AN1127: Power Amplifier Power Conversion Functions in RAIL 2.x

This version of AN1127 has been deprecated.

For the latest version, see docs.silabs.com.

****
In many wireless applications, due to limitations imposed by the transmission protocol, law, or energy constraints, it is critical to know the power output of a given device. However, this issue is complicated by the fact that output characteristics may vary drastically even for a given chip, based on the board or module onto which the chip is mounted, due to poor impedance matching, or due to parasitics. This application note outlines how to account for the variation across custom boards and applications for the Silicon Labs EFR32 family of chips.

While the Proprietary Flex SDK is required for the procedures described in this document, the resulting calibration curves can be used with applications developed with Silicon Labs Connect, Zigbee, Bluetooth, and OpenThread SDKs. Customers developing with Z-Wave should refer to https://www.silabs.com/community/wireless/z-wave/knowledge-base.entry.html/2020/03/03/z-wave_700_tx_powercalibrationandadjustment-YUiB.

Proprietary is supported on all EFR32FG devices. For others, check the device's data sheet under Ordering Information > Protocol Stack to see if Proprietary is supported. In Proprietary SDK version 2.7.n, Connect is not supported on EFR32xG22.

KEY POINTS

- Describes EFR32 Power Amplifiers and their management in RAIL.
- Provides instructions on generating custom curves for the default conversion functions.
- Offers an example of modifying the conversion functions.

1. Overview: EFR32 Power Amplifiers and RAIL

The EFR32 families of chips each come equipped with three Power Amplifiers (PAs):

- EFR32xG1x
  - A high-power * 2.4 GHz PA (for power 20 dBm and lower)**
  - A low-power 2.4 GHz PA (for power 0 dBm and lower)
  - A Sub-GHz PA**
- EFR32xG21
  - A high-power 2.4 GHz PA (for power 20 dBm and lower)**
  - A medium-power 2.4 GHz PA (for power 10 dBm and lower)
  - A low-power 2.4 GHz PA (for power 0 dBm and lower)
- EFR32xG22
  - A high-power 2.4 GHz PA (for power 6 dBm and lower)**
  - A low-power 2.4 GHz PA (for power 0 dBm and lower)
- EFR32xG23x02x
  - A high-power Sub-GHz PA (for power 20 dBm and lower)**
- EFR32xG23x01x
  - A high-power Sub-GHz PA (for power 14 dBm and lower)**
- EFR32xG24x02x
  - A high-power 2.4 GHz PA (for power 20 dBm and lower)**
- EFR32xG24x01x
  - A high-power 2.4 GHz PA (for power 10 dBm and lower)**
  - A low-power 2.4 GHz PA (for power 0 dBm and lower)
- EFR32xG25
  - A single PA for OFDM operation
  - A single PA for SUBGIG (FSK) operation
- EFR32xG26x02x
  - A high-power 2.4 GHz PA (for power 20 dBm and lower)**
- EFR32xG26x01x
  - A high-power 2.4 GHz PA (for power 10 dBm and lower)**
  - A low-power 2.4 GHz PA (for power 0 dBm and lower)
- EFR32xG27x02x
  - A high-power 2.4 GHz PA (for power 4 dBm and lower)**
  - A low-power 2.4 GHz PA (for power 0 dBm and lower)
- EFR32xG27x01x
  - A high-power 2.4 GHz PA (for power 6/8 dBm and lower)**
  - A low-power 2.4 GHz PA (for power 0 dBm and lower)
- EFR32xG28x02x
  - A high-power 2.4 GHz PA (for power 20 dBm and lower)**
  - A high-power Sub-GHz PA (for power 20 dBm and lower)**
- EFR32xG28x01x
  - A high-power 2.4 GHz PA (for power 14 dBm and lower)**
  - A high-power Sub-GHz PA (for power 14 dBm and lower)**
- SixG301
  - A high-power 2.4 GHz PA (for power 10 dBm and lower)**
  - A low-power 2.4 GHz PA (for power 0 dBm and lower)

Each of these PAs has a unique number of discrete “power levels”, which are simply abstractions of the different register settings that control the active PA. For each PA, the following power levels are available:

- EFR32xG1x
  - High-power 2.4 GHz: 0-252**
  - Low-power 2.4 GHz: 1-7
  - Sub-GHz: 0-248

- EFR32xG21
  - High-power 2.4 GHz: 1-180**
  - Medium-power 2.4 GHz: 1-90
  - Low-power 2.4 GHz: 1-64
- EFR32xG22
  - High-power 2.4 GHz: 0-127**
  - Low-power 2.4 GHz: 0-15
- EFR32xG23x02x
  - High-power Sub-GHz: 0-240**
- EFR32xG23x01x
  - High-power Sub-GHz: 0-240**
- EFR32xG24x02x
  - High-power 2.4 GHz: 0-180**
- EFR32xG24x01x
  - High-power 2.4 GHz: 0-90**
  - Low-power 2.4 GHz: 0-15
- EFR32xG25
  - “power levels” not supported on dBm-to-powerSetting Pas (see 2.3 dBm-to-powerSetting Mode)
- EFR32xG26x02x
  - High-power 2.4 GHz: 0-180**
- EFR32xG26x01x
  - High-power 2.4 GHz: 0-90**
  - Low-power 2.4 GHz: 0-15
- EFR32xG27x02x
  - High-power 2.4 GHz: 0-127**
  - Low-power 2.4 GHz: 0-15
- EFR32xG27x01x
  - High-power 2.4 GHz: 0-127**
  - Low-power 2.4 GHz: 0-15
- EFR32xG28x02x
  - High-power 2.4 GHz: 0-240**
  - High-power Sub-GHz: 0-240**
- EFR32xG28x01x
  - High-power 2.4 GHz: 0-240**
  - High-power Sub-GHz: 0-240**
- SixG301
  - “power levels” not supported on dBm-to-powerSetting PAs (see 2.3 dBm-to-powerSetting Mode)

*Note that the use of 'high', 'medium', and 'low' in the names of these PAs refers to power consumption, not power output. It is possible, for instance, to configure the high-power PA to transmit at a lower dBm output than the low-power PA.

**Maximum power/use of these PAs may be restricted by your specific OPN. Please see the data sheet for more details regarding your particular part.

Although the number of power levels vary across PAs, for a given PA a higher power level generally indicates a higher dBm output power. It is important to realize, however, that a PA power level does not always correspond to the same dBm output power across boards and applications.

The power level of the active PA is controlled by the `RAIL_GetTxPower` and `RAIL_SetTxPower` APIs. `RAIL_GetTxPowerDbm` and `RAIL_SetTxPowerDbm` also allow users to interface with dBm output power values, but they are merely utility functions that combine calls to `RAIL_ConvertRawToDbm`, `RAIL_ConvertDbmToRaw`, `RAIL_SetTxPower`, and `RAIL_GetTxPower`. They add no new functionality or features on their own. See the RAIL documentation that comes with the Simplicity Studio SDK for more details.

Although these PA power levels add another layer of complexity to PA control, they give the user much more control to account for parasitics and impedance mismatch between the PA and a custom board. In RAIL 1.x, customers were only able to interface with PA output power in dBm and provide a linear offset, which was often not sufficiently robust to account for the complex PCB-to-PA interaction. Now, RAIL allows users to assign an exact dBm power to each of the PA power levels for their custom boards through the functions `RAIL_ConvertDbmToRaw` and `RAIL_ConvertRawToDbm`. Note that these functions convert between raw power levels and deci-dBm, or dBm * 10, to allow for higher precision dBm values. Though Silicon Labs provides a default implementation for these functions to do the conversion, the user can also override these functions with other algorithms, ranging from a highly precise but space-intensive lookup table for all possible PA-power level combinations, to less robust solutions such as a lookup table containing only a few power levels that correspond to the dBm output power levels of interest (see 3. Appendix: Alternative Methods to Convert between Power Levels and dBm Output Power for more information on creating custom conversion functions).

In the default conversion function implementation, Silicon Labs uses an 8-segment, piecewise linear curve fit of the mapping between PA power levels and dBm output powers to convert between these quantities. On EFR32xG1x chips, there is one curve for the high-power 2.4GHz PA, and another for the sub-GHz PA. On EFR32xG21 and EFR32xG23 chips, there is one curve for each PA. On EFR32xG22 and EFR32xG24 chips, there is one curve for the high power 2.4GHz PA. For these PAs, this method provides a good balance between accuracy across power levels, computational speed, and code size. The conversion for low-power 2.4GHz on EFR32xG1x, EFR32xG22, and EFR32xG24, which have fewer than 20 possible values, is handled with a simple lookup table. The majority of this application note focuses on how to generate the curve data (slopes and y-intercepts) for a custom application, similar to the default values currently found in **pa_curves_efr32xg1x.h** and **pa_curves_efr32xg2x.h** in RAIL 2.8 and earlier versions and in **sl_rail_util_pa_curves.h** in RAIL 2.9 and later versions. In other words, it assumes that the existing methodologies found in `RAIL_ConvertDbmToRaw` and `RAIL_ConvertRawToDbm` are appropriate for your application and will not be over- written, but that the parameters for these functions need to be adjusted for the custom board.

This application note is not intended for use with Silicon Labs' pre-certified Wireless SiP and PCB modules. The power amplifiers on these parts have been calibrated specifically to adhere to certain regulations. Trying to alter these calibrations in software will cause the RAIL library to assert and lock the chip, preventing further use until Silicon Labs' default libraries are restored.

In the following procedures, use the **railtest** and **pa_customer_curve_fits.py** script from RAIL 2.5 or higher in Proprietary Flex SDK version 2.4.0 or higher, even if you are developing in an earlier Proprietary Flex SDK 2.x version. There were improvements to the app and script that will subtly improve results. If you have already generated the CSV files from the procedure in section 2.1 Piecewise Linear Curve Fits, you can simply re-run the newer **pa_customer_curve_fits.py** on the existing CSVs. There is no need to re-generate the CSVs by running the sweep again.

2. Generating Conversion Data

2.1 Piecewise Linear Curve Fits

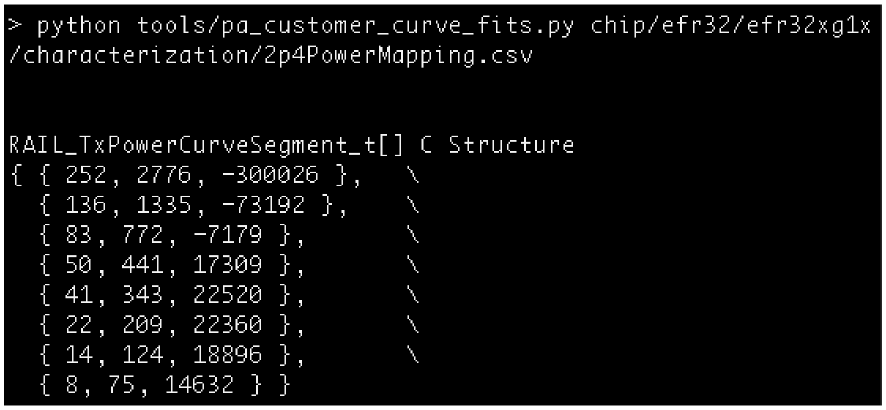
The following steps explain how to generate the piecewise fit curves for the following PAs:

- EFR32xG1x: High-power 2.4 GHz and sub-GHz
- EFR32xG21: All PAs
- EFR32xG22: High-power 2.4 GHz
- EFR32xG23: All PAs
- EFR32xG24: High-power 2.4 GHz

The procedure uses the 2.4 GHz high-power PA on EFR32xG1x devices as an example, but the same procedure is applicable to all the aforementioned PAs. For the other PAs, which do not use the piecewise linear fit approach, please see section 2.2 Lookup Table Fits.

1. In Simplicity Studio, generate a RAILtest application configured with the desired PA, PA power source, and PA ramp time. The configuration process depends on the Proprietary Flex SDK version. If you are using version 2.x with Simplicity Studio 4, use Hardware Configurator (see *AN1115: Configuring Peripherals for 32-Bit Devices using Hardware Configurator* for more detail). If you are using version 3.x with Simplicity Studio 5, use the Component Editor to modify the **RAIL Utility, Power Amplifier** component. Build the RAILtest application, and flash the resulting binary to the chip mounted to the PCB you are trying to characterize.
2. Connect the RF output of the board (U.FL test connector, SMA connector, or similar, with antenna disconnected from the RF path) to the input of a spectrum analyzer. The spectrum analyzer should be in dBm peak detection mode to sense values between approximately -30 to 20 dBm and centered on the base frequency/channel for your PHY and channel combination. Make sure to account for any cable / adapter loss in your measurements.
3. Connect to the RAILtest CLI and enter the command `sweepTxPower`. For more information on RAILtest functionality and the CLI interface, see *Using RAILTest* in the Simplicity Studio Flex SDK RAIL documents group (RailTestUserGuide.html).
4. Your chip should begin a CW tone at power level 1. Record the power level and dBm power measured on the spectrum analyzer in a CSV file in the format `<POWER_LEVEL>,<ACTUAL_DBM_OUTPUT>` with one reading per line. Note that unlike the conversion functions `RAIL_GetTxPowerDbm` and `RAIL_SetTxPowerDbm`, which use deci-dBm values, values in these csv files should be specified in dBm. Sample files **SubgigPowerMapping.csv** and **2p4PowerMapping.csv** are included in the SDK for reference, and can be found in the `/platform/radio/rail_lib/chip/efr32/efr32xg1x/characterization` folder of the SDK installation.
5. Repeat step 4 for all power levels. Press enter in the RAILtest CLI to progress to the next power level.
6. From a terminal prompt, run the **pa_customer_curve_fits.py** script (in /platform/radio/rail_lib/tools), with the `python pa_customer_curve_fits.py <CSV_FILE>`. If you don't already have them installed, you will need to install the **numpy** and **pylab** python libraries. Running the commands `pip install numpy` and `pip install matplotlib` will install those libraries in the appropriate directories on your machine to be accessible by python. The following figure shows an example of an intended result, after the **pa_customer_curve_fits.py** script has been run on the **2p4PowerMapping.csv** file from a terminal prompt.

   Note: The script also provides extra parameters such as –maxPower/-m and –increment/-i to accommodate the maximum power from the csv and to divide the data in segments of needed "increment" dBm steps.

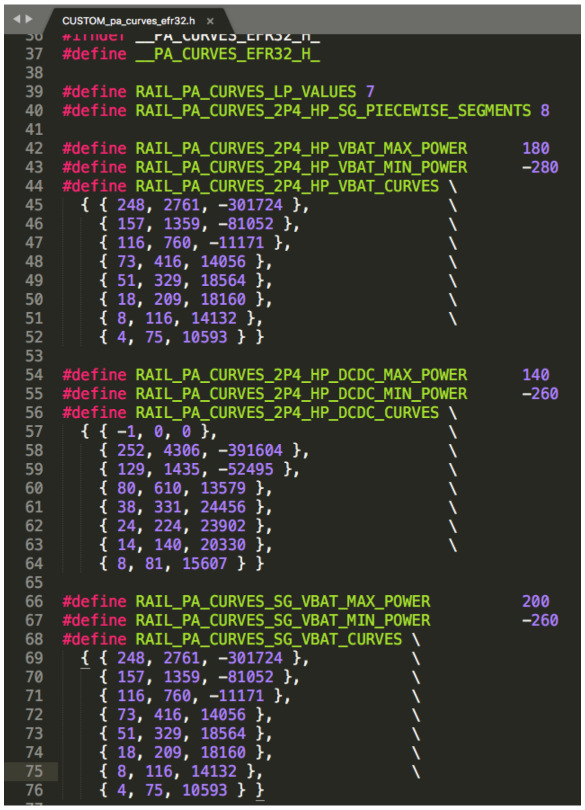


```
> python tools/pa_customer_curve_fits.py chip/efr32/efr32xg1x
/characterization/2p4PowerMapping.csv


RAIL_TxPowerCurveSegment_t[] C Structure
{ { 252, 2776, -300026 },   \
  { 136, 1335, -73192 },    \
  { 83, 772, -7179 },       \
  { 50, 441, 17309 },       \
  { 41, 343, 22520 },       \
  { 22, 209, 22360 },       \
  { 14, 124, 18896 },       \
  { 8, 75, 14632 } }
```

Figure 2.1. pa_customer_curve_fits.py Script Results

7. If using multiple PAs (other than the EFR32xG1x 2.4 GHz low-power PA), repeat steps 1-6 for the PA you did not characterize previously.

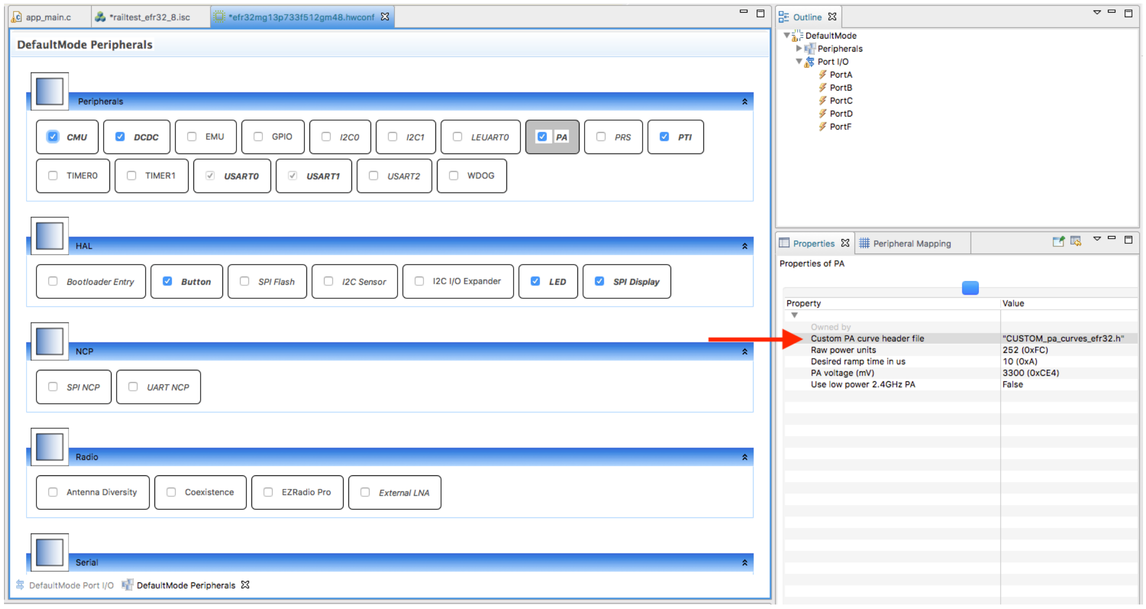
8. Create a copy of the PA curve file (here and in subsequent steps, "PA curve file" refers to **pa_curves_efr32xg1x.h** or **pa_curves_efr32xg2x.h** in RAIL 2.8 and earlier versions, and **sl_rail_util_pa_curves.h** in RAIL 2.9 versions and later depending on the platform currently being characterized). Copy the terminal output from the python script under the appropriate macro. Note that there is one `RAIL_PA_CURVES_…` define for each combination of PA and PA power source available for your hardware. You do not need to copy data for a PA/PA power supply (`…_VBAT_…` or `…_DCDC_…`) combination that is not being used. Additionally, update the `…_MIN_POWER` and `…_MAX_POWER` macros to reflect the values you observed. The following figure shows an example of the copying process, where the values from the previous figure have been copied into the appropriate (Sub-GHz and 2.4 GHz, battery-powered) macros in a new **CUSTOM_pa_curves_efr32xg1x.h** file. Some of the `…MIN_POWER` and `…_MAX_POWER` macros have been updated.



```
#ifndef __PA_CURVES_EFR32_H_
#define __PA_CURVES_EFR32_H_

#define RAIL_PA_CURVES_LP_VALUES 7
#define RAIL_PA_CURVES_2P4_HP_SG_PIECEWISE_SEGMENTS 8

#define RAIL_PA_CURVES_2P4_HP_VBAT_MAX_POWER      180
#define RAIL_PA_CURVES_2P4_HP_VBAT_MIN_POWER      -280
#define RAIL_PA_CURVES_2P4_HP_VBAT_CURVES \
  { { 248, 2761, -301724 },               \
    { 157, 1359, -81052 },                \
    { 116, 760, -11171 },                 \
    { 73, 416, 14056 },                   \
    { 51, 329, 18564 },                   \
    { 18, 209, 18160 },                   \
    { 8, 116, 14132 },                    \
    { 4, 75, 10593 } }

#define RAIL_PA_CURVES_2P4_HP_DCDC_MAX_POWER      140
#define RAIL_PA_CURVES_2P4_HP_DCDC_MIN_POWER      -260
#define RAIL_PA_CURVES_2P4_HP_DCDC_CURVES \
  { { -1, 0, 0 },                         \
    { 252, 4306, -391604 },               \
    { 129, 1435, -52495 },                \
    { 80, 610, 13579 },                   \
    { 38, 331, 24456 },                   \
    { 24, 224, 23902 },                   \
    { 14, 140, 20330 },                   \
    { 8, 81, 15607 } }

#define RAIL_PA_CURVES_SG_VBAT_MAX_POWER          200
#define RAIL_PA_CURVES_SG_VBAT_MIN_POWER          -260
#define RAIL_PA_CURVES_SG_VBAT_CURVES \
  { { 248, 2761, -301724 },           \
    { 157, 1359, -81052 },            \
    { 116, 760, -11171 },             \
    { 73, 416, 14056 },               \
    { 51, 329, 18564 },               \
    { 18, 209, 18160 },               \
    { 8, 116, 14132 },                \
    { 4, 75, 10593 } }
```

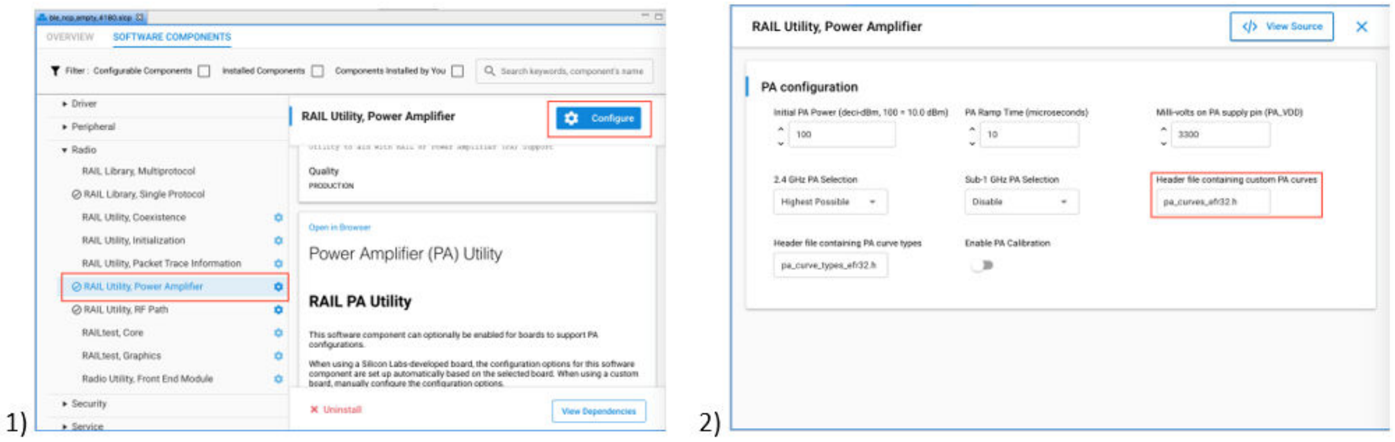
Figure 2.2. Terminal Output Copied into a Custom pa_curves_efr32xg1x.h File

9. Update the hardware config `HAL_PA_CURVE_HEADER` header to point to your new file. The following figures illustrates where to update the header.The interface depends on the tool used (Hardware Configurator in Proprietary Flex SDK 2.x, or Component Editor in Proprietary Flex SDK 3.x).


Properties of PA

Property	Value
Owned by	
Custom PA curve header file	"CUSTOM_pa_curves_efr32.h"
Raw power units	252 (0xFC)
Desired ramp time in us	10 (0xA)
PA voltage (mV)	3300 (0xCE4)
Use low power 2.4GHz PA	False

Figure 2.3. Updating the Custom PA Curve Header File Name with Hardware Configurator


RAIL Utility, Power Amplifier

PA configuration

Initial PA Power (deci-dBm, 100 = 10.0 dBm)	100
PA Ramp Time (microseconds)	10
Milli-volts on PA supply pin (PA_VDD)	3300
2.4 GHz PA Selection	Highest Possible
Sub-1 GHz PA Selection	Disable
Header file containing custom PA curves	pa_curves_efr32.h
Header file containing PA curve types	pa_curve_types_efr32.h
Enable PA Calibration	

Figure 2.4. Updating the Custom PA Curve Header File Name with Component Editor

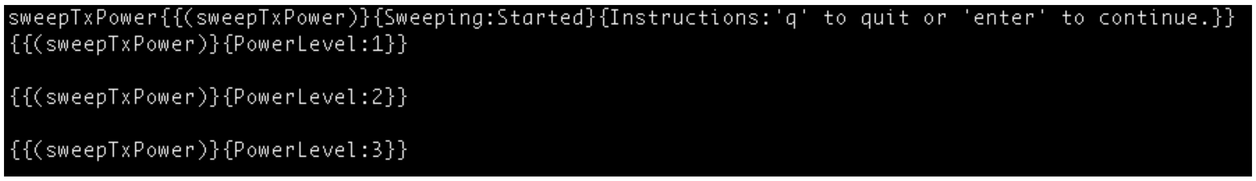
10. For customers using only one or two of the available PAs, Silicon Labs recommends removing references to these functions from **pa_conversions_efr32.c** and from the `RAIL_DECLARE_TX_POWER` macros in your new version of the PA curves file. Although not strictly necessarily, doing this can save substantial code size versus compiling in useless data for a PA that is never engaged.

2.2 Lookup Table Fits

This information applies to the following PAs:

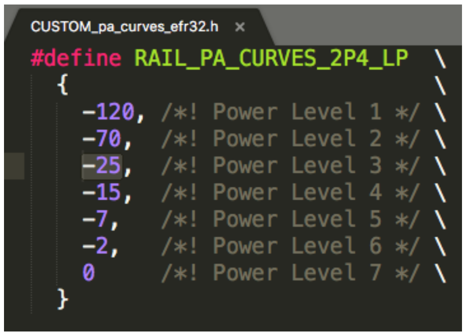
- EFR32xG1x: Low-power 2.4 GHz
- EFR32xG22: Low power 2.4 GHz
- EFR32xG24x01x: Low power 2.4 GHz

With only a few unique power levels, some PAs on some chip families use a simple lookup table to translate power levels to output powers, as opposed to a more code-size-intensive curve fit. Using the EFR32xG1x low-power PA as an example, follow steps 1-5 in the previous section, but for step 4, place the results directly into the RAIL_PA_CURVES_2P4_LP macro of the custom PA curves file. These values should be entered as deci-dBm (for example. enter -100 to indicate -10 dBm). A similar process can be followed for any other PAs using the lookup table approach. The following two figures contain an example of this process.



```
sweepTxPower{{(sweepTxPower)}{Sweeping:Started}{Instructions:'q' to quit or 'enter' to continue.}}
{{(sweepTxPower)}{PowerLevel:1}}

{{(sweepTxPower)}{PowerLevel:2}}

{{(sweepTxPower)}{PowerLevel:3}}
```

Figure 2.5. Running RAILtest's sweepTxPower Command



CUSTOM_pa_curves_efr32.h

```
#define RAIL_PA_CURVES_2P4_LP  \
  {                            \
    -120, /*! Power Level 1 */ \
    -70,  /*! Power Level 2 */ \
    -25,  /*! Power Level 3 */ \
    -15,  /*! Power Level 4 */ \
    -7,   /*! Power Level 5 */ \
    -2,   /*! Power Level 6 */ \
    0     /*! Power Level 7 */ \
  }
```

Figure 2.6. Updating the deci-dBm Output Powers in the 2.4 GHz low-power PA Lookup Table

2.3 dBm-to-powerSetting Mode

A table in memory that contains a "powerSetting" for each supported dBm level is provided for the EFR32xG25 and SixG301 families. The table specifies the minimum and maximum dBm levels that are supported, as well as the step size between entries in the table. The default step size is 0.1 dBm. This allows for precise control over the output power of the PA, making it possible to achieve the desired level of performance across the entire range of supported dBm levels. To accomplish this, a new mode has been added to the EFR32xG25 family that allows for a single PA to support the entire range of supported dBm levels for both the OFDM and SUBGIG (FSK) PAs. Note that this new mode utilizes "power-Setting" instead of the traditional "power levels" for the output power of the PA. As a result, the capability to set a specific "raw" power level is no longer supported in this mode. Only the setting of dBm levels from the table is supported in this mode. The “powerSetting” table is also provided to the SixG301 family, but there is not a special mode to use it.

2.3.1 OFDM

A powerSetting for OFDM is comprised of three primary components: the digital gain, the slice level, and the filter gain. The digital gain is specified by the VPA_OFDM_GAINDIG_MASK and VPA_OFDM_GAINDIG_SHIFT values, which define a mask and shift for extracting the digital gain from the powerSetting value. The slice level is specified by the VPA_OFDM_SLICE_MASK and VPA_OFDM_SLICE_SHIFT values, which define a mask and shift for extracting the slice level from the powerSetting value. Finally, the filter gain is specified by the VPA_OFDM_FILGAIN_MASK and VPA_OFDM_FILGAIN_SHIFT values, which define a mask and shift for extracting the filter gain from the powerSetting value. These three components work together to determine the overall power output of the PA for OFDM signals.

To ensure proper usage and optimal performance, it is important to adhere to the valid ranges for the digital gain, slice level, and filter gain. The valid range for the digital gain is 0 to 204 (the value is multiplied by 5 in RAIL, resulting in a range of 0 to 1020), the valid range for the slice level is 0 to 191, and the valid range for the filter gain is 0 to 3.

In addition to the valid ranges, the following are the recommended ranges: digital gain between 200 and 800, filter gain of 2.

```
#define VPA_OFDM_GAINDIG_MASK       (0xFFUL)
#define VPA_OFDM_GAINDIG_SHIFT      (0U)
#define VPA_OFDM_SLICE_MASK         (0xFF00UL)
#define VPA_OFDM_SLICE_SHIFT        (8U)
#define VPA_OFDM_FILGAIN_MASK       (0xFF0000UL)
#define VPA_OFDM_FILGAIN_SHIFT      (16U)
```

Figure 2.7. Format of the OFDM PA powerSetting

To generate a table using the pa_dbm_mapping_table_generator.py script (in platform/radio/rail_lib/tools), you will need a CSV file containing the necessary data for OFDM mode. The file should have a header row with column names for:

- Digital gain
- Slice level
- Filter gain

Each subsequent row should contain data for a specific dBm level within the desired range. Each column's data should be separated by a comma, and you can add optional comments for each row using the "#" symbol. It is crucial to ensure that the CSV file is properly formatted, with the correct data in each column and the correct number of rows for the desired dBm range.

Table 2.1. Format of the OFDM PA CSV File

TXTRIMPASLICE	GAINDIG	FILGAIN	#dBm
x	x	x	#-317
x	x	x	#-316
x	x	x	#-315
x	x	x	#-314

This table shows an example of how the CSV file should be formatted for generating a table for OFDM mode. The first row is the header row, which specifies the column names. Each subsequent row contains data on the power settings for a specific dBm level, as well as an optional comment indicating the corresponding dBm value. The data for each column is separated by a comma. It is important to include a row for every dBm level within the desired range, with the appropriate power setting values for each level. Please be aware that the "dBm" column in the table is preceded by a "#" symbol, which indicates that it is a comment and will be ignored by the script.

2.3.2 SUBGIG (FSK)

A powerSetting for SUBGIG (FSK) mode is comprised of four primary components: the fine current setting, the coarse current setting, the slice level, and the stripe setting. The fine current setting is specified by the VPA_SUBGIG_CURRENTFINE_MASK and VPA_SUBGIG_CURRENTFINE_SHIFT values, which define a mask and shift for extracting the fine current setting from the powerSetting value. The coarse current setting is specified by the VPA_SUBGIG_CURRENTCOARSE_MASK and VPA_SUBGIG_CURRENTCOARSE_SHIFT values, which define a mask and shift for extracting the coarse current setting from the powerSetting value. The slice level is specified by the VPA_SUBGIG_SLICE_MASK and VPA_SUBGIG_SLICE_SHIFT values, which define a mask and shift for extracting the slice level from the powerSetting value. Finally, the stripe setting is specified by the VPA_SUBGIG_STRIPE_MASK and VPA_SUBGIG_STRIPE_SHIFT values, which define a mask and shift for extracting the stripe setting from the powerSetting value. These four components work together to determine the overall power output of the PA for SUBGIG (FSK) signals.

The valid range for the fine current setting is 0 to 8, the valid range for the coarse current setting is 0 to 31, the valid range for the slice level is 0 to 2, and the valid range for the stripe setting is 0 to 31.

In addition to the valid ranges, the following are the recommended ranges: current fine of 4, and slice level between 1 and 2.

```
#define VPA_SUBGIG_CURRENTFINE_MASK       (0xFF000000UL)
#define VPA_SUBGIG_CURRENTFINE_SHIFT      (24U)
#define VPA_SUBGIG_CURRENTCOARSE_MASK     (0xFF0000UL)
#define VPA_SUBGIG_CURRENTCOARSE_SHIFT    (16U)
#define VPA_SUBGIG_SLICE_MASK             (0xFF00UL)
#define VPA_SUBGIG_SLICE_SHIFT            (8U)
#define VPA_SUBGIG_STRIPE_MASK            (0xFFUL)
#define VPA_SUBGIG_STRIPE_SHIFT           (0U)
```

Figure 2.8. Format of the SUBGIG (FSK) PA powerSetting

To generate a table using the pa_dbm_mapping_table_generator.py script (in platform/radio/rail_lib/tools), you will need a CSV file containing the necessary data for SUBGIG (FSK) mode. The file should have a header row with column names for:

- Coarse current setting
- Fine current setting
- Slice level
- Stripe setting

Each subsequent row should contain data for a specific dBm level within the desired range. Each column's data should be separated by a comma, and you can add optional comments for each row using the "#" symbol. It is crucial to ensure that the CSV file is properly formatted, with the correct data in each column and the correct number of rows for the desired dBm range.

Table 2.2. Format of the SUBGIG (FSK) PA CSV File

COARSE	FINE	SLICE	STRIPE	#dBm
x	x	x	x	#-317
x	x	x	x	#-316
x	x	x	x	#-315
x	x	x	x	#-314

This table shows an example of how the CSV file should be formatted for generating a table for SUBGIG (FSK) mode. The first row is the header row, which specifies the column names. Each subsequent row contains data on the power settings for a specific dBm level, as well as an optional comment indicating the corresponding dBm value. The data for each column is separated by a comma. It is important to include a row for every dBm level within the desired range, with the appropriate power setting values for each level. Please be aware that the "dBm" column in the table is preceded by a "#" symbol, which indicates that it is a comment and will be ignored by the script.

2.3.3 Common

A common powerSetting which is applicable for SixG301 is comprised of two primary components: the ramplevel and pa sub-mode setting. The ramp level is specified by the VPA_2P4G_RAMPLEVEL_MASK and VPA_2P4G_RAMPLEVEL_SHIFT values, which define a mask and shift for extracting the ramplevel setting from the powerSetting value. The PA sub-mode SLI_RAIL_UTIL_PA_TABLE_SUBMODE_MASK and SLI_RAIL_UTIL_PA_TABLE_SUBMODE_SHIFT values define a mask and shift for extracting the PA sub-mode setting from the powerSetting value. The ramplevel component work determines the overall power output of the PA for SixG301.

The valid range for the ramp level setting is 0 to 95.

In addition to the valid range, the following are the recommended ranges:

```
#define VPA_2P4G_RAMPLEVEL_MAX                (95U)
#define VPA_2P4G_RAMPLEVEL_MIN                (0U)
#define VPA_2P4G_RAMPLEVEL_MASK               (0xFFUL)
#define VPA_2P4G_RAMPLEVEL_SHIFT              (0U)
#define SLI_RAIL_UTIL_PA_TABLE_SUBMODE_MASK   (0xC000UL)
#define SLI_RAIL_UTIL_PA_TABLE_SUBMODE_SHIFT  (14U)
```

Figure 2.9. Format of the Common PA powerSetting

To generate a table using the pa_dbm_mapping_table_generator.py script (in platform/radio/rail_lib/tools), you will need a CSV file containing the necessary data for Common powersetting. The --type input from the choices 10dBm or 0dBm is needed for the script. Generated output will also have that appended along with the ramplevel. The CSV file should have a header row with column names for:

- Ramp Level

Each subsequent row should contain data for a specific dBm level within the desired range. Each column's data should be separated by a comma; you can add optional comments for each row using the "#" symbol. It is crucial that the CSV file is properly formatted, with the correct data in each column and the correct number of rows for the desired dBm range.

Table 2.3. Format of the Common(FSK) PA CSV File

RAMPLEV	#dBm
x	#-317
x	#-316
x	#-315
x	#-314

This table shows an example of how the CSV file should be formatted for generating a table for Common mode for SixG301. The first row is the header row, which specifies the column names. Each subsequent row contains data on the power settings for a specific dBm level, as well as an optional comment indicating the corresponding dBm value. The data for each column is separated by a comma. It is important to include a row for every dBm level within the desired range, with the appropriate power setting values for each level. Note that the "dBm" column in the table is preceded by a "#" symbol, which indicates that it is a comment and will be ignored by the script.

2.3.4 Generating the Table

The script **pa_dbm_mapping_table_generator.py** is located in the `platform/radio/rail_lib/tools` directory. It takes data from the CSV file specified with the `File` argument and generates a list of power setting values based on the `-min` or `--minPower` and `-max` or `--maxPower` arguments. To understand the CSV format for each respective mode, refer to sections 2.3.1 OFDM and 2.3.2 SUBGIG (FSK). Note that a `-t` or `--type` argument is also required, with a default value of `OFDM`, to specify the type of curve.

It also accepts command-line arguments for:

- An optional flag `-f` or `--fem` to indicate whether the table is for a front-end module (FEM) with a default value of False.
- An optional argument `-o` or `--output` to specify the output file for the generated code.

The script has a default increment value for the dBm levels, which you can customize as needed. Additionally, the script generates C code for defining the table, including:

- The number of values
- The step size between dBm levels
- The minimum and maximum dBm levels
- The actual power setting values

Finally, the script outputs the generated C code to `stdout`. To use the generated table, the user should replace the applicable contents of the plugin/pa-conversions/efr32xg25/ files with the output of the script.

3. Appendix: Alternative Methods to Convert between Power Levels and dBm Output Power

For your application, having highly granular dBm output levels may not be necessary. Therefore, it may make sense to overwrite the `RAIL_ConvertRawToDbm` and `RAIL_ConvertDbmToRaw` functions that Silicon Labs provides. This example demonstrates how to simplify those functions to return only a few values of interest. Doing this would also allow you to remove calls to the `RAIL_InitTxPowerCurves` function and `RAIL_DECLARE_TX_POWER_XXXX_CURVES` macro. Removing these dependencies on the piecewise linear curve fits allows you to reduce application code size significantly. In this example, the functions are meant to work with common dBm values, such as 20, 13, 3, 0 and -10 (200, 130, 30, 0 and -100 deci-dBm, respectively).

```
RAIL_TxPowerLevel_t RAIL_ConvertDbmToRaw(RAIL_Handle_t railHandle,
                                         RAIL_TxPowerMode_t mode,
                                         RAIL_TxPower_t power)
{
  // Anything above this power is likely still an error, so it
  // should return an invalid value
  if (power >= RAIL_TX_POWER_MAX) {
    return RAIL_TX_POWER_LEVEL_INVALID;
  }

  if (mode == RAIL_TX_POWER_MODE_2P4_LP) {
    if (power >= 0) {
      return RAIL_TX_POWER_LEVEL_LP_MAX;
    } else if (power >= -100) {
      return 2;
    } else {
      return 1;
    }
  }

  if (mode == RAIL_TX_POWER_MODE_2P4_HP) {
    if (power >= 200) {
      return RAIL_TX_POWER_LEVEL_HP_MAX;
    } else if (power >= 130) {
      return 93;
    } else if (power >= 30) {
      return 33;
    } else if (power >= 0) {
      return 22;
    } else if (power >= -100) {
      return 7;
    } else {
      return 1;
    }
  }

  if (mode == RAIL_TX_POWER_MODE_SUBGIG) {
    if (power >= 200) {
      return RAIL_TX_POWER_LEVEL_SUBGIG_MAX;
    } else if (power >= 130) {
      return 86;
    } else if (power >= 30) {
      return 26;
    } else if (power >= 0) {
      return 18;
    } else if (power >= -100) {
      return 6;
    } else {
      return 1;
    }
  }

  // return an error value if no PA is initialized
  return RAIL_TX_POWER_MIN;

}

RAIL_TxPower_t RAIL_ConvertRawToDbm(RAIL_Handle_t railHandle,
                                    RAIL_TxPowerMode_t mode,
                                    RAIL_TxPowerLevel_t powerLevel)
{
  if (mode == RAIL_TX_POWER_MODE_2P4_LP) {
    if (powerLevel >= RAIL_TX_POWER_LEVEL_LP_MAX) {
      return 0;
    } else if (powerLevel >= 2) {
      return -100;
    } else {
      return 1;
    }
  }

  if (mode == RAIL_TX_POWER_MODE_2P4_HP) {
    if (powerLevel >= RAIL_TX_POWER_LEVEL_HP_MAX) {
      return 200;
    } else if (powerLevel >= 93) {
      return 130;
    } else if (powerLevel >= 33) {
      return 30;
    } else if (powerLevel >=22) {
      return 0;
    } else if (powerLevel >= -7) {
      return -100;
    } else {
      return -250;
    }
  }

  if (mode == RAIL_TX_POWER_MODE_SUBGIG) {
    if (powerLevel >= RAIL_TX_POWER_LEVEL_SUBGIG_MAX) {
      return 200;
    } else if (powerLevel >= 86) {
      return 130;
    } else if (powerLevel >= 26) {
      return 30;
    } else if (powerLevel >= 18) {
      return 0;
    } else if (powerLevel >= 6) {
      return -100;
    } else {
      return 1;
    }
  }

  // return an error value if no PA is initialized
  return RAIL_TX_POWER_MIN;

}
```

IoT Portfolio
www.silabs.com/products

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

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

www.silabs.com