AN1245: EFP01 Configuration Tool Guide

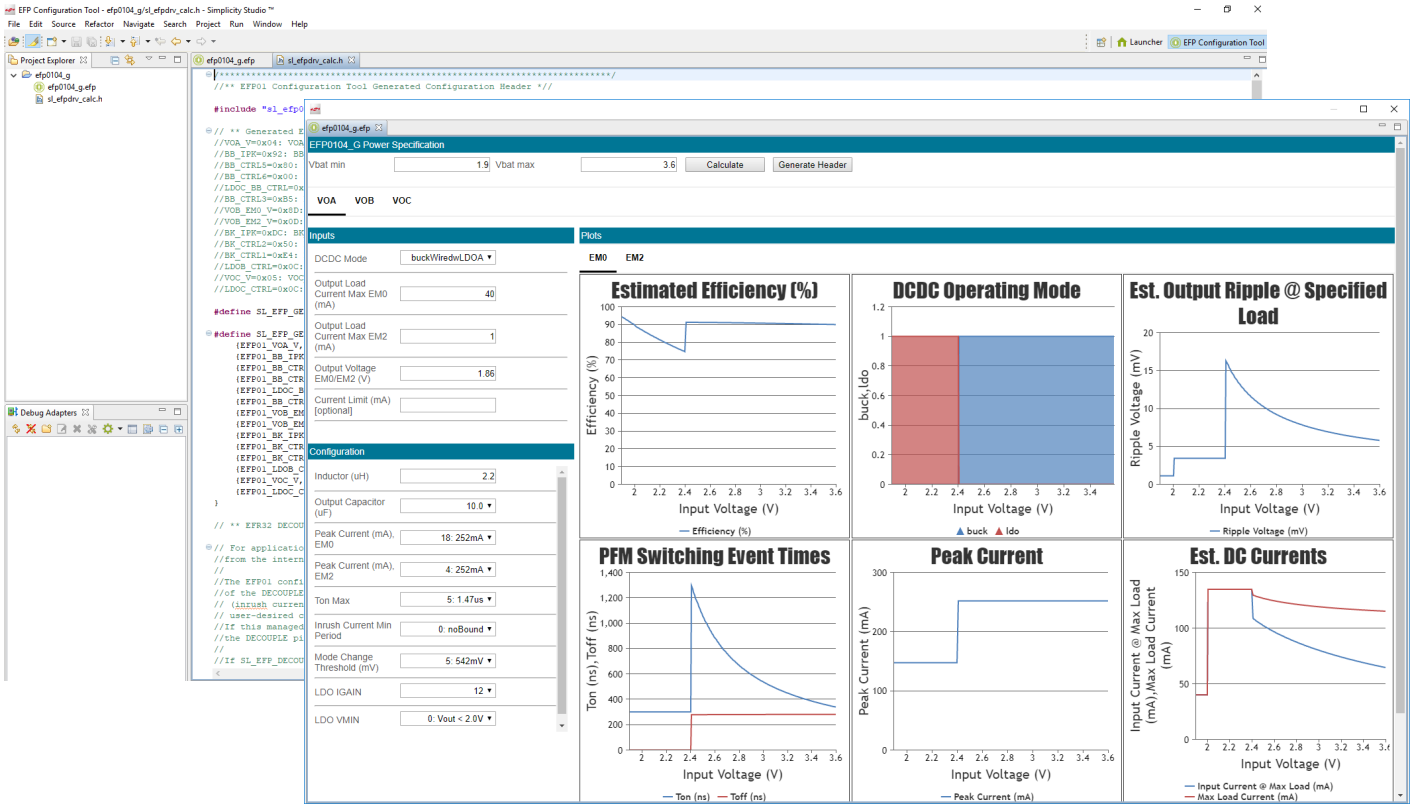
The EFP01 family energy friendly PMIC devices are extremely flexible, highly efficient, multi-output power management IC, providing complete system power and primary cell battery Coulomb counting for EFM32 and EFR32 devices

The EFP01 Configuration Tool (EFP Configurator) determines the best component values and connections given design constraints, caculates expected performance, and auto-generates configuration code for EFP01 Configuration Tool devices.

This document discusses how to use the EFP01 Configuration Tool as part of an EFx32 project.

KEY POINTS

- EFP01 expands the input supply range of EFR32-based systems.
- EFP01 reduces power consumption of systems that use EFR32 devices without an integrated DCDC converter.
- EFP0111 enables EFR32's radio to operate at higher output power using coin cell batteries.
- Use the EFP01 Configuration Tool to configure EFP01 devices.
- Use the EFP01 Configuration Tool to evaluate performance and optimize EFP01 device operation.



1. Getting Started

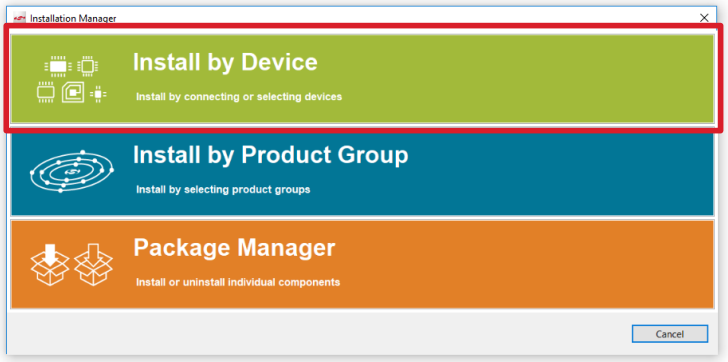
Required Software

Simplicity Studio is the development environment for EFP01 projects. Download the latest version of Studio at:

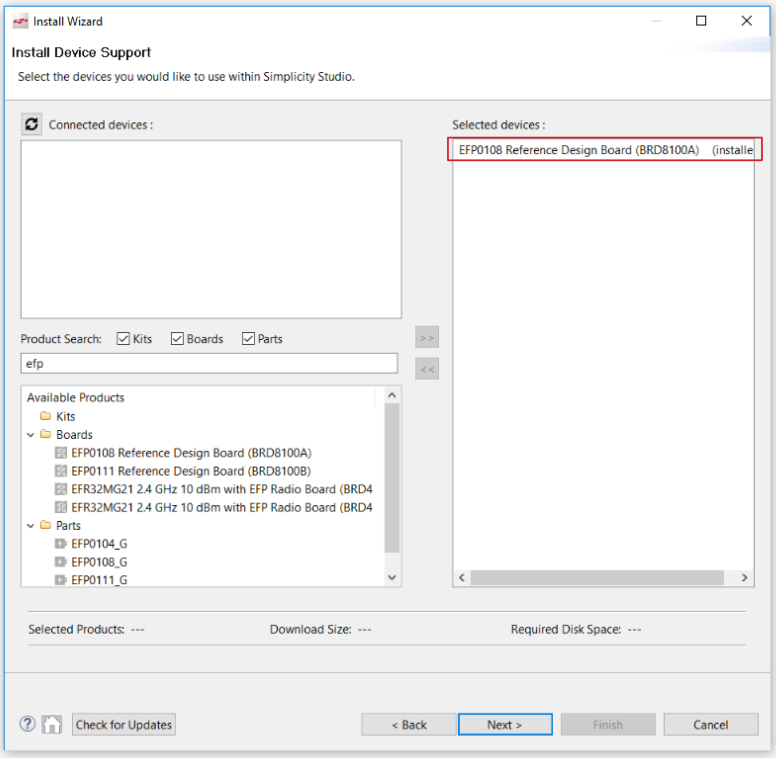
https://www.silabs.com/products/development-tools/software/simplicity-studio

Installation and Setup

1. Download and install the latest Simplicity Studio version.
2. In Simplicity Studio, navigate to [**Help->Update Software**].



3. Select [**Install by Device**] and in the [**Product Search**] enter the name of the desired EFP01 device and EFx32 Gecko device. Click [**Next**] and install all required components. For more detail on Simplicity Studio updates, configuration, and tool support, see *AN0822: Simplicity Studio User's Guide*



.

4. Create an EFP01 Configurator project. To do this navigate to [**File**]>[**New**]>[**Project**] and select [**EFP Configurator Project**]. Select your EFP01 device from the parts list, and click through to generate the project.

2. EFP01 Configuration Tool Overview

The EFP01 Configuration Tool allows a system designer to configure and optimize the EFP01 register settings for a specific application. There are two supported use-cases for the EFP01 Configuration Tool:

1. For all orderable part numbers (OPNs), the EFP01 Configuration Tool can generate a header file containing EFP01 register settings that can be included in an EFR32 or EFM32 Studio project. See 4. EFP01 Generated Header File for more details on the header file content and structure.
2. For certain OPNs with a corresponding Evaluation Kit (EFP0108 and EFP0111), the EFP01 Configuration Tool can directly interface to the Evaluation Kit on-board EFP01 and host EFR32 devices. The EFP01 Configuration Tool can write register settings to either the EFP01 device alone or to both the EFP01 device and the host EFR32 device flash. See ● Write to Device (for EFP0108 and EFP0111) on page 4 and ● Store to Flash (for EFP0108 and EFP0111) on page 5 for more details.

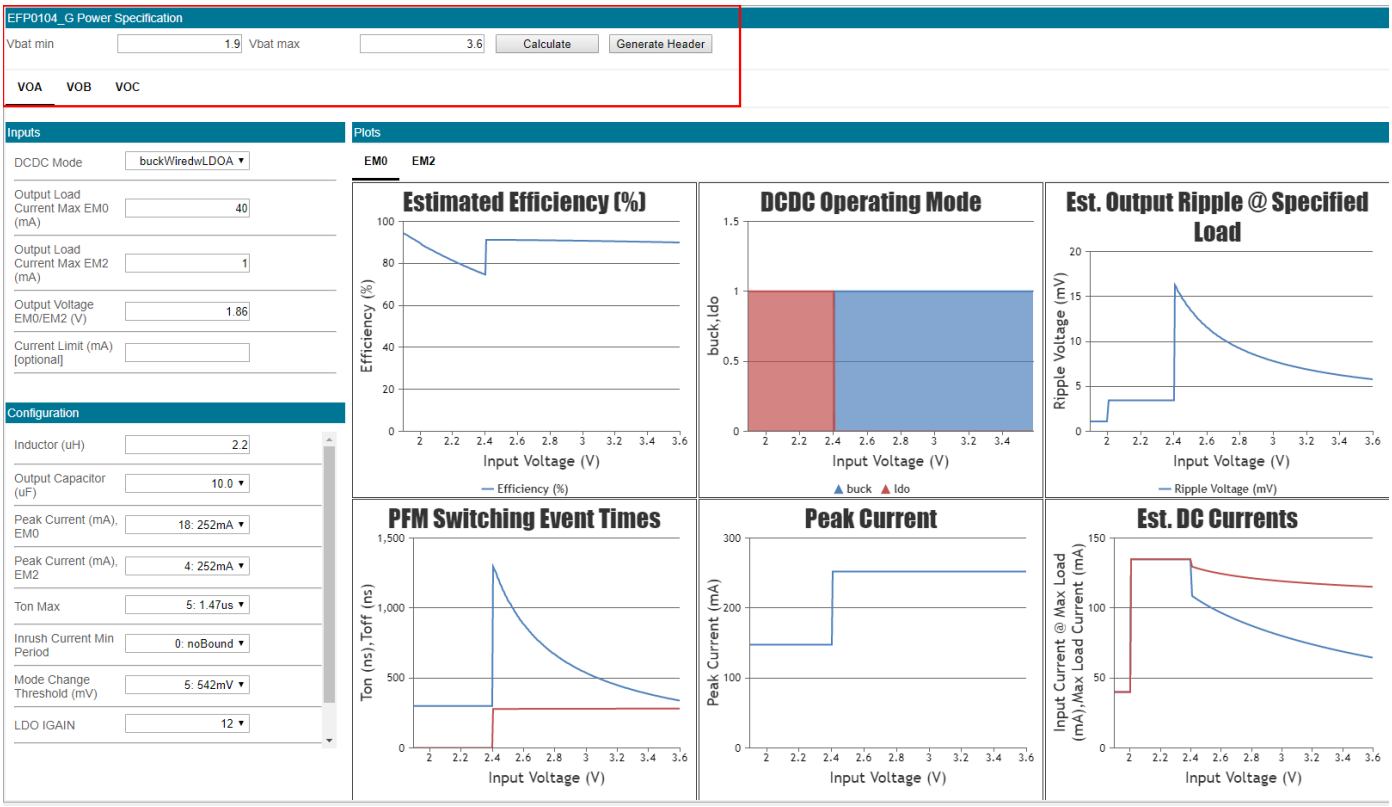
Note: the EFP01 Configuration Tool itself does not modify any of the settings in the [**Inputs**] or [**Configuration**] sections directly. Instead, after every run of the tool (initiated on any value change or by clicking the [**Calculate**] button), a pop-up window will appear with recommended changes to the settings based off the current inputs. The user is expected to implement any change in the [**Configuration**] themselves.

Note: the EFP01 Configuration Tool is intended to simplify configuration of the EFP01's DCDC converters and LDOs in EM0 and EM2 mode. It is not a comprehensive EFP01 configuration utility - as such, certain EFP01 features are not configurable through this tool and should be implemented in customer C code. Specific EFP01 features that are not included in the EFP01 Configuration Tool include :

- Coulomb counter
- Interrupt masking / unmasking
- ADC readings
- EM4 coarse regulator configuration
- Energy mode transition method (I2C vs Direct Mode)
- VOA_SW state

See *AN1187: EFP01 Design Considerations* and the *EFP01 Energy Friendly PMIC Family Data Sheet* for more technical details of different parameters, modes of operations, and formulas relevant to EFP01 devices.

2.1 Power Specification



The power specification section defines the expected battery voltage range, configured by the [**Vbat min**] and [**Vbat max**] fields. These should be set based on the type of battery used in the application.

The [**VOA**], [**VOB**], and [**VOC**] tabs toggle between the configuration options for each of the EFP01 outputs. For the most part, these outputs are configured independently, but share source generation. One exception is for the EFP0104 OPN, in which the VOA DCDC A output is shorted together to the VOC LDO C output on the PCB for a "Buck with LDO in parallel" configuration. For the EFP0104 VOA/VOC output, the [**VOA**] tab configures the DCDC A and the [**VOC**] tab configures the LDO C.

Calculate

Performs a recalculation. Note that a value change to any of the inputs will typically result in an automatic recalculation, so use of this button is typically unnecessary. A recalculation will redisplay the pop-up window containing any errors, warnings, or information, as well as recommended configuration changes.

Generate Header

Generates a `sl_efpdrv_calc.h` header file in the project directory containing the current EFP01 configuration register settings.

Note: The following two button options appear only for the EFP0108 and EFP0111 OPNs, and are only supported if the corresponding Evaluation Kit (i.e., EFP0108 Evaluation Kit BRD8100A or EFP0111 Evaluation Kit BRD8100B) are attached via USB. Those Evaluation Kits incorporate an on-board EFR32TG11 device acting as an I2C host interfacing to the on-board EFP01 device. The EFR32TG11 device has two mechanisms for writing register configuration to the EFP01 device:

1. Via USB download while attached to the EFP01 Configuration Tool
2. From the EFR32TG11 flash at startup or on EFR32TG11 reset

Write to Device (for EFP0108 and EFP0111)

This button will download the current configuration to the EFP01 on the Evaluation Board. The configuration will *not* be stored on the Evaluation Board anywhere except in the EFP01 device's internal registers. In the event of a EFP01 device power cycle or a EFR32TG11 reset, this downloaded configuration will be lost.

Store to Flash (for EFP0108 and EFP0111)

This button will cause the current configuration to be written to the EFP01 device registers as well as the EFR32TG11 device flash memory on the Evaluation Board. The configuration stored in the EFR32TG11 flash will be re-loaded into the onboard EFP01 device registers whenever the Evaluation Board EFR32TG11 is reset (e.g., when the EFR32TG11 power is cycled by disconnecting / reconnecting the USB cable). Note that simply power cycling the EFP01 device input supply will *not* cause a reload of the register configuration from the EFR32TG11 flash.

2.2 Inputs

Fields in the [**Inputs**] section should be determined based on the intended application and use case for the EFP01 device.

Note: the EFP01 Configuration Tool itself does not modify any of the settings in the [**Inputs**] or [**Configuration**] sections directly. Instead, after every run of the tool (initiated on any value change or by clicking the [**Calculate**] button), a pop-up window will appear with recommended changes to the settings based off the current inputs. The user is expected to implement any change in the [**Configuration**] themselves.

DCDC Mode

Indicates what mode the selected output is to be configured for. Different modes of EFP01 operation require different external connections, see *Section 4. Typical Connection Diagrams* of the *EFP01 Energy Friendly PMIC Family Data Sheet* for more details on each mode and its intended use case.

Output Load Current Max EM0

Indicates the expected maximum load on the EFP01 device output when the device is in EM0 - Active/Run Mode. This is the highest expected current consumption of everything powered by the EFP01 device.

Output Load Current Max EM2

Indicates the expected maximum load on the EFP01 device output when the device is in EM2 - Deep Sleep Mode. For more information on the EFP01 energy modes see *Section 3.3 Energy Modes* of the device datasheet.

Output Voltage EM0/EM2

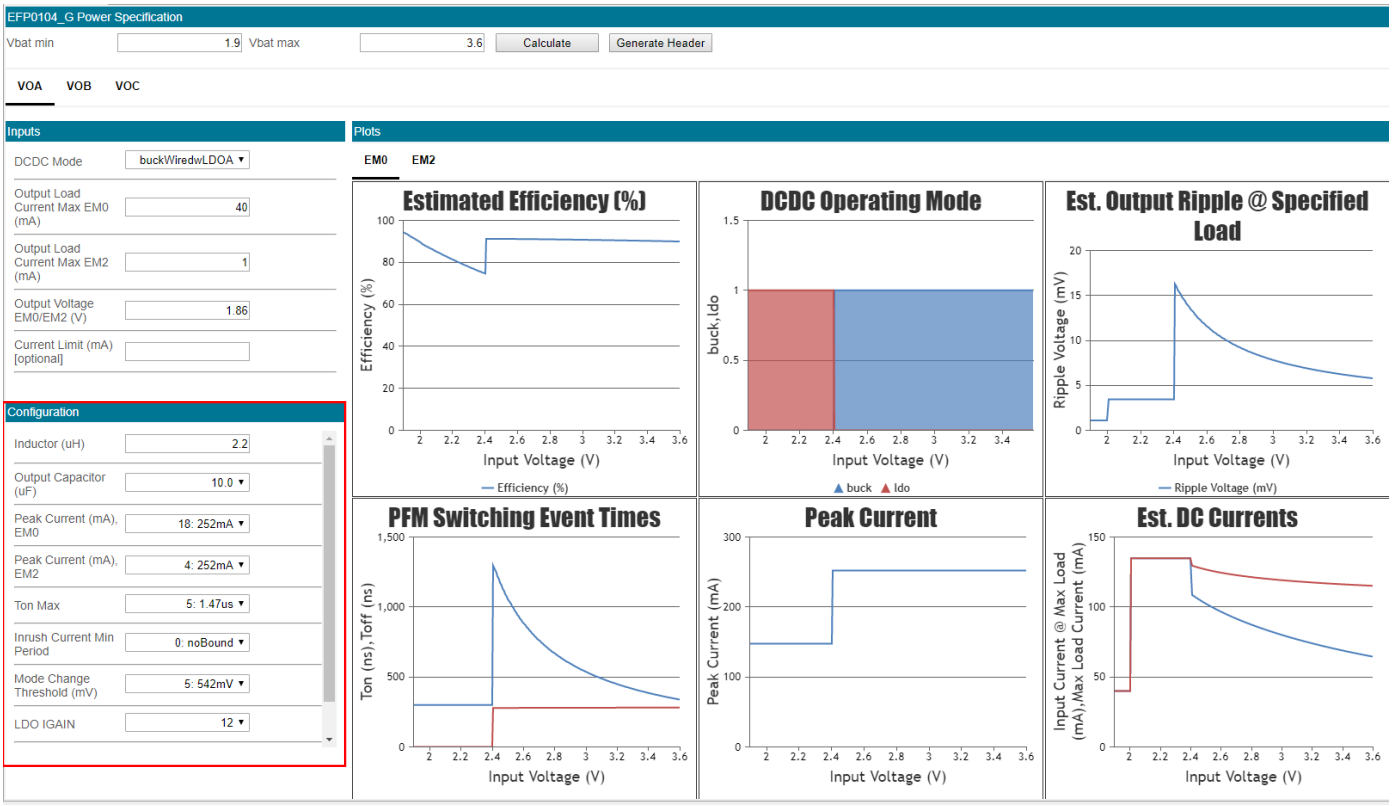
The desired output voltage level for the selected output in the specified energy mode

Battery Current Limit

Battery Current Limit should be defined for weak or high internal impedance power sources. More detail is found in *Section 3.2.1.4 Current Limiting* of the device data sheet.

Entering a battery current limit in mA into this field will cause a suggested Inrush Current Min Period setting to appear in the post-calculation pop-up. The user can then adjust the Inrush Current Min Period based off the suggestion. Leave this field blank if no current limit is desired.

2.3 Configuration



Fields in the [**Configuration**] section have default values suitable for many applications. These settings may need to be modified. Carefully review *AN1187: EFP Hardware Design Considerations* and the *EFP01 Energy Friendly PMIC Family DataSheet* before modifying these values.

When a new EFP01 Configuration Tool project is created, the default values in the [**Configuration**] section will generally match the EFP01 device programmed OTP default settings. There are a few exceptions, however. For example, the VOB converter is disabled by default on certain OPNs. However, most of the VOB converter [**Configuration**] settings in the EFP01 Configuration Tool will default to useable values, so that the the user has only to change the [**DCDC Mode**] setting to "Enabled" to get a usable configuration.

Note: the EFP01 Configuration Tool itself does not modify any of the settings in the [**Inputs**] or [**Configuration**] sections directly. Instead, after every run of the tool (initiated on any value change or by clicking the [**Calculate**] button), a pop-up window will appear with recommended changes to the settings based off the current inputs. The user is expected to implement any change in the [**Configuration**] themselves.

Configuration Fields

The [**Inductor**] and [**Output Capacitor**] control fields refer to external component values to the EFP01 device. Under most conditions these should be left to the default value.

Note: In some conditions the default 2.2 uH inductor value will be incorrect. See *Table 5.5 VOA Output* and *Table 5.6 VOB Output* of the device datasheet for details on the external inductor value.

The other fields represent programmable EFP01 register values, and are detailed in *Section 3. System Overview* of the device datasheet.

2.4 Plots

The EFP01 Configuration Tool can also plot estimated device performance over the input suply voltage range.

Note: The EFP01 Configuration Tool output plots represent estimates based on simulated EFP01 behavior, and are not a guarantee of device performance.

Both the [**VOA**] and [**VOB**] tabs have [**Plots**] section, which displays estimated EFP01 performance metrics based off the given input parameters. Each [**Plots**] section has [**EM0**] and [**EM2**] tabs controlling which energy mode is displayed.

Note that the [**VOC**] output does not have its own [**Plots**] section - the VOC Converter output is either an independent LDO (e.g., EFP0108/EFP0109/EFP0111) with no plot data, or it is used in conjunction with the VOA output (e.g., EFP0104), in which case the VOC LDO output data is included in the VOA [**Plots**] section.

Also note that for some EFP01 devices (e.g, EFP0108 and EFP0111), the VOB converter input supply is expected to be connected to the VOA converter output. Since the VOA converter output is expected to be a constant voltage, the [**VOB**] output plots for these devices will show only a single data point at that VOA output voltage.

Estimated Efficiency

Displays the estimated efficiency (including quiescent current) for the selected output across the battery voltage range. High efficiency will directly improve battery life.

DCDC Operating Mode

Displays the converter operating mode as it varies with the input supply voltage. For EFP01 converters that support operation in either DCDC and LDO modes (e.g., the VOB converter for all devices and the VOA converter for EFP0104), this plot will display the input voltage crossover point from "Buck" to "LDO" operation (the operating mode crossover voltage is controlled by the [**Mode Change Threshold**] input parameter). Note that some EFP01 devices will only ever operate in a single mode - for example, the VOA converter on the EFP0108, EFP0109, and EFP0111 devices will always operate in "Boost".

Est. Output Ripple @ Specified Load

Estimates the worst case output ripple voltage versus input supply voltage for the selected output. The ripple voltage estimate assumes the load current is equal to the value entered in the [**Output Load Current Max (mA)**] input.

PFM Switching Event Times

Estimates the duration of the on-time and off-time switching events of the PFM converter versus the input supply voltage.

Peak Current

When theEFP01 converter is operating in a DCDC mode (e.g, "Buck" or "Boost"), this plot displays the estimated inductor peak current versus the supply input voltage. The inductor peak current is configured by the [**Peak Current**] input parameters. This value is controlled during DCDC operation via fields in the [**Configuration**] section. Note that on certain outputs, adaptive peak current control is possible for advanced use cases. See *Section 3.2.1.3 Peak Current Adjustment* in the device datasheet.

For EFP01 converters that support operation in either DCDC and LDO modes (e.g., the VOB converter for all devices and the VOA converter for EFP0104), when operating in LDO modes this plot displays the estimated LDO current versus the supply input voltage.

The system designer should ensure that the selected inductor saturation current exceeds the displayed peak current (allowing for some margin).

Est. DC Currents

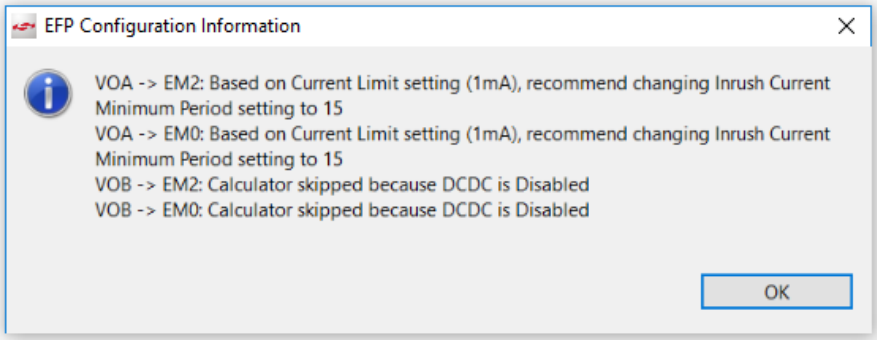
When the converter is operating in DCDC or LDO mode, this plot displays the estimated the maximum allowable load current versus input supply voltage for the given settings, as well as the estimated input supply current for that given maximum load.

Converters operating in LDO mode will go into Bypass mode (i.e., the input is directly shorted to the output) as the input supply voltage approaches the output voltage and the LDO can no longer regulate. When the converter is in Bypass mode, this plot will simply display the value entered in the [**Output Load Current Max (mA)**].

3. EFP01 Configuration Tool Workflow

This section describes the workflow using the EFP01 Configuration Tool.

1. Consult the *EFP01 Energy Friendly PMIC Family Data Sheet* to understand which connections and hardware configuration are needed for the application.
2. Enter the expected voltage range of the power source to the EFP01 device in the [**Vbat min**] and [**Vbat max**] fields.
3. Enter the planned DCDC mode and other inputs based on the datasheet descriptions and expected application parameters. See 2. EFP01 Configuration Tool Overview for more detail on each available option.
4. The configuration tool will automatically calculate each new input parameter. If any metrics fall out of specification, the tool will throw an error and recommend changes to either the [**Inputs**] or the [**Configuration**] fields. Make changes to the recommended fields until no errors show.



5. Observe the [**Plots**] performance graphs for both [**EM0**] and [**EM2**]. Confirm that metrics such as maximum output load current, peak current, efficiency, and output ripple voltage are compatible with the application.
6. If multiple outputs are used, repeat this process for each output ([**VOA**], [**VOB**], and [**VOC**]) as needed. Note that the outputs may have different properties and options which are defined in the device datasheet.
7. Once the configuration for each output satisfies the application requirements, click [**Generate Header**] to generate a `sl_efpdrv_calc.h` header file with the current configuration (refer to 4. EFP01 Generated Header File for more information on the header file contents). The generated `sl_efpdrv_calc.h` header file must be included into the EFR32 project for use by the EFP01 driver during initialization.

   Driver documentation can be found for any EFx32 device at https://docs.silabs.com/#section-mcu-wireless. Select the desired EFR32 device and navigate to [**External Device Drivers**]>[**EFP Driver**]. For more information on using the EFP01 driver, see the documentation on docs.silabs.com and in *AN1187: EFP01 Design Considerations*.

4. EFP01 Generated Header File

The `sl_efpdrv_calc.h` generated header file consists of two sections: a set of register address / data pairs and an optional array of data used for the case when the EFP01 VOB output powers the EFR32 device DECOUPLE input.

Generated Header File: EFP01 Address / Data Pairs

The first half of the generated header file contains the following #defines:

- SL_EFP_GEN: an array containing register address and data pairs (e.g., { {ADD1, DATA1}, {ADD2, DATA2}, etc})
- SL_EFP_GEN_SIZE: a constant equal to the count of all the elements in the SL_EFP_GEN array

The `sl_efp_init()` initialization function will interate through the SL_EFP_GEN array to write each data byte to the corresponding address.

A few notes on the SL_EFP_GEN array:

- The address / data pairs in the SL_EFP_GEN array are listed in a specific order, such that the essential configuration for each converter (target voltage, peak currents, etc) is written before the converter is enabled.
- The header file comments preceding the SL_EFP_GEN array provide:
  1. A user-readable listing of each bitfield setting for the current configuration
  2. A list of changes between the current configuration and the original OTP defaults for that particular EFP01 device.
- When the EFP01 VOB output powers the EFR32 device's DECOUPLE input, the EFP01_BK_CTRL1 address will be removed from the SL_EFP_GEN array to prevent enabling the VOB converter. For this case, the VOB converter will be enabled during the DECOUPLE Handoff sequence, described below.

```
//** Generated EFP01 General Configuration Settings **//
//        VOA_V=0x04: VOA_V=4
//        BB_IPK=0x92: BB_IPK_EM2=4, BB_IPK=18
//        BB_CTRL5=0x80: BB_DRVR_SPEED=2, BB_IPK_BOOST_ADJ=0
//        BB_CTRL6=0x00: BB_IPK_NOADJ=0, SW_FAST=0, BB_IRI_CON=0, BB_TOFF_MAX=0
//        LDOC_BB_CTRL=0x50: SEQ_BB_FIRST=0, BB_TON_MAX=5, VOC_IRI_CON=0
//        BB_CTRL3=0xB5: NTM_LDO_THRSH=5, NTM_DUR=2, BB_MODE=5
//        VOB_EM0_V=0x8D: OOR_DIS=1, VOB_EM0_V=13
//        VOB_EM2_V=0x0D: VOB_EM2_V=13
//        BK_IPK=0x2A: BK_IPK_EM2=1, BK_IPK=10
//        BK_CTRL2=0x50: BK_RES_TON_ONLY=0, BK_LDO_THRESH=5, BK_IRI_CON=0
//        LDOB_CTRL=0x0C: LDO_NO_AUTO_BYP=0, LDOB_BYP=0, LDOB_VMIN=0, LDOB_IGAIN=12
//        VOC_V=0x05: VOC_V=5
//        LDOC_CTRL=0x0C: LDOC_ENA_SA=0, LDOC_BYP=0, LDOC_VMIN=0, LDOC_IGAIN=12

//** Configuration Changes from Original (i.e., OTP Default) Configuration **//
//        BB_CTRL6.BB_TOFF_MAX value changed from 3 to 0
//        VOB_EM0_V.VOB_EM0_V value changed from 0 to 13
//        VOB_EM2_V.VOB_EM2_V value changed from 0 to 13
//        BK_IPK.BK_IPK value changed from 0 to 10
//        BK_IPK.BK_IPK_EM2 value changed from 0 to 1
//        LDOB_CTRL.LDOB_IGAIN value changed from 0 to 12
//        VOC_V.VOC_V value changed from 0 to 5

#define SL_EFP_GEN_SIZE 13

#define SL_EFP_GEN {    \
    {EFP01_VOA_V, 0x04},    \
    {EFP01_BB_IPK, 0x92},    \
    {EFP01_BB_CTRL5, 0x80},    \
    {EFP01_BB_CTRL6, 0x00},    \
    {EFP01_LDOC_BB_CTRL, 0x50},    \
    {EFP01_BB_CTRL3, 0xB5},    \
    {EFP01_VOB_EM0_V, 0x8D},    \
    {EFP01_VOB_EM2_V, 0x0D},    \
    {EFP01_BK_IPK, 0x2A},    \
    {EFP01_BK_CTRL2, 0x50},    \
    {EFP01_LDOB_CTRL, 0x0C},    \
    {EFP01_VOC_V, 0x05},    \
    {EFP01_LDOC_CTRL, 0x0C},    \
}
```

Generated Header File: DECOUPLE Handoff

Note: The sl_efp_decouple_handoff() routine and the header file parsing discussed below may not be implemented in the EFP01 drivers until later in 2020. If that routine is not present in the drivers, it can be manually implemented by copying the code in 5. EFP01 Decouple Handoff Routine into the project. This sl_efp_decouple_handoff() routine should be called immediately following EFP01 initialization (i.e., el_efp_init()) whenever EFP01 is powering the EFR32 DECOUPLE pin.

The second half of the generated header file contains the following #defines:

- SL_DECOUPLE_HANDOFF_ARGS: an array containing the following bitfield settings { BK_IRI_CON, BK_TON_MAX, BK_IPK}
- SL_EFP_DECOUPLE_HANDOFF_ARGS_SIZE: a constant equal to the count of all the elements in the SL_DECOUPLE_HANDOFF_ARGS array

If SL_EFP_DECOUPLE_HANDOFF_ARGS_SIZE is > 0, the `sl_efp_init()` initialization function will call the `sl_efp_decouple_handoff()` function.

```
// ** EFR32 DECOUPLE Handoff Sequence Configuration Settings ** //

// For applications where EFP01's DCDC B output (VOB) is powering the EFR32's DECOUPLE supply,
// EFR32 must manage the handoff from EFR32's internal LDO to the EFP01 DCDC output.
//
// The EFP01 configuration settings below are passed to a EFP01 driver function (sl_efp_decouple_handoff())
// to manage a seamless transition of the DECOUPLE power supply from the internal EFR32 LDO to the EFP01
// DCDC B output. During this transition, certain EFP01 settings (inrush current, on-time maximum, peak
current)
// are momentarily set to conservative values, and then restored to the user-desired configuration values.
//
// If this managed-handoff sequence is not used, there is the possibility of creating a momentary voltage
// overshoot on the DECOUPLE that exceeds the DECOUPLE pin maximum voltage. Therefore, the
// sl_efp_decouple_handoff() function should be used whenever powering DECOUPLE from EFP01.
//
// If SL_EFP_DECOUPLE_HANDOFF_ARGS_SIZE > 0, the EFP01 init() function will attempt to call
// the sl_efp_decouple_handoff() function; otherwise, sl_efp_decouple_handoff() will not be called.

#define SL_EFP_DECOUPLE_HANDOFF_ARGS_SIZE 3

#define SL_DECOUPLE_HANDOFF_ARGS {0, 7, 10}    // BK_IRI_CON=0, BK_TON_MAX=7, BK_IPK=10
```

5. EFP01 Decouple Handoff Routine

Note: The sl_efp_decouple_handoff() routine and the header file parsing discussed in ● Generated Header File: DECOUPLE Handoff on page 11 may not be implemented in the EFP01 drivers until later in 2020. If that routine is not present in the drivers, it can be manually implemented by copying the routine below into the project. This sl_efp_decouple_handoff() routine should be called immediately following EFP01 initialization (i.e., el_efp_init()) whenever EFP01 is powering the EFR32 DECOUPLE pin, as shown below:

```
// Example sl_efp_decouple_handoff() manual implementation
static sl_efp_handle_t      efp = &efp_handle_data;
int main(void)
{
    sl_efp_init_data_t init = SL_EFP_INSTANCE_INIT_BRD4179B;
    sl_efp_init(efp, &init);  // Initialize EFP01

    // Parameters from the header file output can be manually passed to the
    // sl_efp_decouple_handoff() routine
   // Note for EFR32xG21 devices, use of the sl_efp_decouple_handoff() function requires updating
   // the EFR32xG21 Secure Element (SE) firmware to version 1.1.3 or later
    sl_efp_decouple_handoff(efp, 0, 7, 10);  // BK_IRI_CON=0, BK_TON_MAX=7, BK_IPK=10

    // Optional: if using Direct Mode, it should be manually enabled after the handoff routine
    // sl_efp_enable_direct_mode(efp);

}
```

Manual sl_efp_decouple_handoff() code

```
/*****************************************************************************//**
* @brief
*   Perform DECOUPLE LDO->DCDC Handoff sequence
*****************************************************************************/
void sl_efp_decouple_handoff(sl_efp_handle_t handle, uint8_t bk_iri_con, uint8_t bk_ton_max, uint8_t bk_ipk)
{
  sl_status_t status;
  uint8_t tmp;

  // Set VOB target to higher level to guarantee it will overdrive the EFR32 DECOUPLE LDO output
  sl_efp_set_vob_em01_voltage(handle, 1130);

  // Set peak current to minimum
  sl_efp_write_register_field(handle, EFP01_BK_IPK, 0,
                              _EFP01_BK_IPK_BK_IPK_MASK,
                              _EFP01_BK_IPK_BK_IPK_SHIFT);

  // Set Ton time to minimum
  sl_efp_write_register_field(handle, EFP01_BK_CTRL1,
                              1,
                              _EFP01_BK_CTRL1_BK_TON_MAX_MASK,
                              _EFP01_BK_CTRL1_BK_TON_MAX_SHIFT);

  // Set current limit to maximum
  sl_efp_write_register_field(handle, EFP01_BK_CTRL2,
                              15,
                              _EFP01_BK_CTRL2_BK_IRI_CON_MASK,
                              _EFP01_BK_CTRL2_BK_IRI_CON_SHIFT);

  // Enable VOB DCDC in buck only mode
  sl_efp_set_vob_mode(handle, efp_vob_mode_buck);

  // Make sure VOB output is ready before turning off internal LDO regulator.
  do {
    status = sl_efp_read_register(handle, EFP01_STATUS_LIVE, &tmp);
  } while (((tmp & _EFP01_STATUS_LIVE_VOB_INREG_LIVE_MASK) == 0)
           || (status != SL_STATUS_OK));

  // Set desired peak current
  sl_efp_write_register_field(handle, EFP01_BK_IPK, bk_ipk,
                              _EFP01_BK_IPK_BK_IPK_MASK,
                              _EFP01_BK_IPK_BK_IPK_SHIFT);

  // Set desired TON MAX
  sl_efp_write_register_field(handle, EFP01_BK_CTRL1,
                              bk_ton_max,
                              _EFP01_BK_CTRL1_BK_TON_MAX_MASK,
                              _EFP01_BK_CTRL1_BK_TON_MAX_SHIFT);

  // Set desired current limit
  sl_efp_write_register_field(handle, EFP01_BK_CTRL2,
                              bk_iri_con,
                              _EFP01_BK_CTRL2_BK_IRI_CON_MASK,
                              _EFP01_BK_CTRL2_BK_IRI_CON_SHIFT);

  // Turn off internal LDO regulator.
  sl_efp_emu_ldo_enable(handle, false);

  // Set desired VOB voltage
  sl_efp_set_vob_em01_voltage(handle, 1100);
}
```

6. Revision History

Revision 1.1

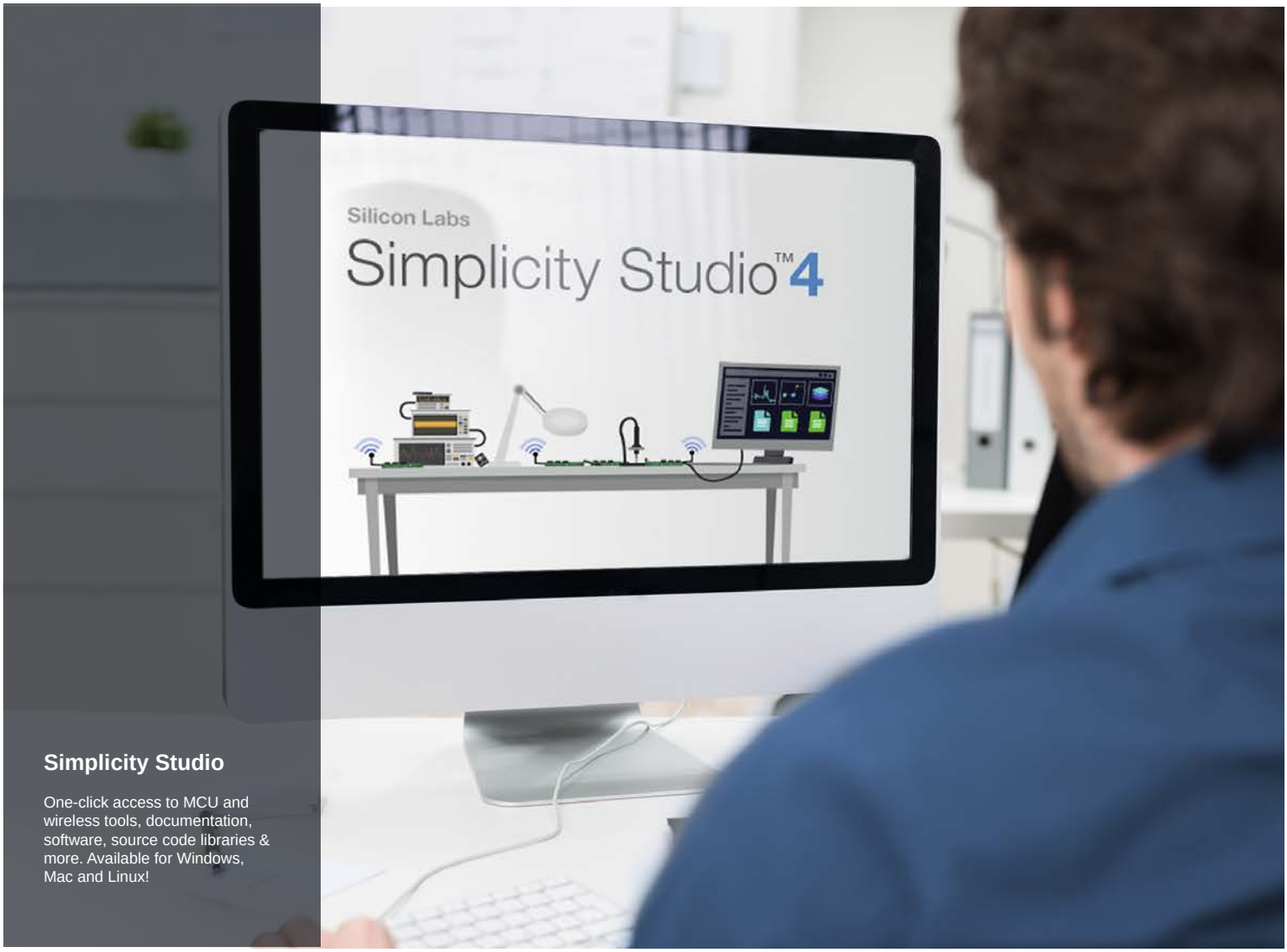
May, 2020

- Added manual DECOUPLE handoff routine code example and usage notes 5. EFP01 Decouple Handoff Routine
- Updated key points on front page.

Revision 1.0

March, 2020

- Initial revision



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

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