UG404: WF(M)200 Configuration Guide with PDS

This document provides information to help users configure WF200 and WFM200 hardware and application settings through Platform Data Set (PDS) files.

KEY FEATURES

- Easy configuration to match hardware and application requirements.
- Specific section to perform RF TX and RX tests.

Table of Contents

1. Introduction

The present document details WF200 and WFM200 configuration via the Platform Data Set (PDS), a set of parameter values required to configure the firmware to match both hardware and application requirements. The PDS file includes the following sections:

- PROG_PINS_CFG controls device programmable input/output pins when device is active or sleeping
- HIF_PINS_CFG configures bus interface with host MCU
- HF_CLK sets capacitive load applied to 38.4MHz high frequency crystal oscillator
- FEM_CFG controls external FEM
- RF_POWER_CFG configures RX loss, TX loss, and TX output power according to RF channel, modulation, and code rate used
- RF_ANTENNA_SEL_DIV_CFG sets TX and RX RF ports and enable antenna diversity if applicable

In addition, PDS features a specific section, TEST_FEATURE_CFG, to configure device in test mode and perform RF TX and RX tests.

The first part of this document describes the flow and the second part describes the various parameters. The third section is dedicated to the RF test feature mode.

Note that this guide, in order to stay up to date, contains Github links pointing to branches, hence file content can change over time. However, for development it is recommended to reference git files by tags to be sure content does not move without explicit action.

2. PDS General Information

The PDS configuration file uses a JSON format which facilitates editing and includes comments. A Python3 tool (pds_compress) is used to compress this file into a PDS data file before it is sent to WF200 firmware.

2.1 Terminology and File Extensions

- A PDS 'node' is a logical branch or sub-branch in the PDS hierarchy. A node starts with its node name, followed by ':' and contains (optional) sub-nodes and attributes, surrounded by '{'and'}'.
- A PDS 'attribute' is the end of a PDS hierarchy branch and corresponds to a firmware parameter. An attribute starts with its attribute name, followed by ':' and its attribute value.
- A comma ',' is used to separate nodes or attributes.
- A PDS 'section' is a node or a group of nodes and their attributes, not always starting at the root of the hierarchy. When a PDS section starts at the root of the hierarchy it can be sent independently to the firmware. In the example of section RF_ANTENNA_SEL_DIV_CFG below, section is highlighted in blue, attributes are highlighted in red and values are highlighted in green.

Table 2.1. PDS Section RF_ANTENNA_SEL_DIV_CFG

```
/********************/
/* RF configuration */
/********************/

RF_ANTENNA_SEL_DIV_CFG: {
    //
    // Antenna selection: allows to select the RF port used. Can be different
for Tx and Rx (FEM case)
    //   - TX1_RX1: RF_1 used (default)
    //   - TX2_RX2: RF_2 used
    //   - TX1_RX2: Tx on RF_1 and Rx on RF_2
    //   - TX2_RX1: Tx on RF_2 and Rx on RF_1
    //   - TX12_RX12: antenna diversity case: both Tx and Rx on the same RF
port which is automatically selected (requires DIVERSITY to be set)
    //
    RF_PORTS: TX1_RX1,

    //
    // Diversity control mode:
    //   - OFF (default)
    //   - INTERNAL   :  requires RF_PORTS to be set to TX12_RX12
    //
    DIVERSITY: OFF
},
```

- PDS definitions file https://github.com/SiliconLabs/wfx-firmware/blob/master/PDS/definitions.in
  - lists all node references as a series of letters with a defined hierarchy
  - lists all possible attribute values as a series of integer values
  - is used to generate PDS data file that will be sent to firmware
  - uses a '.in' extension

PDS definitions file is closely related to firmware release, since the node and attribute codes need to match the firmware, and all values need to fall within the expected range.

In the example of section RF_ANTENNA_SEL_DIV_CFG, names of section and attributes are highlighted in blue and their corresponding letters are highlighted in red while attribute values are highlighted in green and their corresponding integer values in orange.

Table 2.2. Definitions Associated to PDS Section RF_ANTENNA_SEL_DIV_CFG

```
#define RF_ANTENNA_SEL_DIV_CFG j
#define
RF_PORTS                a
#define
DIVERSITY               b
// RF_ANTENNA_SEL_DIV_CFG.RF_PORTS
#define TX1_RX1   0
#define TX2_RX2   1
#define TX2_RX1   2
#define TX1_RX2   3
#define TX12_RX12 4
// RF_ANTENNA_SEL_DIV_CFG.DIVERSITY
#define OFF      0
#define INTERNAL 1
```

PDS Configuration Files

- include PDS definitions file
- contain the settings required by the hardware and the application used
- contain comments to document the use of each section as well as the possible options
- are easy to read and modify

The latest PDS files for WF200 and WFM200 dev kits can be found in Github repository https://github.com/SiliconLabs/wfx-pds:

- BRD8022A_Rev_A06.pds.in
- BRD8023A_Rev_B01.pds.in

The PDS template (https://github.com/SiliconLabs/wfx-firmware/blob/master/PDS/template.pds.in) lists all possible PDS sections and parameters. When creating a custom PDS file, the user should

- copy one of the existing .pds.in files in a custom folder
- update link to PDS definitions file
- remove unused sections and add required sections from the template
- update value of attributes according to hardware and application requirements

To reduce the amount of PDS data sent to firmware, PDS configuration file is compressed to a PDS data file containing a minimal set of bytes, enough to convey the necessary configuration information.

PDS Data Files

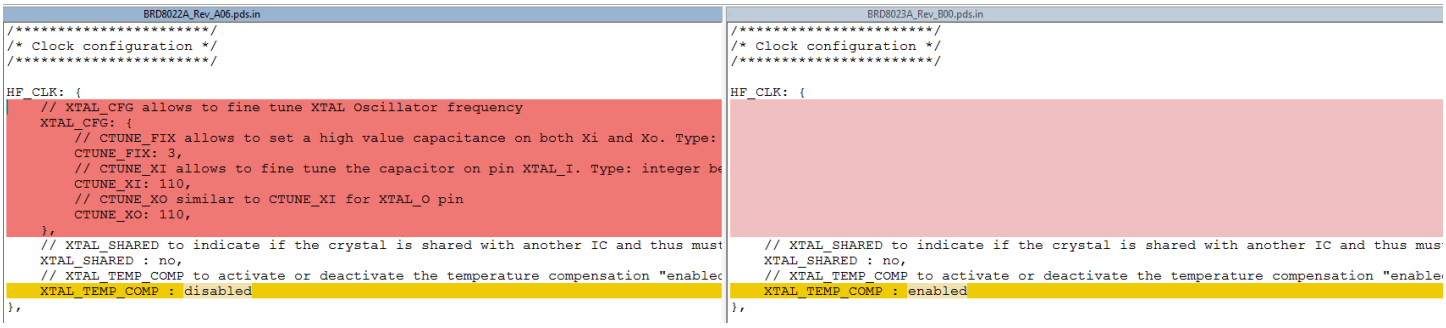
- are the result of compressing the PDS files resulting in
  - a single line string for Linux applications
  - a one-line-per-section table of strings for RTOS applications
- use
  - a '.pds' extension for Linux applications
  - a '.h' extension for RTOS applications

For compactness, numbers in PDS data files are in uppercase hexadecimal (without any prefix).

In the example of section RF_ANTENNA_SEL_DIV_CFG, it becomes after compression:

- string `j:{a:0,b:0}` in file wf200.pds for Linux applications
- line "`{j:{a:0,b:0}}`" of table wf200_pds[ ] for RTOS applications

The PDS data file being not easily readable, it is always preferable to change value of PDS attributes in the PDS configuration file, and then go through the PDS compression. When looking for differences between PDS files, a comparison of PDS data files is most likely going to be unusable. It is highly recommended to compare PDS configuration files instead as shown in example below which highlights differences between WF200 BRD8022A_Rev_A06.pds.in and WFM200 BRD8023A_Rev_B00.pds.in, WFM200 integrating a 38.4MHz temperature compensated crystal oscillator whereas WF200 does not.



BRD8022A_Rev_A06.pds.in

```
/***********************/
/* Clock configuration */
/***********************/

HF_CLK: {
    // XTAL_CFG allows to fine tune XTAL Oscillator frequency
    XTAL_CFG: {
        // CTUNE_FIX allows to set a high value capacitance on both Xi and Xo. Type:
        CTUNE_FIX: 3,
        // CTUNE_XI allows to fine tune the capacitor on pin XTAL_I. Type: integer be
        CTUNE_XI: 110,
        // CTUNE_XO similar to CTUNE_XI for XTAL_O pin
        CTUNE_XO: 110,
    },
    // XTAL_SHARED to indicate if the crystal is shared with another IC and thus must
    XTAL_SHARED : no,
    // XTAL_TEMP_COMP to activate or deactivate the temperature compensation "enablec
    XTAL_TEMP_COMP : disabled
},
```

BRD8023A_Rev_B00.pds.in

```
/***********************/
/* Clock configuration */
/***********************/

HF_CLK: {

    // XTAL_SHARED to indicate if the crystal is shared with another IC and thus mus
    XTAL_SHARED : no,
    // XTAL_TEMP_COMP to activate or deactivate the temperature compensation "enable
    XTAL_TEMP_COMP : enabled
},
```

Figure 2.1. Example of Differences between WF200 BRD8022A and WFM200 BRD8023A PDS Configuration Files

2.2 PDS Flow

2.2.1 PDS Data File Generation Flow

PDS configuration file is compressed into PDS data file using Python3 pds_compress tool (https://github.com/SiliconLabs/wfx-linux-tools/blob/master/pds_compress). PDS output format depends on the application:

- .pds file, for Linux application where it is read at runtime by the wfx Linux driver
- .h file, for RTOS application where it is included at compilation in the wfx driver code

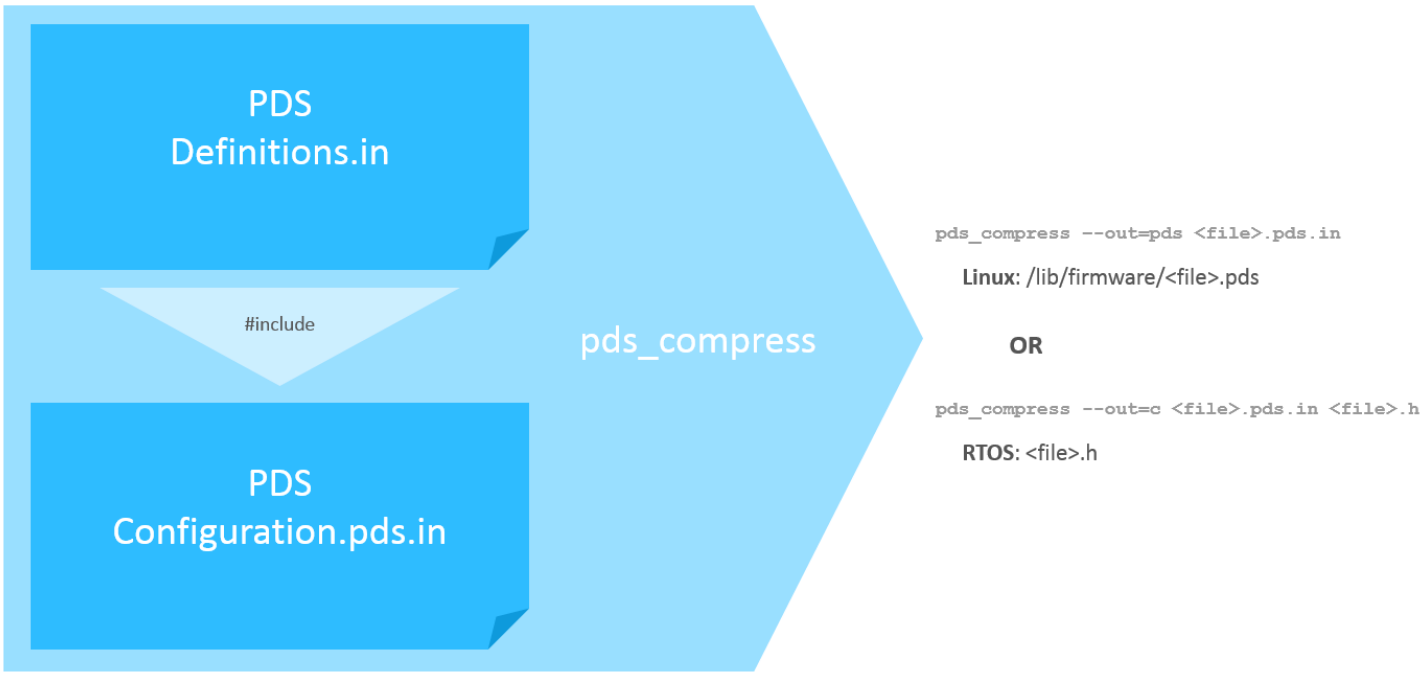


Figure 2.2. Generation Flow of PDS Data File

2.2.1.1 *pds_compress*

pds_compress tool is written in Python3, for ease of writing and maintenance. To get more information, use `pds_compress --help`.

Table 2.3. Help on pds_compress

```
usage: pds_compress [options] INPUT [OUTPUT]
Generate a compressed version of PDS from a full PDS
positional arguments:
  INPUT                 input file (except C format, all output formats can
be used as input)
  OUTPUT                output file (standard output if not specified)
optional arguments:
  -h, --help            show this help message and exit
  -I DIR, --include DIR
                        search includes in this subdirectory
  -D DEF[=VAL], --define DEF[=VAL]
                        predefine DEF with value VAL
  -f, --force           try to produce (probably broken) output even if errors
                        are detected
  --out {pds,tinypds,c,json}
                        specify output format. Accepted values: pds
                        (compressed PDS), tinypds (indented PDS), c (C file),
                        json. Default: pds
  -j                    shortcut for --out=json
  -c                    shortcut for --out=c
  -p                    shortcut for --out=pds
  -t                    shortcut for --out=tinypds
```

For Linux applications only, `pds_compress –-out=tinypds` can be used to look for differences between .pds data files in case associated PDS configuration files are no more available. Option tinypds allows a side-by-side/line-by-line comparison. However, the tinypds PDS format is not understood by WF200 firmware.

2.2.1.2 *Linux Application*

The Linux PDS data file is obtained using the ‘pds’ output format: `pds_compress --out=pds <file>.pds.in <file>.pds`.

In the example of WF200 development kit, BRD8022A_Rev_A06 PDS data file for PDS API version 3.0 contains one-line string:

Table 2.4. Content of PDS Data File for WF200 BRD8022A_Rev_A06 Dev Kit for Linux


```
{a:{a:3,b:0},b:{a:{a:4,b:0,c:0,d:0,e:A},b:{a:4,b:0,c:0,d:0,e:B},c:{a:4,b:0,c:0,d:0,e:C},d:{a:4,b:0,c:0,d:
0,e:D},e:{a:4,b:0,c:0,d:0,e:E},f:{a:4,b:0,c:0,d:0,e:F},g:{a:4,b:0,c:0,d:0,e:G},h:{a:4,b:0,c:0,d:0,e:H},i:{a:
4,b:0,c:0,d:0,e:I},j:{a:4,b:0,c:0,d:0,e:J},k:{a:4,b:0,c:0,d:0,e:K},l:{a:4,b:0,c:0,d:1,e:L},m:{a:4,b:0,c:0,d:
1,e:M}},c:{a:{a:6,b:0,c:0},b:{a:6,b:0,c:0},c:{a:6,b:0,c:1},d:{a:6,b:0,c:0},e:{a:6,b:0,c:0},f:{a:6,b:0,c:
0}},e:{a:{a:3,b:6E,c:6E},b:0,c:0},h:{e:0,a:50,b:0,c:[{a:1,b:[0,0,0,0,0,0]},{a:[2,3],b:[0,0,0,0,0,0]},{a:
[4,9],b:[0,0,0,0,0,0]},{a:[A,C],b:[0,0,0,0,0,0]},{a:D,b:[0,0,0,0,0,0]},{a:E,b:[0,0,0,0,0,0]}],d:0},j:{a:0,b:
0}}
```

2.2.1.3 *RTOS Application*

The RTOS PDS data file is obtained using the ‘c’ output format and is generally stored into a .h file included at compilation:

```
pds_compress --out=c <file>.pds.in <file>.h
```

In the example of WF200 development kit, BRD8022A_Rev_A06 PDS header file for PDS API version 3.0 contains ‘wf200_pds’ table of strings below:

Table 2.5. Content of PDS Header File for WF200 BRD8022A_Rev_A06 Dev Kit for RTOS

```
#ifndef WF200_PDS_H
#define WF200_PDS_H

static const char* const wf200_pds[] = {
"{a:{a:3,b:0}}",
"{b:{a:{a:4,b:0,c:0,d:0,e:A},b:{a:4,b:0,c:0,d:0,e:B},c:{a:4,b:0,c:0,d:0,e:C},d:{a:4,b:0,c:0,d:0,e:D},e:{a:
4,b:0,c:0,d:0,e:E},f:{a:4,b:0,c:0,d:0,e:F},g:{a:4,b:0,c:0,d:0,e:G},h:{a:4,b:0,c:0,d:0,e:H},i:{a:4,b:0,c:0,d:
0,e:I},j:{a:4,b:0,c:0,d:0,e:J},k:{a:4,b:0,c:0,d:0,e:K},l:{a:4,b:0,c:0,d:1,e:L},m:{a:4,b:0,c:0,d:1,e:M}}}",
"{c:{a:{a:6,b:0,c:0},b:{a:6,b:0,c:0},c:{a:6,b:0,c:1},d:{a:6,b:0,c:0},e:{a:6,b:0,c:0},f:{a:6,b:0,c:0}}}",
"{e:{a:{a:3,b:6E,c:6E},b:0,c:0}}",
"{h:{e:0,a:50,b:0,c:[{a:1,b:[0,0,0,0,0,0]},{a:[2,3],b:[0,0,0,0,0,0]},{a:[4,9],b:[0,0,0,0,0,0]},{a:[A,C],b:
[0,0,0,0,0,0]},{a:D,b:[0,0,0,0,0,0]},{a:E,b:[0,0,0,0,0,0]}],d:0}}",
"{j:{a:0,b:0}}",
};

#endif
```

2.2.2 PDS Data File Transmission Flow

The content of PDS data file is processed by the firmware, not by the driver. The driver role only consists in sending the PDS data file to the WF200 firmware. Note that WF200 and WFM200 use the same firmware. Neither the Lower MAC Linux driver nor the full MAC driver have control over the content of the PDS data file which is sent ‘as is’, without any checking. The user shall refer to PDS release note https://github.com/SiliconLabs/wfx-firmware/blob/master/PDS/CHANGES.md to check compatibility between firmware and PDS.

2.2.2.1 *Download at Firmware Startup*

Initial PDS data file has to be sent to the WF200 firmware right after the firmware has been downloaded and started, to configure the firmware according to hardware and application requirements. In most cases, only this initial PDS data file is sent to the WF200 firmware.

- Linux
  - Initial PDS data is stored in /lib/firmware/wf200.pds.
  - PDS data file /lib/firmware/wf200.pds is sent to the firmware by the driver using wfx function wfx_send_pdata_pds().
- RTOS
  - Initial PDS data is stored in table wf200_pds[ ].
  - The content of table wf200_pds[ ] is sent to the firmware using functions detailed in Chapter 5 of document https://docs.silabs.com/wifi/wf200/rtos/latest/driver-initialization.

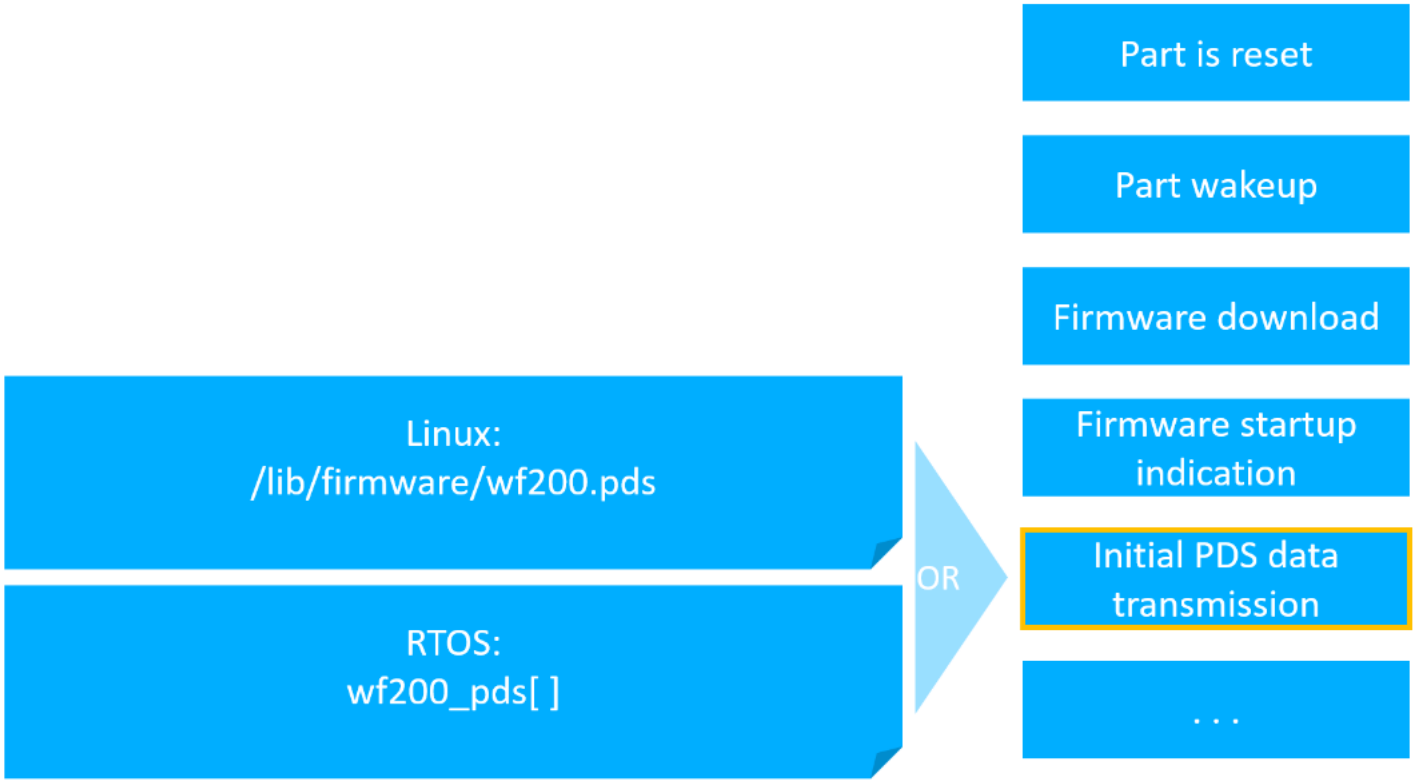


Figure 2.3. Transmission Flow of PDS Data File at WF200 Firmware Startup

2.2.2.2 *Download while Application is Running*

Runtime PDS data can be sent during execution, at any moment, to update any section of the PDS.

- Linux
  - Runtime PDS data file is sent to the wfx driver using a write-only file pointer at `/sys/kernel/debug/ieee80211/phy*/wfx/send_pds`, phy* usually being phy0
  - As soon as there is readable content in `/sys/kernel/debug/ieee80211/phy0/wfx/send_pds`, the wfx driver forwards it to the WF200 firmware.
- RTOS
  - Runtime PDS data is sent to the WF200 firmware using sl_ function sl_wfx_send_configuration().

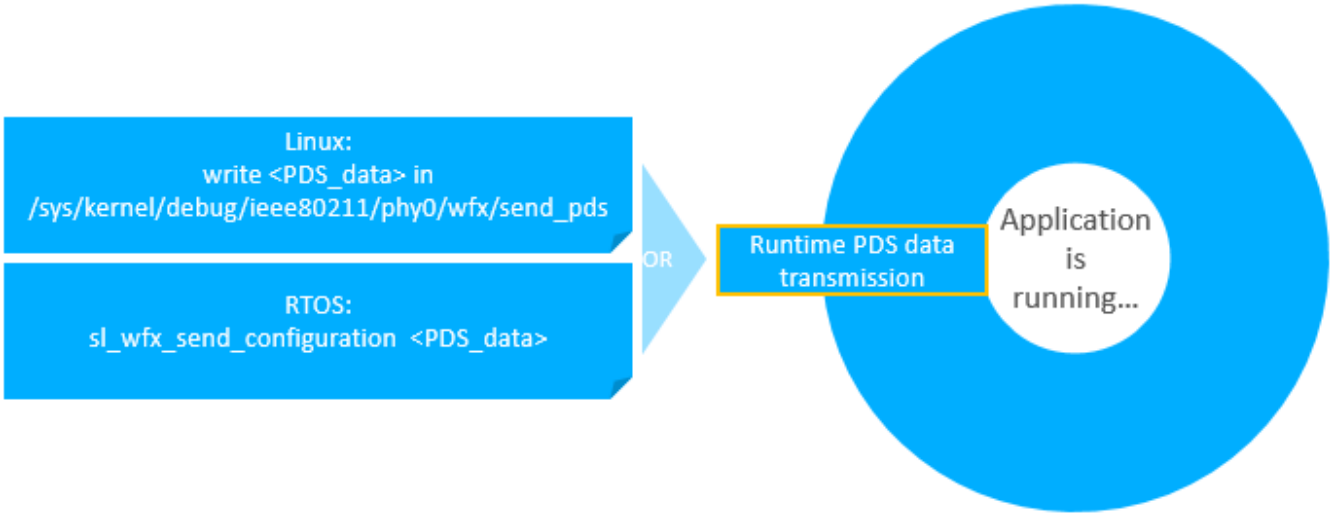


Figure 2.4. Transmission Flow of PDS Data File while Application is Running

2.2.2.3 *Partial PDS Transmission*

When transmitting runtime PDS data, the user will most likely want to transmit only a limited portion of a PDS file. To achieve this, one method consists of editing the PDS configuration file and commenting out unchanged sections. Another method consists of removing most of the unwanted content, to focus on sections that need to be sent.

To be properly interpreted by WF200 firmware, the entire section 'from root to values' must be sent, such that the firmware can properly decode the PDS hierarchy and update WF200/WFM200 configuration accordingly.

To comment a PDS section, block commenting should be used:

- Add '`/*`' before the section name
- Add '`*/`' after the closing '}' or after the closing '},'

For example, section HF_CLK:

```
HF_CLK: {
    // XTAL_CFG allows to. . .
    XTAL_CFG: {
        CTUNE_FIX: 3,
        CTUNE_XI: 110,
        CTUNE_XO: 110,
    },

    XTAL_SHARED : no,
    XTAL_TEMP_COMP : disabled
},
```

becomes this, after commenting the XTAL_CFG section:

```
HF_CLK: {
    // XTAL_CFG allows to. . .
/*    XTAL_CFG: {
        CTUNE_FIX: 3,
        CTUNE_XI: 110,
        CTUNE_XO: 110,
    },
*/
    XTAL_SHARED : no,
    XTAL_TEMP_COMP : disabled
},
```

3. PDS Detailed Settings

As shown in the PDS template https://github.com/SiliconLabs/wfx-firmware/blob/master/PDS/template.pds.in, the PDS configuration file is split in several sections:

- HEADER
- PROG_PINS_CFG
- HIF_PINS_CFG
- HF_CLK
- FEM_CFG
- RF_POWER_CFG
- RF_ANTENNA_SEL_DIV_CFG
- TEST_FEATURE_CFG

TEST_FEATURE_CFG has a specific behavior and is described in next section.

3.1 Inclusion of PDS Definitions File

The first active (i.e. non-empty and not a comment) line in the PDS configuration file includes the PDS definitions file. This is like copying the entire PDS definitions file instead of the inclusion line, allowing a single definitions file to be used for multiple hardware configurations.

Table 3.1. Inclusion of PDS Definitions File

```
#include "definitions.in”
```

3.2 HEADER Section

HEADER section is quite simple and only contains information about PDS API format in order to ensure that the PDS format fits the WF200 firmware requirements. This section is not used by the firmware, so there is no compatibility verification. The user shall refer to PDS release note https://github.com/SiliconLabs/wfx-firmware/blob/master/PDS/CHANGES.md to check compatibility. However, note that in cases where some sections are not downloaded, compatibility with older PDS versions may still be achieved.

The rule for versioning is as follows:

- VERSION_MAJOR is increased when there is a compatibility issue
  - Then VERSION_MINOR restarts at 0
- VERSION_MINOR is increased for any change that is backward compatible, including
  - Additional features (new section or new field)
  - Fields renaming (made compatible by the update of definitions.in file)
  - Changed comments

In example below, PDS API format is 4.0.

Table 3.2. Example of a PDS Header

```
// PDS API version
HEADER: {
    VERSION_MAJOR: 4,
    VERSION_MINOR: 0
},
```

3.3 PROG_PINS_CFG Section

Most of digital pins can be configured either as functional pins used by WF200 firmware, GPIOs or can be set in tristate. These pins can optionally be configured with an additional pull-up or pull-down resistor for system optimization. Control of pad slew rate is also possible on output pins. Different settings can be configured for active states and sleep modes (sleep, snooze and sleep with XTAL). Note that the pull-up or pull-down resistors are not available when WF200 is in shutdown mode where all pins are in tristate. 13 pins are programmable:

- GPIO_FEM_x (x=1 to 6 + PDET)
- GPIO_PTA_x (x=1 to 4)
- GPIO_WUP
- GPIO_WIRQ

Table 3.3. Abstract of PDS Section PROG_PINS_CFG

```
PROG_PINS_CFG: {
    // For each programmable pin in this section
    // SLEW_RATE sets the maximum slew rate on the pin. Type: integer value between 0 and 6 (6=max drive
strength)
    // PULL_UP_DOWN allows to add a pull-up or pull-down on the pad. Type: enum = 'none', 'down', 'up'
    // SLEEP_CFG allows to add a pull-up or pull-down on the pad while in sleep mode. Type: enum = 'none',
'down', 'up', 'maintain'
    //          for the case of pads used as gpio it is also possible to maintain the current driven gpio
value
    // PIN_MODE allows to configure the pin in tristate, functional mode or gpio. Type: enum = 'tri',
'func', 'gpio'
    // GPIO_ID allows to assign a GPIO_ID to a given pin when configured as gpio. Type = char must be an
UPPER case letter
    GPIO_FEM_1: {
        SLEW_RATE: 4,
        PULL_UP_DOWN: none,
        SLEEP_CFG: none,
        PIN_MODE: tri,
        GPIO_ID: A
    },
```

Table 3.4. Details of PDS Section PROG_PINS_CFG

Field	Description	Enumerates and Values	Default Value	Comments
SLEW_RATE	The slew rate allows to control the steepness of the edges of output signal	Integer between 0 and 6 0: slowest edges 6: fastest edges	4	The main purpose is to optimize the spurious emissions versus timing potential trade-off. To a lower extend, slew rate can have an impact on power consumption.
PULL_UP_DOWN	The pins can be configured with an optional pull-up or pull-down when not in sleep modes	none: neither pull-up nor pull-down down: pull-down enabled up: pull-up enabled	none	The pull-up or pull down configured is always active whatever the PIN_MODE value, except during sleep modes where SLEEP_CFG settings apply.
SLEEP_CFG	This field allows to configure the pin status while the device is in sleep modes, i.e. between beacons reception when power save is activated, and without traffic	none: neither pull-up nor pull-down down: pull-down enabled up: pull-up enabled maintain: in this case, a pull-up or pull-down is added depending on the pin status before going to sleep	none	During sleep, all programmable IOs are set in tristate. Then this field allows to control the status of the pin while the device is in sleep modes, in order to: - maintain system operation: typically, a signal should not be left floating or PTA output pin will be configured to give priority to the coexisting RF during sleep. - optimize power consumption during sleep: if the selected pull-up/down mode within SLEEP_CFG is different from the one when active, then the pin status will change between active and sleep mode. SLEEP_CFG operates whatever the PIN_MODE value. It is recommended to use maintain only for pins configured as output.
PIN_MODE	This field is used to select the operating mode of each pin	tri: pin is set in tristate func: pin is set in its functional mode gpio: pin is used as a GPIO	tri	Functional mode depends on each pin as described in Table 3.5 Table 4.3 on page 16
GPIO_ID	This field allows to associate a GPIO_ID to a pin used as GPIO to simplify software development	Must be a single upper-case letter	n/a	Field GPIO_ID is meaningless if PIN_MODE is not set to gpio. This field creates the link between hardware and software to facilitate management. For example, if three GPIOs are used to control a red, a yellow and a green LED, then GPIO_IDs for these pins can be set respectively to “R”, “Y” and “G”, and software does not need to care anymore about the pin used. Note that in case 2 identical GPIO_IDs are used in PDS (including pins not set in GPIO mode), only the last one is valid.

Table 3.5. Details of Functional Modes of PDS Section PROG_PINS_CFG

Pin	Description	Pin Type	Comments
GPIO_FEM1, GPIO_FEM2, GPIO_FEM3, GPIO_FEM5, GPIO_FEM6	When configured in functional mode, these pins can be used to control a Front-End Module (FEM) as detailed in PDS section FEM_CFG	Digital output referring to VDD_IO logic levels	For full details on FEM implementation, refer to AN1226.
GPIO_FEM4	When configured in functional mode, this pin can be used to enable the Power Amplifier (PA) of a FEM as detailed in PDS section FEM_CFG		
GPIO_FEM_PDET	When configured in functional mode, this pin is typically connected to the Power detector output of a FEM, to control TX output power	Analog input. ranging from 0V to 1.2V. **Warning**: this pin is a digital I/O when used as GPIO	
GPIO_PTA_TX_CONF, GPIO_PTA_RF_ACT, GPIO_PTA_STATUS, GPIO_PTA_FREQ	When configured in functional mode, these pins are used as part of the Packet Traffic Arbitration (PTA) interface to manage coexistence (COEX) with other RF protocols. PTA can use 1 to 4 wires. PTA_FREQ is used only for 4-wire PTA. PTA_STATUS is used only for 3 & 4-wire PTA. PTA_RF_ACT is used for 2,3 and 4-wire PTA. PTA_TX_CONF is also used if PTA is enabled. Dynamic aspects of the PTA is achieved through WF200 firmware API. Only PTA pin assignments are managed through PDS configuration.	Digital I/O referring to VDD_IO logic levels	For full details on PTA implementation, refer to AN1224.
GPIO_WUP	This pin is used by the host to wake-up device when power save is enabled. **It is required to set this pin in functional mode to reach sleep states for low power cases.** When this pin is configured in functional mode, it should be LOW to achieve sleep states and shutdown. When this pin is not configured in functional mode, sleep states are not reachable (device stays in Rx_listen mode), but shutdown can be reached.		More details on the device pins GPIO/WUP and GPIO/WIRQ can be found in Appendix B of AN1219.
GPIO_WIRQ	IRQ is required when WF200 needs to inform the host that there is data to read in the host interface buffer. When configured in functional mode, this pin can be used as a duplication of the IRQ signal from SPI or SDIO. It can typically be used when the host MCU is set in low power modes where SPI or SDIO interface is not active.		

3.4 HIF_PINS_CFG Section

Some configuration is available on the Host Interface (HIF) pins dedicated to SPI or SDIO bus:

- SDIO_CLK_SPI_CLK for device pin SDIO_CLK/SPI_CLK
- SDIO_CMD_SPI_MOSI for device pin SDIO_CMD/SPI_MOSI
- SDIO_D0_SPI_MISO for device pin SDIO_DAT0/SPI_MISO
- SDIO_D1_SPI_WIRQ for device pin SDIO_DAT1/SPI_WIRQ
- SDIO_D2_HIF_SEL for device pin SDIO_DAT2/HIF_SEL
- SDIO_D3_SPI_CSN for device pin SDIO_DAT3/SPI_CSn

Table 3.6. PDS Section HIF_PINS_CFG

```
HIF_PINS_CFG: {
    // For each HIF pin in this section
    // SLEW_RATE sets the maximum slew rate on the pin. Type: integer value between 0 and 6 (6=max drive
strength)
    // SLEEP_CFG for SDIO_D0_SPI_MISO allows to add a pull-up or pull-down on the pad while in sleep mode.
Type: enum = 'none', 'down', 'up'
    SDIO_CLK_SPI_CLK: {
        SLEW_RATE: 4
    },
    SDIO_CMD_SPI_MOSI: {
        SLEW_RATE: 6
    },
    SDIO_D0_SPI_MISO: {
        SLEW_RATE: 6,
        SLEEP_CFG: none
    },
    SDIO_D1_SPI_WIRQ: {
        SLEW_RATE: 6
    },
    SDIO_D2_HIF_SEL: {
        SLEW_RATE: 6
    },
    SDIO_D3_SPI_CSN: {
        SLEW_RATE: 6
    }
},
```

Table 3.7. Details of PDS Section HIF_PINS_CFG

Field	Description	Enumerates and Values	Default Value	Comments
SLEW_RATE	The slew rate allows to control the steepness of the edges of output signal	Integer between 0 and 6 0: slowest edges 6: fastest edges	6 for all pins except SDIO_CLK_SPI_CLK which is set to 4 (always an input)	The main purpose is to optimize the spurious emissions versus timing potential trade-off. To a lower extend, slew rate can have an impact on power consumption.
SLEEP_CFG	This field configures the pin status while the device is in sleep modes, i.e., between beacons reception when power save is activated, and without traffic	none: neither pull-up nor pull-down down: pull-down enabled up: pull-up enabled	none	During sleep, host interface pins are all set as inputs. In SDIO, this should not be a problem as all signals should have a pull-up (as defined in SDIO standard), so the levels are set. In SPI, the MISO pin is an input to the host, so it becomes a floating signal if it is also an input on device side. Then, a pull-up or a pull-down can be used to prevent this signal to float. Other signals from SPI interface are managed by the host and do not require this feature.

3.5 HF_CLK Section

This section allows configuration of the 38.4MHz crystal oscillator. It includes 3 sub-sections:

- XTAL_CFG controls load capacitors on crystal oscillator to ensure less than +/-25ppm frequency tolerance on 2.4GHz Wi-Fi transmitted signals.
- XTAL_SHARED indicates if the crystal is shared with another device.
- XTAL_TEMP_COMP enables temperature of the crystal, **applicable only to WFM200**.

Table 3.8. PDS Section HIF_CLK

```
HF_CLK: {
    // XTAL_CFG allows to fine tune XTAL Oscillator frequency
    XTAL_CFG: {
        // CTUNE_FIX allows to set a high value capacitance on both XTAL_I and XTAL_O. Type: integer between
0 and 3 (de-fault = 3)
        CTUNE_FIX: 3,
        // CTUNE_XI allows to fine tune the capacitor on pin XTAL_I. Type: integer between 0 and 255
(default = 140)
        CTUNE_XI: 41,
        // CTUNE_XO similar to CTUNE_XI for XTAL_O pin
        CTUNE_XO: 41
    },
    // XTAL_SHARED indicates if the crystal is shared with another IC and thus must be kept active during
sleep. Type: enum = 'no', 'yes'
    XTAL_SHARED: no,
    // XTAL_TEMP_COMP activates or deactivates the XTAL temperature compensation. Type: enum = 'enabled',
'disabled'
    XTAL_TEMP_COMP: disabled
},
```

Table 3.9. Details of PDS section HF_CLK

Field		Description	Enumerates and Values	Default Value	Device Operation	Comments
XTAL_CFG	CTUNE_FIX	Configures raw load capacitance in case the load is not balanced	0 to 3 (See Table 3.10 on page 19)	3	Shall be commented for WFM200 to use the calibrated settings stored in OTP, otherwise they are overwritten with the XTAL_CFG settings	CTUNE_FIX should be 3 (default) for the balance case.
	CTUNE_XI	Fine tuning of the load capacitance on device pin XTAL_I	0 to 255 80fF per unit	140		In case of large unbalance between CTUNE_XI and CTUNE_XO, to achieve balanced Cload, value of CTUNE_FIX should be changed to keep room on CTUNE_XI and CTUNE_XO.
	CTUNE_XO	Fine tuning of the load capacitance on device pin XTAL_O		140		
XTAL_SHARED		Indicates if the XTAL clock is shared with another device	No: Xtal not shared Yes: Xtal shared	No	Must be set to No for WFM200 as it is not shared	If the Xtal is shared: - the load unbalance should be solved by decreasing the load on the crystal pin that has higher load. - the Xtal oscillator is not shut down while WF200 is in sleep modes, so the power consumption is higher.
XTAL_TEMP_COMP		Controls XTAL temperature compensation	Enabled: temperature compensation is enabled Disabled: temperature compensation is disabled	Disabled	Must be set to disabled for WF200 as it is only supported on WFM200	WFM200 embeds a thermistor and a XTAL with known characteristics versus temperature (calibrated). When XTAL_TEMP_COMP is enabled, this allows operation over -40 to +105 C temperature range.

CTUNE_FIX is used to achieve raw value of crystal load capacitor as shown in the table below.

Table 3.10. Details of PDS Node CTUNE_FIX

CTUNE_FIX	Capacitive Load on Pin XTAL_IN (pF)	Capacitive Load on Pin XTAL_OUT (pF)
0	6.5	4
1	9	4
2	6.5	9
3	9	9

3.6 FEM_CFG Section

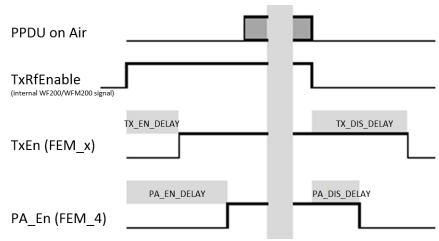
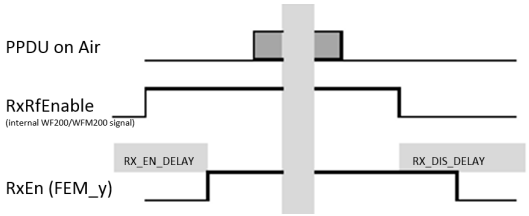
This section allows configuration of the pins GPIO/FEM_1 to GPIO/FEM_6 controlling the Front End Module (FEM), except GPIO/FEM_4 which is allocated to enable FEM power amplifier. FEM_CFG section includes 2 sub-sections:

- FEM_CTRL_PINS defines the states of FEM pins to dynamically configure the FEM according to PTA status and WLAN device state
- FEM_TIMINGS allows to adjust timings of FEM control signals

Table 3.11. PDS Section FEM_CFG

```
FEM_CFG: {
    FEM_CTRL_PINS: {
        // defines state of FEM pins 1 to 6 depending on priority given to COEX and WLAN, and WLAN interface
TX/RX state
        // notes:
        // - each bit indicates the pin level for each state
        // - pin FEM_4 is not present because it is the PA_enable signal
        // - keys with prefix WLAN_ONLY are the only used if PTA is not enabled
        //                       .-- FEM_6
        //                       | .- FEM_5
        //                       | |.- FEM_3
        //                       | ||.- FEM_2
        //                       | |||.- FEM_1
        //                       | ||||
        WLAN_ONLY_IDLE        0b0_0000,// FEM control signals for WLAN when not transmitting nor receiving
        WLAN_ONLY_RX:         0b0_0000,// FEM control signals for WLAN when receiving
        WLAN_ONLY_TX:         0b0_0000,// FEM control signals for WLAN when transmitting
        COEX_ONLY:            0b0_0000,// FEM control signals to provide the antenna to coexisting RF
        COMBINED_WLAN_IDLE:   0b0_0000,// control signals to set FEM in Rx for both WLAN and COEX (WLAN not
receiving)
        COMBINED_WLAN_RX:     0b0_0000,// control signals to set FEM in Rx for both WLAN and COEX (WLAN
actually receiving)
    },
    FEM_TIMINGS: {
        // define related timings on FEM signals
        // Delays are in 12.5ns units
        // Format integer
        TX_EN_DELAY:  16,  // max 65535, default value 16 => 0.2 µs
        TX_DIS_DELAY: 13,  // max 255, default value 13 => 0.1625 µs
        PA_EN_DELAY:  130, // max 255, default value 130 => 1.625 µs
        PA_DIS_DELAY: 5,   // max 255, default value 5 => 0.0625 µs
        RX_EN_DELAY:  0,   // max 255, default value 0
        RX_DIS_DELAY: 0    // max 255, default value 0
    }
},
```

Table 3.12. Details of PDS section FEM_CFG

Field		Description	Enumerates and Values	Default Value
FEM_CTRL_PINS	WLAN_ONLY_IDLE	Level of FEM pins when in WLAN and idle (neither TX nor RX)	Five bits FEM[6,5,3,2,1] set according to application schematics 0 for logic level “0” 1 for logic level “1”	0
	WLAN_ONLY_RX	Level of FEM control signals for WLAN when receiving		0
	WLAN_ONLY_TX	Level of FEM control signals for WLAN when transmitting		0
	COEX_ONLY	Level of FEM control signals to provide the antenna to coexisting RF		0
	COMBINED_WLAN_IDLE	Level of control signals to set FEM in Rx for both WLAN and COEX when WLAN is not receiving		0
	COMBINED_WLAN_RX	Level of control signals to set FEM in Rx for both WLAN and COEX when WLAN is receiving		0
FEM_TIMINGS	TX_EN_DELAY		Integer 12.5 ns unit max 65535	16 => 0.2 µs
	TX_DIS_DELAY		Integer 12.5 ns unit max 255	13 => 0.16 µs
	PA_EN_DELAY			130 => 1.62 µs
	PA_DIS_DELAY			5 => 0.06 µs
	RX_EN_DELAY			0
	RX_DIS_DELAY			0

3.7 RF_POWER_CFG Section

This section allows configuration of the TX input/output port as well as TX output power versus bit rate and RF channel thanks to sub-section BACKOFF_QDB.

Table 3.13. PDS Section RF_POWER_CFG

```
RF_POWER_CFG: {
 // Designate the RF port affected by the following configurations
 // RF_PORT_1, RF_PORT_2, RF_PORT_BOTH (default)
    RF_PORT: RF_PORT_BOTH,
 // The max Tx power value in quarters of dBm. Type: signed integer between -128 and 127 (range in dBm:
[-32; 31.75])
 // It is used to limit the Tx power. Thus a value larger than 80 does not make sense.
    MAX_OUTPUT_POWER_QDBM: 80,
 // Front-end loss (loss between the chip and the antenna) in quarters of dB.
 // Type: signed integer between -128 and 127 (range in dB: [-32; 31.75])
 // This value must be positive when the front end attenuates the signal and negative when it amplifies it.
 // Values on Rx path and Tx path can be different thus 2 values are provided here
    FRONT_END_LOSS_TX_QDB: 0,
    FRONT_END_LOSS_RX_QDB: 0,
 // Backoff vs. Modulation Group vs Channel
 // CHANNEL_NUMBER: Designate a channel number (an integer) or a range of channel numbers (an array)
 // e.g. CHANNEL_NUMBER: [3, 9] : Channels from 3 to 9
 // Each backoff value sets an attenuation for a group of modulations.
 // BACKOFF_VAL is given in quarters of dB. Type : unsigned integer. Covered range in dB: [0; 63.75]
 // A modulation group designates a subset of modulations :
 //   *  MOD_GROUP_0 : B_1Mbps, B_2Mbps, B_5.5Mbps, B_11Mbps
 //   *  MOD_GROUP_1 : G_6Mbps, G_9Mbps, G_12Mbps, N_MCS0,  N_MCS1,
 //   *  MOD_GROUP_2 : G_18Mbps, G_24Mbps, N_MCS2, N_MCS3,
 //   *  MOD_GROUP_3 : G_36Mbps, G_48Mbps, N_MCS4, N_MCS5
 //   *  MOD_GROUP_4 : G_54Mbps, N_MCS6
 //   *  MOD_GROUP_5 : N_MCS7
 //   BACKOFF_VAL: [MOD_GROUP_0, ..., MOD_GROUP_5]
 BACKOFF_QDB: [
        {
            CHANNEL_NUMBER: 1,
            BACKOFF_VAL: [0, 0, 0, 0, 0, 0]
        },
        {
            CHANNEL_NUMBER: 2,
            BACKOFF_VAL: [0, 0, 0, 0, 0, 0]
        },
        {
            CHANNEL_NUMBER: [3, 9],
            BACKOFF_VAL: [0, 0, 0, 0, 0, 0]
        },
        {
            CHANNEL_NUMBER: 10,
            BACKOFF_VAL: [0, 0, 0, 0, 0, 0]
        },
        {
            CHANNEL_NUMBER: 11,
            BACKOFF_VAL: [0, 0, 0, 0, 0, 0]
        },
        {
            CHANNEL_NUMBER: [12, 13],
            BACKOFF_VAL: [0, 0, 0, 0, 0, 0]
        },
        {
            CHANNEL_NUMBER: 14,
            BACKOFF_VAL: [0, 0, 0, 0, 0, 0]
        }
    ]
}
```

Table 3.14. Details of PDS Section RF_POWER_CFG

Field	Description	Enumerates and Values	Default Value	Comments
RF_PORT	This field selects if the section applies to RF port 1, RF port 2 or both.	RF_PORT_1 RF_PORT_2 RF_PORT_BOTH	RF_PORT_BOTH	In case settings are different between RF port 1 and port 2, this section has to be repeated once for each RF port.
MAX_OUTPUT_POWER_QDBM	This field allows to further limit the maximum TX power target from device internal maximum value stored in OTP.	Unit is ¼ of dBm. The value should be a signed integer between -128 and 127, corresponding to a range of -32 to + 31.75 dBm.	80	Maximum TX power target **Ptar** is the minimum of (OTP value, MAX_OUTPUT_POWER_QDBM+FRONT_END_LOSS_TX_QDB) .
FRONT_END_LOSS_TX_QDB	This field allows to compensate for loss on the transmission side, between the RF port and the antenna.	Unit is ¼ of dB. The value should be a signed integer between -128 and 127, corresponding to a range of -32 to + 31.75 dB.	0	
FRONT_END_LOSS_RX_QDB	This field allows to compensate for loss on the reception side between the RF port and the antenna.		0	This field is taken into account during computation of Received Signal Strength Indicator (RSSI).
BACKOFF_QDB	This sub-section defines the attenuation on nominal output power applied per channel number and modulation group. For WF200 these are the backoff values set by user to ensure regulations such as FCC are met. For WFM200, backoff values are already stored in OTP, then this table is used for additional backoffs if required.	The unit is ¼ of dB. The value should be an integer between 0 and 255, corresponding to a range of 0 to + 63.75 dB.	0	For WF200, this table defines the backoff values to apply on nominal power **Pnom** for the current modulation and code rate as detailed in UG382. The real power at device RF port is the minimum between **Pnom** and **Ptar** and varies in 1 dB step.

3.8 RF_ANTENNA_SEL_DIV_CFG Section

This section allows configuration of the way RF ports are used depending on hardware implementation:

- When a single antenna is used, both Rx and TX are achieved with the same RF port and this section allows to select the port used.
- When 2 antennas are used for station (STA) diversity, this section allows to enable diversity.
- When an active FEM is used, TX and RX should be achieved with different ports. This section allows to define the usage of each port.

Table 3.15. PDS Section RF_ANTENNA-SEL_DIV_CFG

```
RF_ANTENNA_SEL_DIV_CFG: {
    //
    // Antenna selection: allows to select the RF port used. Can be different for Tx and Rx (FEM case)
    //   - TX1_RX1: RF_1 used
    //   - TX2_RX2: RF_2 used
    //   - TX1_RX2: Tx on RF_1 and Rx on RF_2
    //   - TX2_RX1: Tx on RF_2 and Rx on RF_1
    //   - TX12_RX12: antenna diversity case: both Tx and Rx on the same RF port which is automatically
selected (requires DIVERSITY to be set)
    //
    RF_PORTS: TX1_RX1,

    //
    // Diversity control mode:
    //   - OFF
    //   - INTERNAL   :  requires RF_PORTS to be set to TX12_RX12
    //
    DIVERSITY: OFF
}
```

Table 3.16. Details of PDS Section RF_ANTENNA_SEL_DIV_CFG

Field	Description	Enumerates and Values	Default Value	Comments
RF_PORTS	This field defines which ports are achieving transmission and reception.	TX1_RX1: device pin RF_1 is connected to a single antenna. TX2_RX2: device pin RF_2 is connected to a single antenna. TX1_RX2: TX on pin RF_1 and Rx on pin RF_2 (FEM case) TX2_RX1: TX on pin RF_2 and Rx on pin RF_1 (FEM case) TX12_RX12: both TX and RX are on the same RF port, requires DIVERSITY to be set.	TX1_RX1	
DIVERSITY	This field configures diversity.	OFF: diversity is disabled INTERNAL: diversity is enabled	OFF	Diversity is not supported in AP, Combo and IBSS modes. When diversity is enabled, port RF_1 is initially selected to connect to AP. Once connected, RF port is automatically selected by the firmware according signal quality.

4. TEST_FEATURE_CFG Section

This specific section allows to perform some RF tests using Python3 module wfx_test_dut available in the wfx-common-tools repository 'test-feature' folder. Unlike other sections, firmware switches from WLAN mode to test feature mode as soon as this section is included into the PDS configuration file. However, the user must ensure that no other program uses WLAN interface. Once device is in test feature mode, removal of section TEST_FEATURE_CFG from PDS does not allow to leave test feature mode and requires in addition to reload device driver. For more details regarding RF test feature mode, refer to https://github.com/SiliconLabs/wfx-common-tools/blob/master/test-feature/README.md.

Table 4.1. PDS Section TEST_FEATURE_CFG

```
TEST_FEATURE_CFG: {
    // TEST_CHANNEL_FREQ can designate either a Wi-Fi channel or a frequency in MHz :
    //   - [1-14]: Wi-Fi channel to use
    //   - [2300 - 2600]: RF frequency in MHz
    //
    TEST_CHANNEL_FREQ: 11,

    // TEST_MODE selects the activated test feature: enum = 'rx', 'tx_packet', 'tx_cw'
    TEST_MODE: tx_packet,

    // TEST_IND period in ms at which an indication message is sent.
    //       In the case of rx test, it returns the measurement results (PER and RSSI)
    TEST_IND: 1000,

    // CFG_TX_CW: additional configuration for tx_cw mode
    CFG_TX_CW: {
        // CW_MODE CW mode, either single tone (use FREQ1) or dual tone: enum 'single' or 'dual'
        CW_MODE: single,
        // FREQ1 frequency offset -31 to 31 (in 312.5kHz)
        FREQ1: 1,
        // FREQ2 frequency offset -31 to 31 (in 312.5kHz)
        FREQ2: 2,
        // MAX_OUTPUT_POWER indicates the max Tx power value (for CW only) in quarters of dBm
        MAX_OUTPUT_POWER: 68
    },

    // CFG_TX_PACKET: additional configuration for tx_packet mode
    CFG_TX_PACKET: {
        // FRAME_SIZE_BYTE frame size in byte (without CRC) from 25 to 4091
        FRAME_SIZE_BYTE: 3000,
        // IFS_US interframe spacing in us from 0 to 255
        IFS_US: 0,
        // HT_PARAM HT format (mixed mode or greenfield), enum: 'MM' or 'GF'
        HT_PARAM: MM,
        // RATE rate selection, enum : B_1Mbps, B_2Mbps, B_5_5Mbps,
        //     B_11Mbps, G_6Mbps, G_9Mbps, G_12Mbps, G_18Mbps,
        //     G_24Mbps, G_36Mbps, G_48Mbps, G_54Mbps, N_MCS0,
        //     N_MCS1, N_MCS2, N_MCS3, N_MCS4, N_MCS5, N_MCS6, N_MCS7
        RATE: N_MCS7,
        // NB_FRAME number of frames to send before stopping. Integer from 0 to 65535 (0 means infinite)
        NB_FRAME: 0,
        // (REGION) REGULATORY MODE : the WFM200 can be certified for US (FCC), Europe(ETSI) or Japan
        // and thus its Tx power is limited internally to fulfill these radio transmission specifications.
        // It is possible to select here the regulatory mode for test purposes. Selection has no impact on
channel usage limitation.
        // enum: CERTIFIED_FCC, CERTIFIED_ETSI, CERTIFIED_JAPAN, CERTIFIED_Unrestricted (no power
limitation), CERTIFIED_All (applies the most restrictive limitation)
        REG_MODE: CERTIFIED_All
    },

    // RX: additional configuration for rx mode
    RX: { }
}
```

5. Revision History

5.1 Revision 0.2

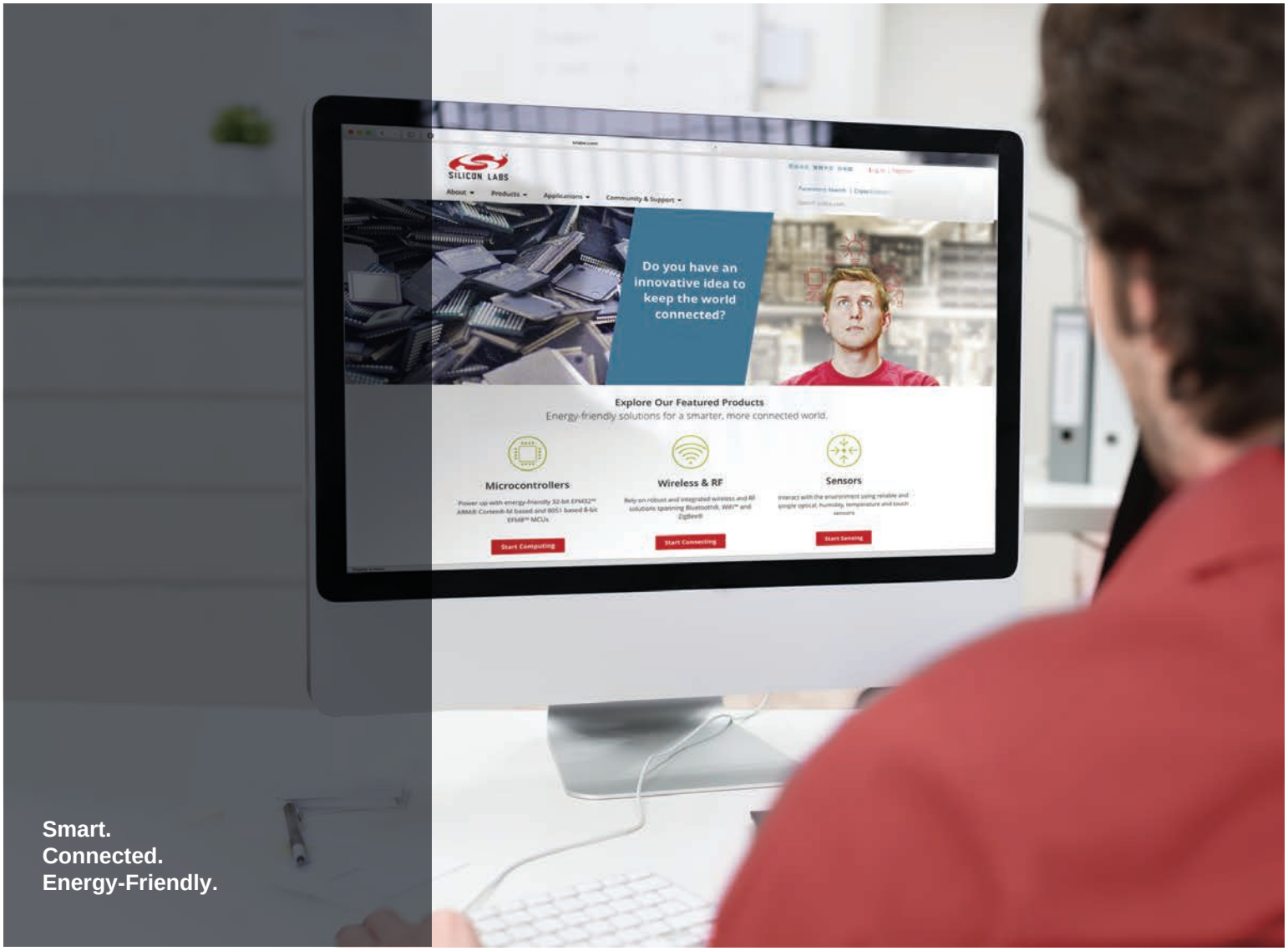
January, 2020

- Minor revision of linked reference in Table 3.5 on page 16.

5.2 Revision 0.1

November, 2019

- Initial release.



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com