the application program may not start if the debugger is not attached to the phyNODE board. This issue is caused by wiping the hardware configuration field, called *Customer Configuration Registers (CCFG)*. The flash section that contains the *CCFG* Registers of the CC2650 SoC is not erased during regular debug sessions with *CCS*. The *TI SmartRF Flash Programmer Software*, in contrast, allows executing a "Forced Mass Erase".

The CC2650 SoC implements a bootloader in its ROM section that is able to communicate with an external device over serial interfaces. The bootloader is described in the *TI CC2650 Reference Manual* in section 8 and allows the implementation of over-the-air updates (OAD). The original SensorTag application implements (OAD); however, the over-the-air update feature has not been tested with the example project in combination with the phyWAVE board. Therefore, the bootloader has to be deactivated in the example project in combination with the phyWAVE board as long as the OAD feature is not being used.

The startup behavior is controlled by the *CCFG Registers* (*TI CC2650 Reference Manual* in section 9.1.1*)*. The *CCFG Registers* are located at the end of the flash section and reflect the hardware configuration of the device. This flash section is either pre-configured during device production or written during flashing of the CC2650 SoC.

There are two different linker scripts which can be used for *the phyWAVE_CC2650* project in *CCS*.

1. *cc26xx_ble_app.cmd*
2. *cc26xx_ble_app_ota.cmd*

The linker scripts have to be imported to the *CCS* workspace to the location ***phyWAVE_CC2650* ► *TOOLS***.

Both linker scripts differ in the listed memory section; *cc26xx_ble_app_ota.cmd* does not add the *CCFG* register section to the hex files while *cc26xx_ble_app.cmd* does. If you do not want to use the OAD feature make sure to select the *cc26xx_ble_app.cmd* linker script to your CCS workspace.

© PHYTEC Messtechnik GmbH 2015 L-812e_1

7 Revision History

Date	Version #	Changes in this manual
10.12.2015	Manual L-812e_1	First edition. Describes the phyNODE Evaluation Board (PCB 1426.2) with the phyWAVE module (PWA-A-001).

© PHYTEC Messtechnik GmbH 2015 L-812e_1

Document: IoT-Enablement-Kit
Document number: L-812e_1, December 2015

How would you improve this manual?

Did you find any mistakes in this manual? **page**

Submitted by:

Customer number:

Name:

Company:

Address:

Return to:

PHYTEC Messtechnik GmbH
Postfach 100403
D-55135 Mainz, Germany
Fax : +49 (6131) 9221-33

Published by

© PHYTEC Messtechnik GmbH 2015

Ordering No. L-812e_1
Printed in Germany