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

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.

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
  - A low-power 2.4 GHz PA (for power 0 dBm and lower)
  - A Sub-GHz PA
- EFR32xG2x
  - A high-power 2.4 GHz PA
  - 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)

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
- EFR32xG2x
  - High-power 2.4 Ghz: 1-180
  - Medium-power 2.4 Ghz: 1-90
  - Low-power 2.4 Ghz: 1-64

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 look-up 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, and on EFR32xG2x chips, there is a curve for each PA. For these PAs, this method provides a good balance between accuracy across power levels, computational speed, and code size. The conversion for EFR32xG1x’s low-power 2.4GHz, which has only seven 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 other words, it assumes that the existing methodologies found in `RAIL_ConvertDbmToRaw` and `RAIL_ConvertRawToDbm` are appropriate for your application and will not be overwritten, 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 Flex SDK version 2.4.0 or higher, even if you are developing in an earlier 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 (EFR32xG1x High-Power 2.4 GHz and Sub-GHz, EFR32xG2x All PAs), 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 (EFR32xG1x High-Power 2.4 GHz and Sub-GHz, EFR32xG2x All PAs)

The following steps explain how to generate the piecewise fit curves for the high-power 2.4GHz PA and sub-GHz PA. Please see section 2.2 Lookup Table Fits (EFR32xG1x Low-Power 2.4 GHz) if you also intend to characterize that PA.

1. In Simplicity Studio, generate a RAILtest application configured with the desired PA, PA power source, and PA ramp time (specified through the Hardware Configurator). See *AN1115: Configuring Peripherals for 32-Bit Devices in Simplicity Studio* for more details on the configuration process. Build it, 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.

```
> 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** 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.

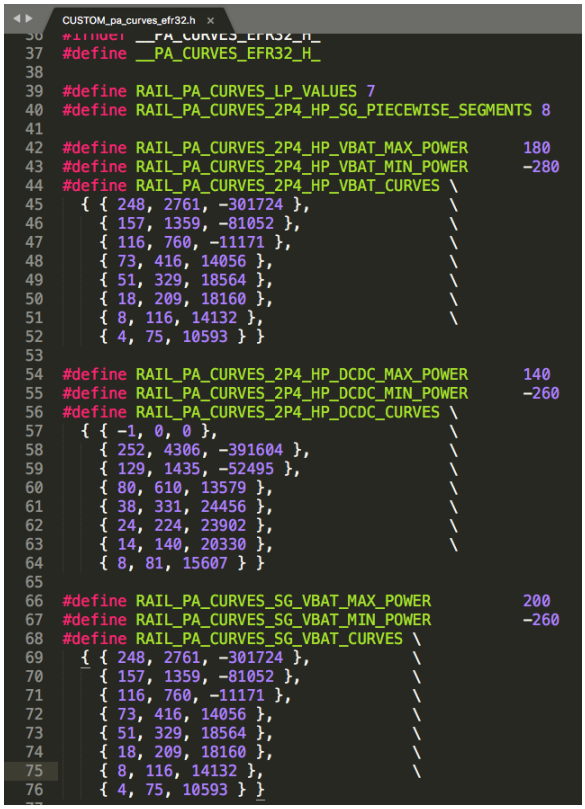


Figure 2.2. Terminal Output Copied into a Custom pa_curves_efr32xg1x.h File

9. Update the hardware config `HAL_PA_CURVE_HEADER` header to point to your new file. The following figure illustrates where to update the header.

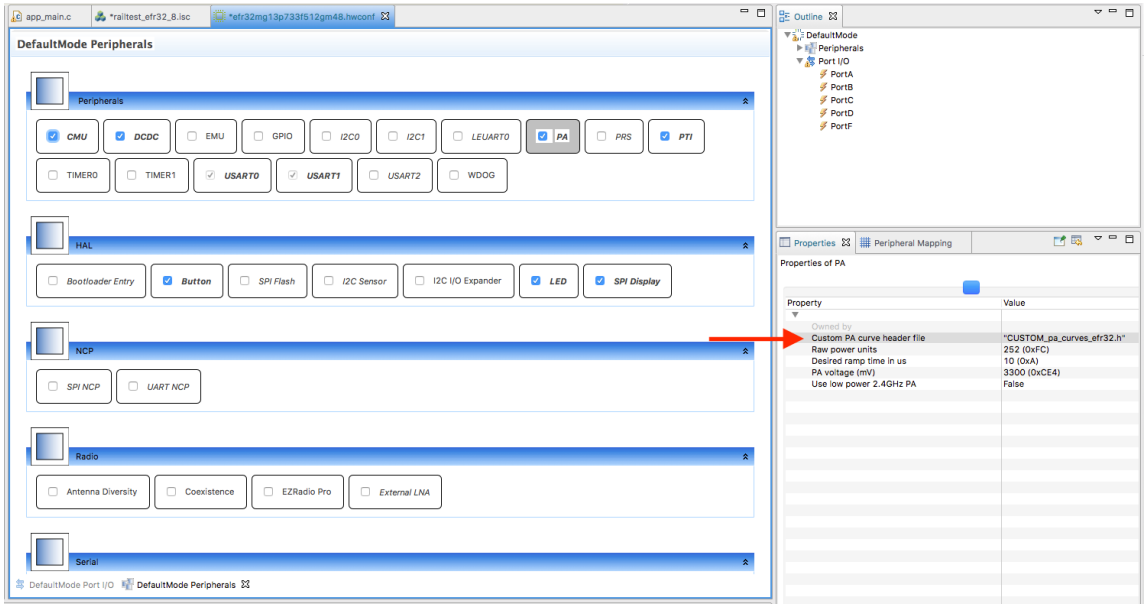


Figure 2.3. Updating the Custom PA Curve Header File Name in Simplicity Studio

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 (EFR32xG1x Low-Power 2.4 GHz)

Since it only has seven power levels, the low-power 2.4 GHz PA uses a simple array lookup table, as opposed to a piecewise linear curve fit. Create a copy of the PA curves file. 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). 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 2.4. 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 2.5. Updating the deci-dBm Output Powers in the 2.4 GHz low-power PA Lookup Table

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) {
```

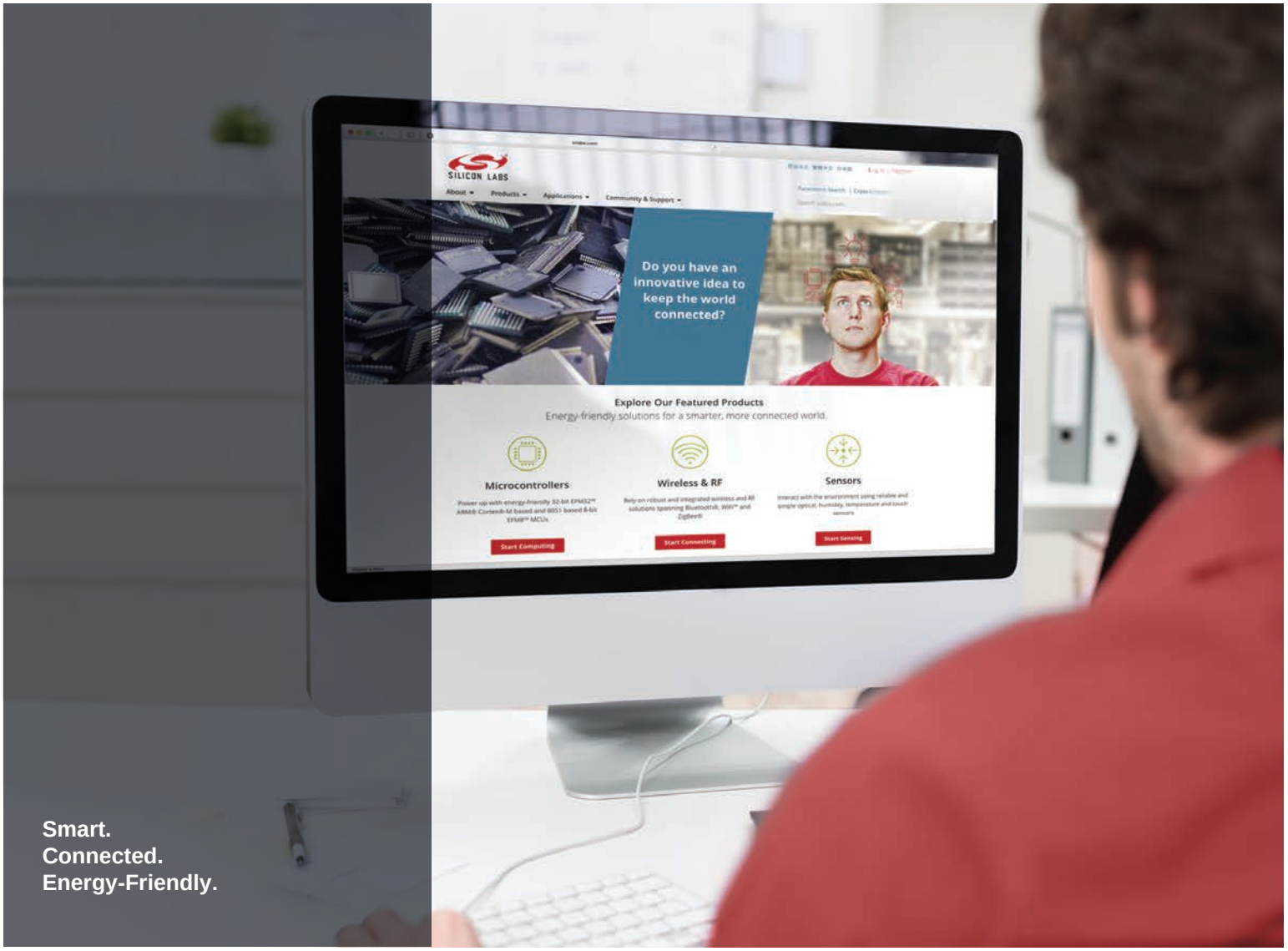
```
    if (powerLevel >= RAIL_TX_POWER_LEVEL_LP_MAX) {
      return 0;
    } else if (powerLevel >= 2) {
      return -100;
    } else {
      return 1;
    }
  }

  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;

}
```



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

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

http://www.silabs.com