Agenda

Problem Definition

System Overview

First power-up

Loading test firmware

Verification

Testing the sleep clock
• 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
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:

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

(*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.
Minimal 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)
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!
• 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:

```c
#define CFG_LP_CLK    LP_CLK_XTAL32
```

[Diagram of DA1458x with external crystal]
• 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_AMPRG bit of the CLK_32K_REG register must be set to 1 in the firmware! See next slide for details.
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:

```
    if ( arch_clk_is_XTAL32() )
        {
            SetBits16(CLK_32K_REG, XTAL32K_ENABLE, 1); // Enable XTAL32KHz
            // Disable XTAL32 amplitude regulation in BOOST mode
            if (GetBits16(ANA_STATUS_REG, BOOST_SELECTED) == 0x1)
                SetBits16(CLK_32K_REG, XTAL32K_DISABLE_AMPREG, 1);
            else
                SetBits16(CLK_32K_REG, XTAL32K_DISABLE_AMPREG, 0);
            SetBits16(CLK_32K_REG, XTAL32K_CUR, 5);
            SetBits16(CLK_32K_REG, XTAL32K_RBIAS, 3);
            SetBits16(SYS_CTRL_REG, CLK32_SOURCE, 1); // Select XTAL32K as LP clock
        }
```

To this:

```
    if ( arch_clk_is_XTAL32() )
        {
            SetBits16(CLK_32K_REG, XTAL32K_ENABLE, 1); // Enable XTAL32KHz
            // Disable XTAL32 amplitude regulation in BOOST mode
            if (GetBits16(ANA_STATUS_REG, BOOST_SELECTED) == 0x1)
            {
                SetBits16(CLK_32K_REG, XTAL32K_DISABLE_AMPREG, 1);
            }
            else
            {
                SetBits16(CLK_32K_REG, XTAL32K_DISABLE_AMPREG, 0);
            }
            SetBits16(CLK_32K_REG, XTAL32K_CUR, 5);
            SetBits16(CLK_32K_REG, XTAL32K_RBIAS, 3);
            SetBits16(SYS_CTRL_REG, CLK32_SOURCE, 1); // Select XTAL32K as LP clock
        }
```
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:

    ```
    // Low Power clock selection.
    /*
    - LP_CLK_XTAL32 External XTAL32 oscillator
    - LP_CLK_RCX20 External internal RCX20 clock
    - LP_CLK_FROM_OTF Use the selection in the corresponding field of OTP Header
    */
    #define CFG_LP_CLK    LP_CLK_RCX20
    ```

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

```c
// BOOST mode + RCX is not supported
if (GetBits16(ANA_STATUS_REG, BOOST_SELECTED) == 1)
    ASSERT_WARNING(0);
```

To:

```c
// BOOST mode + RCX is not supported
//if (GetBits16(ANA_STATUS_REG, BOOST_SELECTED) == 1)
//    ASSERT_WARNING(0);
```
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.
<table>
<thead>
<tr>
<th>Agenda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Definition</td>
</tr>
<tr>
<td>System Overview</td>
</tr>
<tr>
<td>First power-up</td>
</tr>
<tr>
<td>Loading test firmware</td>
</tr>
<tr>
<td>Verification</td>
</tr>
<tr>
<td>Testing the sleep clock</td>
</tr>
</tbody>
</table>
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!
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).
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:

<table>
<thead>
<tr>
<th>Prototype</th>
<th>Development kit J4/J5</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA1458x</td>
<td>SWD</td>
</tr>
<tr>
<td>GND</td>
<td>Pin 2</td>
</tr>
<tr>
<td>SWCLK</td>
<td>Pin 27</td>
</tr>
<tr>
<td>SWDIO</td>
<td>Pin 25</td>
</tr>
<tr>
<td>TX of bootable UART</td>
<td>Pin 12</td>
</tr>
<tr>
<td>RX of bootable UART</td>
<td>Pin 14</td>
</tr>
<tr>
<td>RST (optional)</td>
<td>Pin 3</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Boot Step</th>
<th>RX (Input to DA1458x)</th>
<th>TX (Output from DA1458x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot step 3</td>
<td>P0_1</td>
<td>P0_0</td>
</tr>
<tr>
<td>Boot step 4</td>
<td>P0_3</td>
<td>P0_2</td>
</tr>
<tr>
<td>Boot step 5</td>
<td>P0_5</td>
<td>P0_4</td>
</tr>
<tr>
<td>Boot step 6</td>
<td>P0_7</td>
<td>P0_6</td>
</tr>
</tbody>
</table>

Note: You cannot use the BASIC or PRO development kit with boost mode configuration!
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 ‘Booster’ tool in SmartSnippets toolbox with SWD

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

![Image of SmartSnippets Toolbox with 'Booster' tool and successful download message]

[INFO  Booter  @17-01-15 20:54:55] Successfully downloaded firmware file to the board.
Loading Test Firmware

Using the ‘Booer’ 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 ‘Booster’ 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 ‘Booster’ tool in SmartSnippets toolbox with UART

• Click the ‘Booster’ 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):

Image of download process
Problem Definition
System Overview
First power-up
Loading test firmware
Verification
Testing the sleep clock
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:

```
* Default sleep mode. Possible values are:
*
* - ARCH_SLEEP_OFF
* - ARCH_EXT_SLEEP_ON
* - ARCH_DEEP_SLEEP_ON
*
*******************************************************************************
const static sleep_state_t app_default_sleep_mode = ARCH_EXT_SLEEP_ON;
```

• 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