
DA1458x Prototype Bring-up Guide



Agenda



Problem Definition

System Overview

First power-up

Loading test firmware

Verification

Testing the sleep clock



Problem Definition

Bring-up complexity

- Bringing up hardware prototypes based on the DA14580 and DA14581 obviously involves the hardware itself, but it also requires a set of tools and SoC firmware.
- Because there are so many parameters in play, it is often hard to troubleshoot issues, and troubleshooting often requires the involvement of separate hardware and software developers.
- This presentation is an attempt to help structure the Bring-up procedure in such a way that complexity is reduced to a minimum. This should allow you to troubleshoot actual issues rather than spending time verifying your bring-up procedure.
- It is recommended that you familiarize yourself with the proposed bring-up procedure using a DA14580 or DA14581 development kit as target before you attack a prototype fresh-from-the-oven.



Agenda



Problem Definition

System Overview

First power-up

Loading test firmware

Verification

Testing the sleep clock



System Overview

What type of hardware configuration are we looking at?



To perform a proper bring-up, we will need to determine the following about the system:

- Which interfaces are available for bring-up
- Is the device Boost or Buck mode operated?
- Does the system depend on internal or external sleep clock (RCX20 vs. 32k768Hz XTAL or oscillator)?
- Will firmware be stored in OTP, external memory, or external MCU?

System Overview

Which interfaces are available for Bring-up:

The following Pins should be accessible as test pins for development and production:

Pin	Function	Development	Production
VPP	OTP Programming voltage	Required	Required unless OTP is preprogrammed
SWDIO	Debugger Data line	Required	Not required
SWCLK	Debugger Clock line	Required	Not required
VBAT1V	Boost mode supply	Required for boost mode	Required for boost mode (*1)
VBAT3V	Buck mode supply	Required for buck mode	Required for buck mode (*1)
Bootable UART(*2) RX and TX	Trimming, test, and programming interface	Required	Required
RST	Reset (active High!)	Recommended; not required(*3)	Recommended; not required(*3)
GND	Ground	Required	Required

(*1) RST must be made available if a battery is connected during test/programming

(*2) A Bootable UART pair consists of either P0_0 + P0_1, P0_2 + P0_3, P0_4 + P0_5, or P0_6 and P0_7.

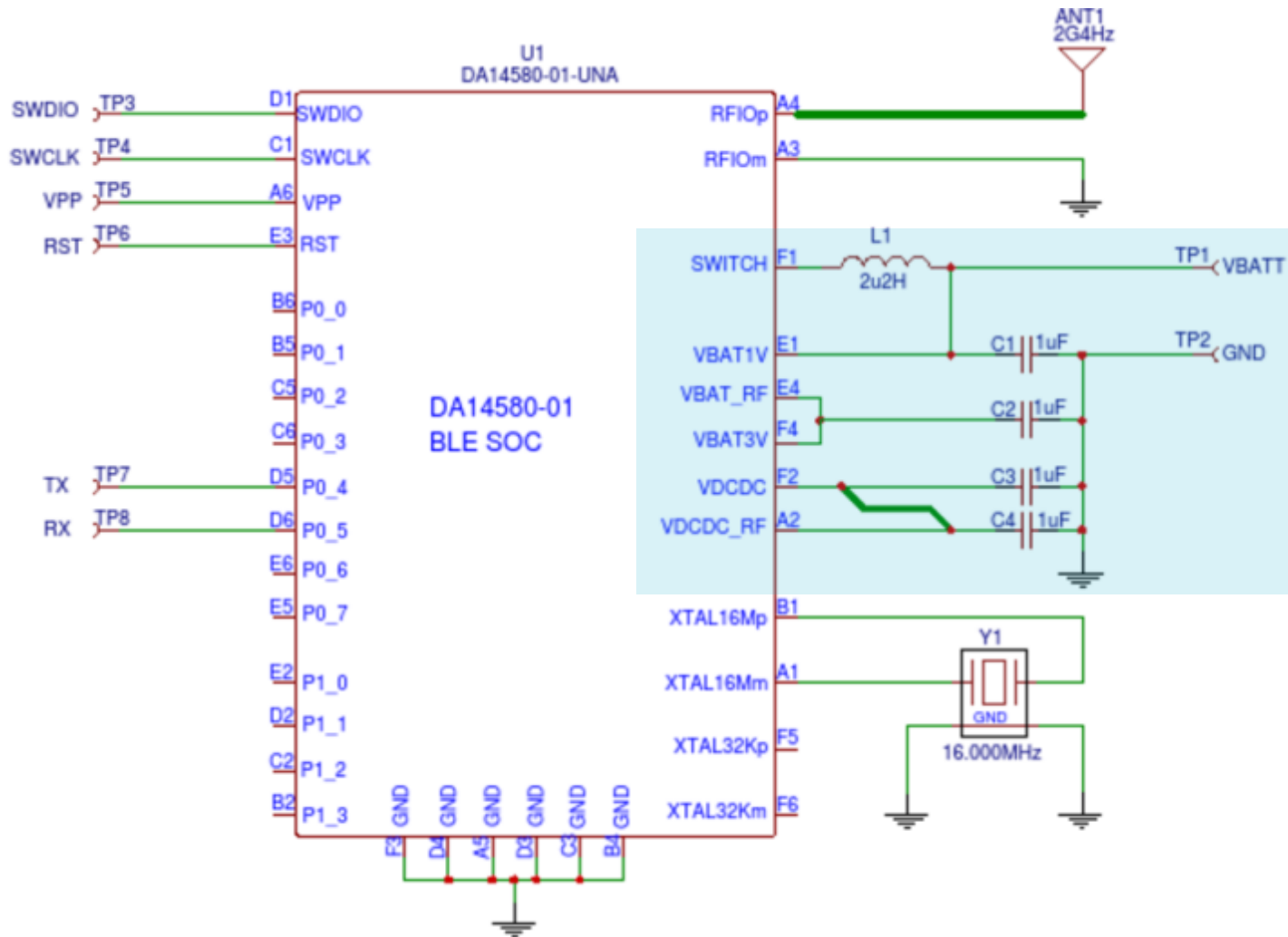
P0_4 + P0_5 is the recommended pair. See AN-B-001 for details.

(*3) Using RST during production is currently not directly supported in Dialog's production line solution, but will be in the future.



System Overview

Minimal Boost mode configuration (Supplied by 0.9V – 1.8V)



System Overview

Boost mode configuration (Supplied by 0.9V – 1.8V)

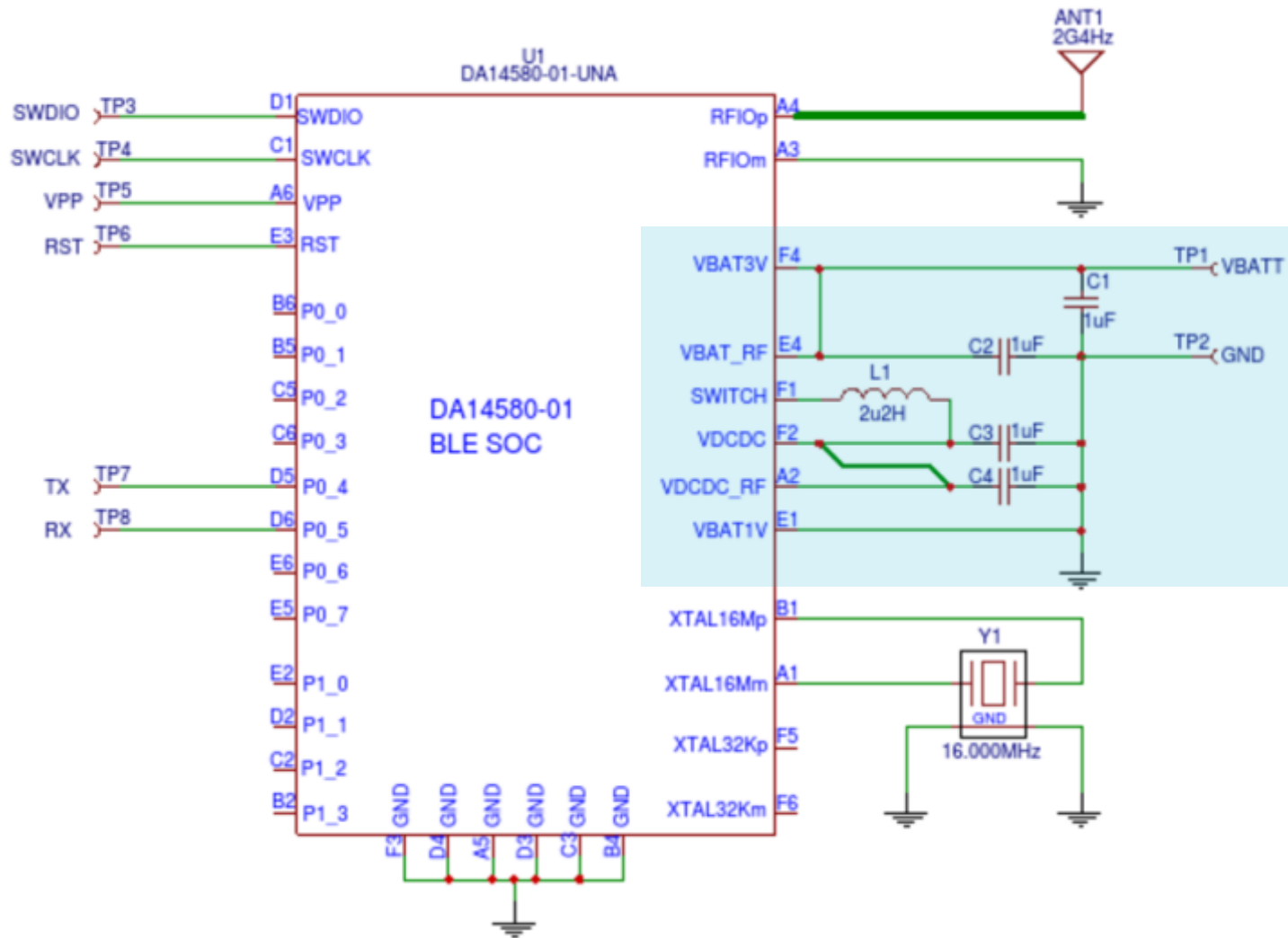
Referring to the schematic on the previous slide:

- The schematic shows the minimal implementation of a boost mode operated DA14580. The DA14581 minimal implementation is identical.
- The power source is connected to VBAT1V. The power source is typically an Alkaline, a Silver/Oxide, or a Zink/Air cell.
- The 2u2H regulator inductor is placed between VBAT1V and SWITCH
- VBAT3V and VBAT_RF are connected directly
- VDCDC and VDCDC_RF are connected via a 4-10mm Meander line and are individually decoupled by a 1uF capacitor (see AN-B-018 for details on the Meander line)
- All capacitors, C1 to C4 are placed as close to the SoC as possible
- A minimum of 4 capacitors is required
- Note that, unless you have pre-loaded firmware into OTP, that all GPIOs of port 0 will reference VBAT3V (2.8V in boost mode) while the primary boot loader runs. This means that anything connected to port 0 will be exposed to 2.8V! (See AN-B-001 for details of the primary boot loader)



System Overview

Minimal Buck mode configuration (Supplied by 2.4V – 3.3V)



System Overview

Buck mode configuration (Supplied by 2.4V – 3.3V)

Referring to the schematic on the previous slide:

- The schematic shows the minimal implementation of a buck mode operated DA14580. The DA14581 minimal implementation is identical.
- The power source is connected to VBAT3V and VBAT_RF. The power source is typically a coin cell or two alkaline cells in series.
- VBAT3V and VBAT_RF are connected directly.
- The 2u2H regulator inductor is placed between VDCDC and SWITCH.
- VDCDC and VDCDC_RF are connected via a 4-10mm Meander line and are individually decoupled by a 1uF capacitor (see AN-B-018 for details on the Meander line).
- The capacitors, C2 to C4 are placed as close to the SoC as possible. C1 is placed close to the supply source.
- A minimum of 4 capacitors is required each 1uF.
- VBAT1V is connected to GND.

Modules, based on the DA14580 (from Murata, Panasonic, TDK, and Alps) all operate in buck mode!



System Overview

Sleep Clock

- A reference clock is needed during extended and deep sleep
- This clock is referred to as sleep clock or Low-Power (LP) clock
- The LP clock source can be one of the following:
 - An external crystal*
 - An external oscillator*
 - An internal RC oscillator (RCX20)
- The system does not require a sleep clock if sleep is disabled in the firmware
- A system in boost mode configuration, using RCX20 as sleep clock, must disable sleep while in a Bluetooth connection – the timing of RCX20 in boost mode is not accurate enough to maintain a connection!

*) An external clock source can provide a fixed frequency in the range from 10kHz to 100kHz, but the current SDK5.0.4 only supports 32.768kHz!

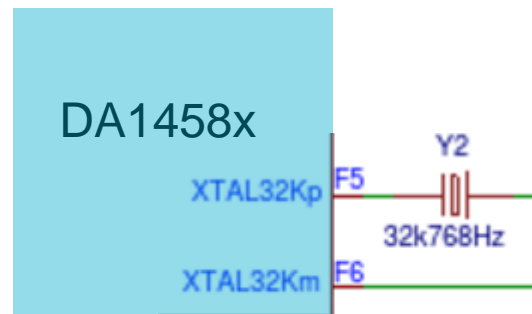


System Overview

Sleep Clock provided by external crystal

- Wired as shown below.
- No load capacitors needed unless frequency trimming is required
- 32k768kHz is the only sleep clock frequency currently supported in the SDK5.0.4
- The SDK defaults to using an external crystal as shown here:

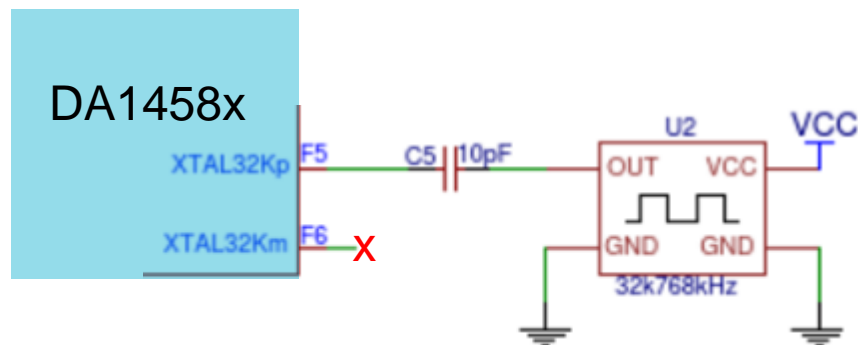
```
da1458x_config_advanced.h
37
38 /*****
39 /* Low Power clock selection.
40 /*   -LP_CLK_XTAL32      External XTAL32 oscillator
41 /*   -LP_CLK_RCX20      External internal RCX20 clock
42 /*   -LP_CLK_FROM_OTP   Use the selection in the corresponding field of OTP Header
43 /*****
44 #define CFG_LP_CLK      LP_CLK_XTAL32
45
```



System Overview

Sleep Clock provided by external oscillator or MCU

- Wired as shown below. The XTAL32Km pin must remain floating!
- The peak-peak voltage level on pin XTAL32Kp must be in the range from 0,1V to 1.5V
- C5 below serves as a signal attenuator. The internal load capacitance is in the range of 6pF to 9pF so the attenuation with a C5 value of 10pF allows for a peak-peak oscillator output of up to approximately 3V.
- The XTAL32Kp pin is internally AC coupled, which allows for the output of the external oscillator to be either sine or square waved.
- The sleep clock must be set to LP_CLK_XTAL32 as shown in previous slide (default)
- The XTAL32K_DISABLE_AMPREG bit of the CLK_32K_REG register must be set to 1 in the firmware! See next slide for details.



System Overview

Sleep Clock provided by external oscillator or MCU

The XTAL32K_DISABLE_AMPREG bit of the CLK_32K_REG register must be set to 1:

- Modify the init_pwr_and_clk_ble() function of arch_system.c
- From this:

```
arch_system.c
213
214     if ( arch_clk_is_XTAL32( ) )
215     {
216         SetBits16(CLK_32K_REG, XTAL32K_ENABLE, 1); // Enable XTAL32KHz
217
218         // Disable XTAL32 amplitude regulation in BOOST mode
219         if (GetBits16(ANA_STATUS_REG, BOOST_SELECTED) == 0x1)
220             SetBits16(CLK_32K_REG, XTAL32K_DISABLE_AMPREG, 1);
221         else
222             SetBits16(CLK_32K_REG, XTAL32K_DISABLE_AMPREG, 0);
223         SetBits16(CLK_32K_REG, XTAL32K_CUR, 5);
224         SetBits16(CLK_32K_REG, XTAL32K_RBIAS, 3);
225         SetBits16(SYS_CTRL_REG, CLK32_SOURCE, 1); // Select XTAL32K as LP clock
226
227     }
```

To this:

```
arch_system.c*
214     if ( arch_clk_is_XTAL32( ) )
215     {
216         SetBits16(CLK_32K_REG, XTAL32K_ENABLE, 1); // Enable XTAL32KHz
217
218         // Disable XTAL32 amplitude regulation in BOOST mode
219         //if (GetBits16(ANA_STATUS_REG, BOOST_SELECTED) == 0x1)
220             SetBits16(CLK_32K_REG, XTAL32K_DISABLE_AMPREG, 1);
221         //else
222             // SetBits16(CLK_32K_REG, XTAL32K_DISABLE_AMPREG, 0);
223         SetBits16(CLK_32K_REG, XTAL32K_CUR, 5);
224         SetBits16(CLK_32K_REG, XTAL32K_RBIAS, 3);
225         SetBits16(SYS_CTRL_REG, CLK32_SOURCE, 1); // Select XTAL32K as LP clock
226
227     }
```

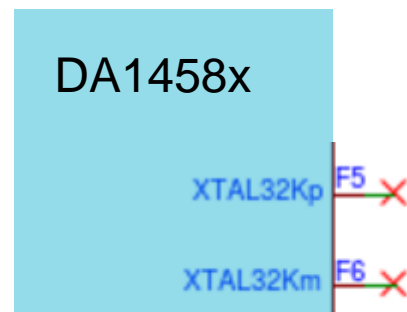


System Overview

Sleep Clock provided by internal RCX20

- Wired as shown below 😊
- RCX20 can be selected as sleep clock in the SDK as shown below:

```
da1458x_config_advanced.h
37
38 /*****
39 /* Low Power clock selection.
40 /*     -LP_CLK_XTAL32      External XTAL32 oscillator
41 /*     -LP_CLK_RCX20      External internal RCX20 clock
42 /*     -LP_CLK_FROM_OTP   Use the selection in the corresponding field of OTP Header
43 /*****
44 #define CFG_LP_CLK          LP_CLK_RCX20
```



- Using RCX20 in boost mode requires a change in the SDK – see next slide.

System Overview

Sleep Clock provided by internal RCX20

Supporting RCX20 in boost mode (sleep must be disabled during a Bluetooth connection!) requires changes in the SDK. Open the file `rwip.c` and change modify the function `rwip_sleep()`

From:

```
rwip.c
566
567 // BOOST mode + RCX is not supported
568 if (GetBits16(ANA_STATUS_REG, BOOST_SELECTED) == 1)
569     ASSERT_WARNING(0);
```

To:

```
rwip.c
566
567 // BOOST mode + RCX is not supported
568 //if (GetBits16(ANA_STATUS_REG, BOOST_SELECTED) == 1)
569 //    ASSERT_WARNING(0);
```


System Overview

Firmware location

- Firmware can be loaded on boot-up from either of the sources below:
 - From internal OTP
 - From external MCU via SPI (DA1458x is slave)
 - From external MCU via UART
 - From external memory via SPI (DA1458x is master)
 - From external memory via I²C
- The primary boot sequence is described in application note AN-B-001
- It is recommended that P0_4 and P0_5 are made available for UART communication during production test and board bring-up. Note that, if the two pins are also used as interface to an external MCU, that this MCU will need to be able to high-Z the pin connected to P0_5.



Agenda



Problem Definition

System Overview

First power-up

Loading test firmware

Verification

Testing the sleep clock



First power-up

No smoking, please!

- Please review your design as described in the previous section BEFORE you power up your board!
- It is recommended that you use a regulated power supply with overcurrent protection for this step.
- Set the voltage of your lab power supply according to the boost/buck configuration of your prototype (0.9V – 1.8V for boost mode and 2.4 – 3.3V for buck mode). Set the current limitation to just above your expected maximal current (The DA1458x draws around 5mA peak in buck mode and up to 20mA peak in boost mode, but remember to add the current consumed by the rest of your implementation).
- Monitor the voltage on VDCDC as you power up the system. You should measure about 1.41V (both in boost and Buck mode). You should have a capacitor mounted really close to the VDCDC pin that you can put your probe on.
- If the voltage on VDCDC isn't close to 1.41V, then power the system off and verify that the regulator inductor is mounted correctly and test that it has not burned open.
- **Boost mode only:** Verify that the voltage of VBAT3V is in the area of 2.8V. If this is not the case, power down the system and verify that the capacitor from VBAT3V to GND is mounted correctly and that it has not short circuited.



Agenda



Problem Definition

System Overview

First power-up

Loading test firmware

Verification

Testing the sleep clock



Loading Test Firmware

Using the `empty_peripheral_template` project from the SDK

Preparation:

- We will use the `empty_peripheral_template` project for this step. This project leaves all GPIOs in their default configuration (Input with pull-down), thus avoiding any potential hardware conflicts with other devices in your system.
- The `empty_peripheral_template` project does not use sleep mode. This allows you to run this test regardless of the sleep clock used in your hardware.
- The test firmware will allow us to observe advertising and to establish a Bluetooth connection with the prototype. The prototype will act as Bluetooth peripheral during this test (which means that it becomes the slave when the Bluetooth connection is established).
- We can use a Smart Phone or tablet to connect to the prototype. Use “Light Blue” for iOS devices or “BLE Scanner” for Android devices.
- Build the `empty_peripheral_template` for the SoC on your prototype (DA14580 or DA14581). Make sure that you are using an unmodified version of the project!



Loading Test Firmware

Selecting a hardware interface for download

- Determine if you wish to load the test software via the serial wire debugger interface or via an available bootable UART on your prototype.
- The serial wire debugger interface (SWD) is recommended, but you may not have access to it on your specific prototype due to board space constraints. In case you don't, you will have to use one of the four UART pin-pairs that are used by the primary bootloader (See AN-B-001 for details):
 - P0_0 and P0_1 at 57.6 kbit/s
 - P0_2 and P0_3 at 115.2 kbit/s
 - P0_4 and P0_5 at 57.6 kbit/s (This is the recommended UART interface)
 - P0_6 and P0_7 at 9.6 kbit/s
- Loading firmware via the serial wire debugger requires a SEGGER debugger (you can use the SEGGER debugger on your development kit, see following slide)
- Loading firmware via a bootable UART requires a RS232 level converter (you can use the FTDI converter on your development kit, see following slide)



Loading Test Firmware

Using a development kit as hardware interface

The table below shows how you can use a BASIC or PRO development kit as the download interface during bring-up. Remove all jumpers from J4 on the BASIC kit or J5 on the PRO kit, and connect to your prototype as follows:

Prototype	Development kit J4/J5	
DA1458x	SWD	UART
GND	Pin 2	Pin 2
SWCLK	Pin 27	
SWDIO	Pin 25	
TX of bootable UART		Pin 12
RX of bootable UART		Pin 14
RST (optional)	Pin 3	Pin 3

Bootable UART pin-pairs are as follows (P0_4 and P0_5 are recommended):

Boot Step	RX (Input to DA1458x)	TX (Output from DA1458x)
Boot step 3	P0_1	P0_0
Boot step 4	P0_3	P0_2
Boot step 5	P0_5	P0_4
Boot step 6	P0_7	P0_6

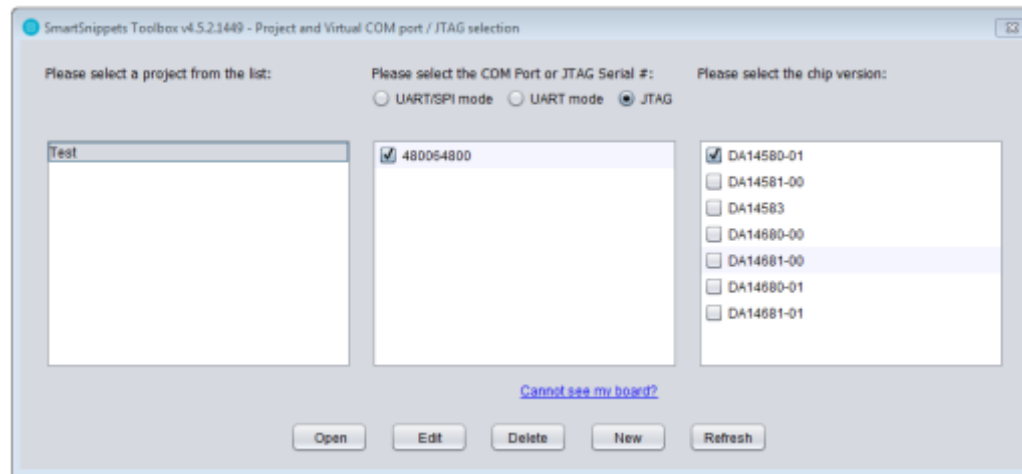
Note: You cannot use the BASIC or PRO development kit with boost mode configuration!



Loading Test Firmware

Using the 'Booter' tool in SmartSnippets toolbox with SWD

- The instructions below are for SWD interface.
- Open SmartSnippets Toolbox
- Select the JTAG option and the proper SoC and version as shown here:

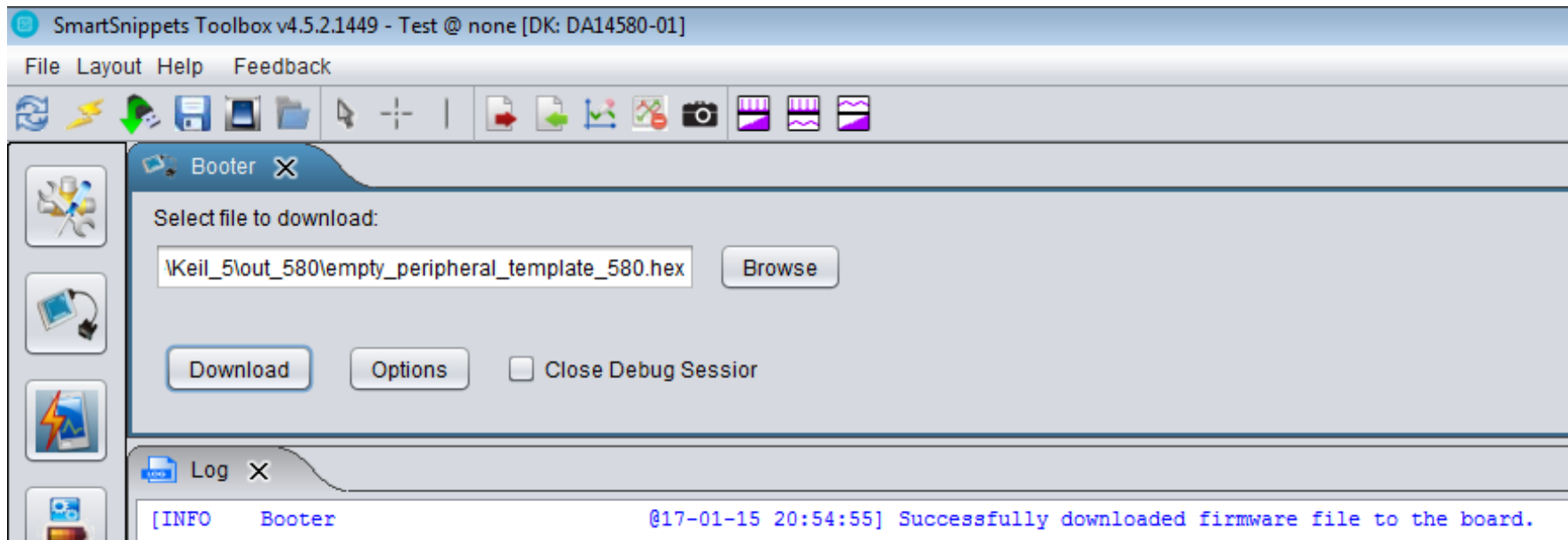


- Click the 'Booter' Icon
- Open SmartSnippets Toolbox
- Select the JTAG option and the proper SoC and version as shown here:

Loading Test Firmware

Using the 'Booter' tool in SmartSnippets toolbox with SWD

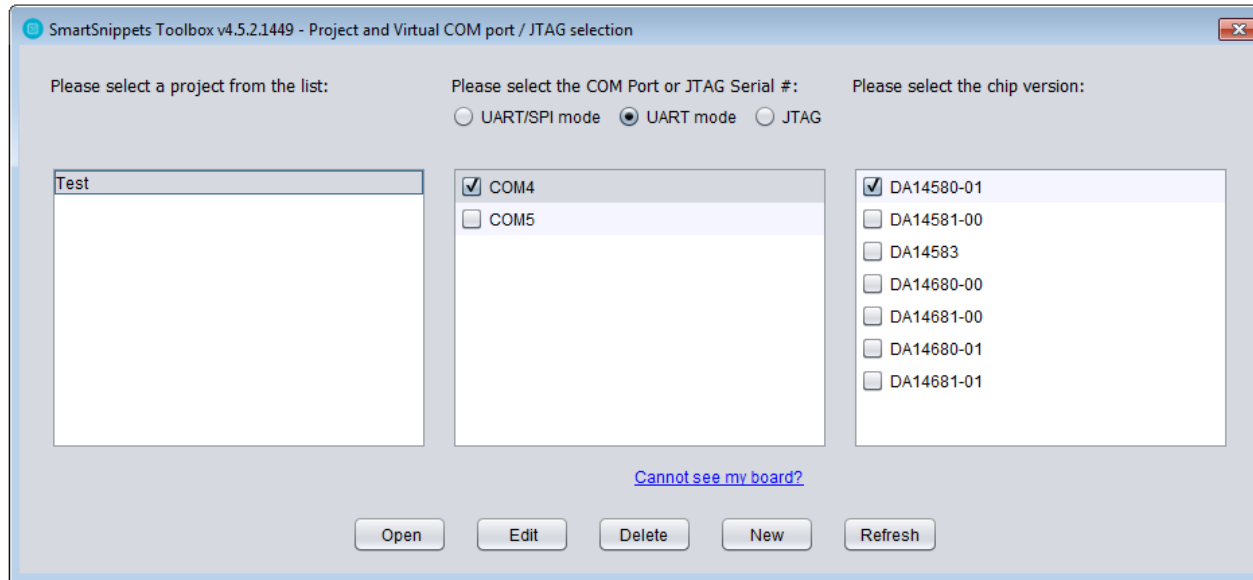
- Click the 'Booter' Icon, click 'Browse' and select the empty_peripheral_template.hex file in the SDK
- Click 'Download'
- The download should succeed as shown here:



Loading Test Firmware

Using the 'Booter' tool in SmartSnippets toolbox with UART

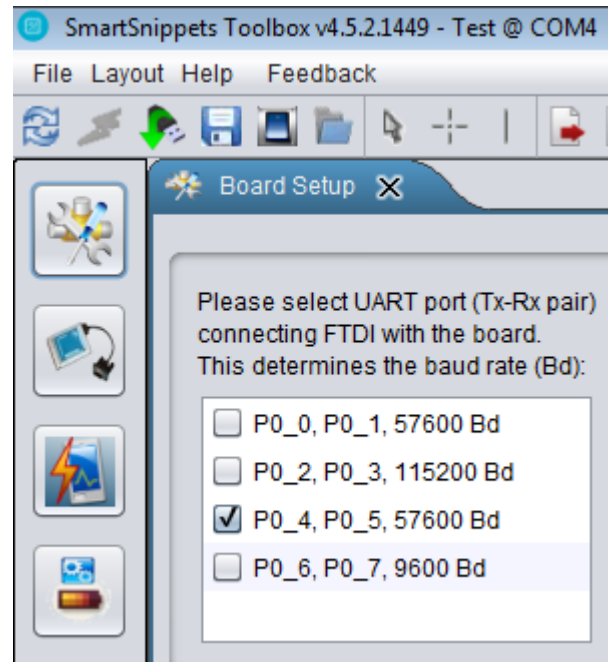
- The instructions below are for UART interface.
- Open SmartSnippets Toolbox
- Select the UART Mode option, the COM port connected to your prototype and the proper SoC and version as shown here:



Loading Test Firmware

Using the 'Booter' tool in SmartSnippets toolbox with UART

- Click the 'Board Setup' icon to the left
- Select the port-pin pair connected on your prototype (ignore all other settings)

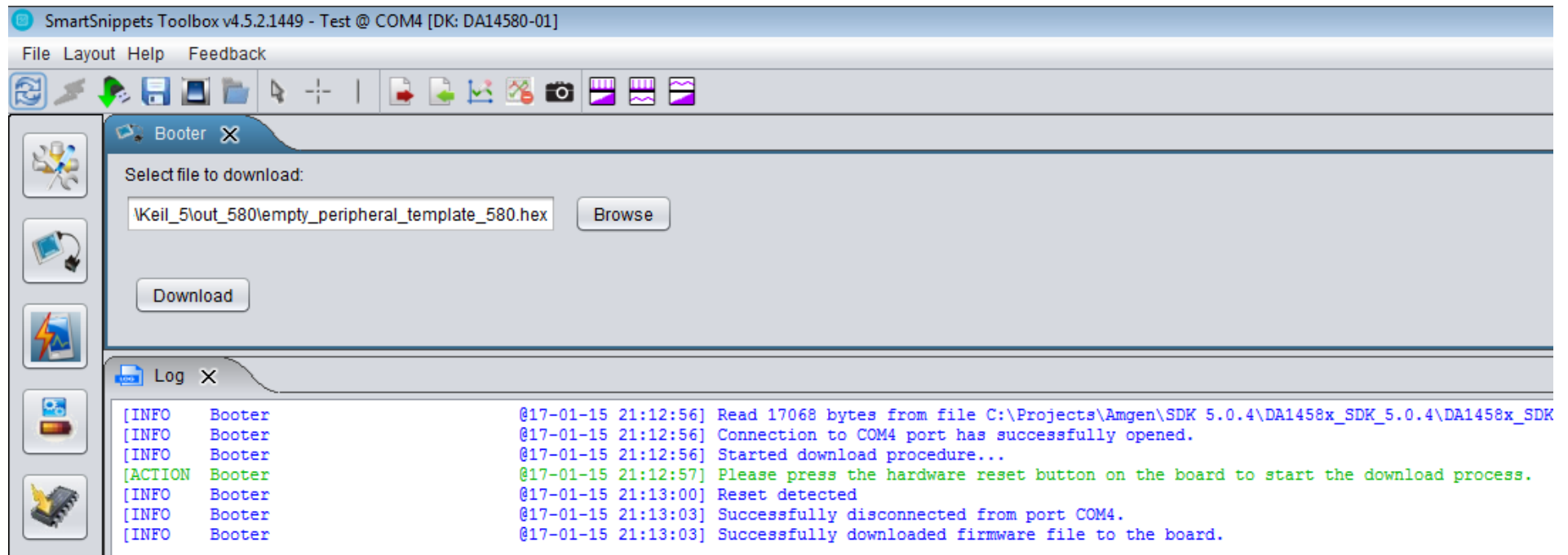


- Close the Board Setup window

Loading Test Firmware

Using the 'Booter' tool in SmartSnippets toolbox with UART

- Click the 'Booter' Icon, click 'Browse' and select the empty_peripheral_template.hex file in the SDK
- Click 'Download' (Reset or power cycle your prototype if requested to do so)
- The download should succeed as shown here (note the reset request in the log):



The screenshot displays the SmartSnippets Toolbox v4.5.2.1449 interface. The main window is titled 'Booter' and shows a 'Select file to download:' dialog box with the file path 'Keil_5\out_580\empty_peripheral_template_580.hex' and a 'Browse' button. Below the dialog is a 'Download' button. The bottom panel shows a 'Log' window with the following text:

```
[INFO   Booter   @17-01-15 21:12:56] Read 17068 bytes from file C:\Projects\Amgen\SDK 5.0.4\DA1458x_SDK_5.0.4\DA1458x_SDK
[INFO   Booter   @17-01-15 21:12:56] Connection to COM4 port has successfully opened.
[INFO   Booter   @17-01-15 21:12:56] Started download procedure...
[ACTION Booter   @17-01-15 21:12:57] Please press the hardware reset button on the board to start the download process.
[INFO   Booter   @17-01-15 21:13:00] Reset detected
[INFO   Booter   @17-01-15 21:13:03] Successfully disconnected from port COM4.
[INFO   Booter   @17-01-15 21:13:03] Successfully downloaded firmware file to the board.
```

Agenda



Problem Definition

System Overview

First power-up

Loading test firmware

Verification

Testing the sleep clock



Verification

Does the prototype come up?

- After successfully loading the test firmware it is time to see the prototype in action
- It is recommended that you use a smart phone or tablet running a Bluetooth smart app (“Light Blue” for iOS or “BLE Scanner” for Android)
- Start the app and initiate a Bluetooth Smart Scan.
- You should see a device named “DIALOG-TMPL’ in the list of discovered devices. If you do not see the ‘DIALOG-TMPL’ name in the list, you can try to turn off and on Bluetooth on your phone or tablet and try again. Try to hold your smartphone a few inches away from your prototype. If you still do not see your prototype in a scan, try to repeat the download procedure described earlier. If you still do not see your device, something is wrong with your antenna or the SoC’s connection to the antenna or the 16MHz crystal is not oscillating. This can be due to bad soldering, so you may want to try another board.
- Use the app to establish a connection with your prototype. You should see the device expose only the two mandatory services (Generic Access Service and Generic Attribute Service)
- If you are not anticipating the use of sleep modes, you can skip the next section and start testing your prototype on a system level.



Agenda



Problem Definition

System Overview

First power-up

Loading test firmware

Verification

Testing the sleep clock



Testing the sleep clock

Putting the prototype to sleep and your mind to rest

- The final thing to verify is the sleep clock. By now you should already know how to modify the `empty_peripheral_template` project in order to support your sleep clock source (see the earlier section, System overview)
- Also, in the `empty_peripheral_template` project, enable extended sleep as shown here:

```
user_config.h
37  * Default sleep mode. Possible values are:
38  *
39  * - ARCH_SLEEP_OFF
40  * - ARCH_EXT_SLEEP_ON
41  * - ARCH_DEEP_SLEEP_ON
42  *
43  *****
44  */
45  const static sleep_state_t app_default_sleep_mode = ARCH_EXT_SLEEP_ON;
46
```

- Rebuild the project and repeat the Verification section. You should again see the prototype advertise with the name “DIALOG-TMPL” and you should be able to establish a connection just as we did earlier. If you cannot see the prototype in a scan or if you cannot connect to the prototype, then there is a problem with your sleep clock. Check with a high impedance scope probe that the XTAL32Kp pin is indeed oscillating (only if you are using an external crystal or oscillator).



Thank You !!!
Q&A

Powering the Smart Connected Future

www.dialog-semiconductor.com



... personal
... portable
... connected

