Boost Regulator Datasheet BOOST V 1.0

Copyright © 2009-2012 Cypress Semiconductor Corporation. All Rights Reserved.

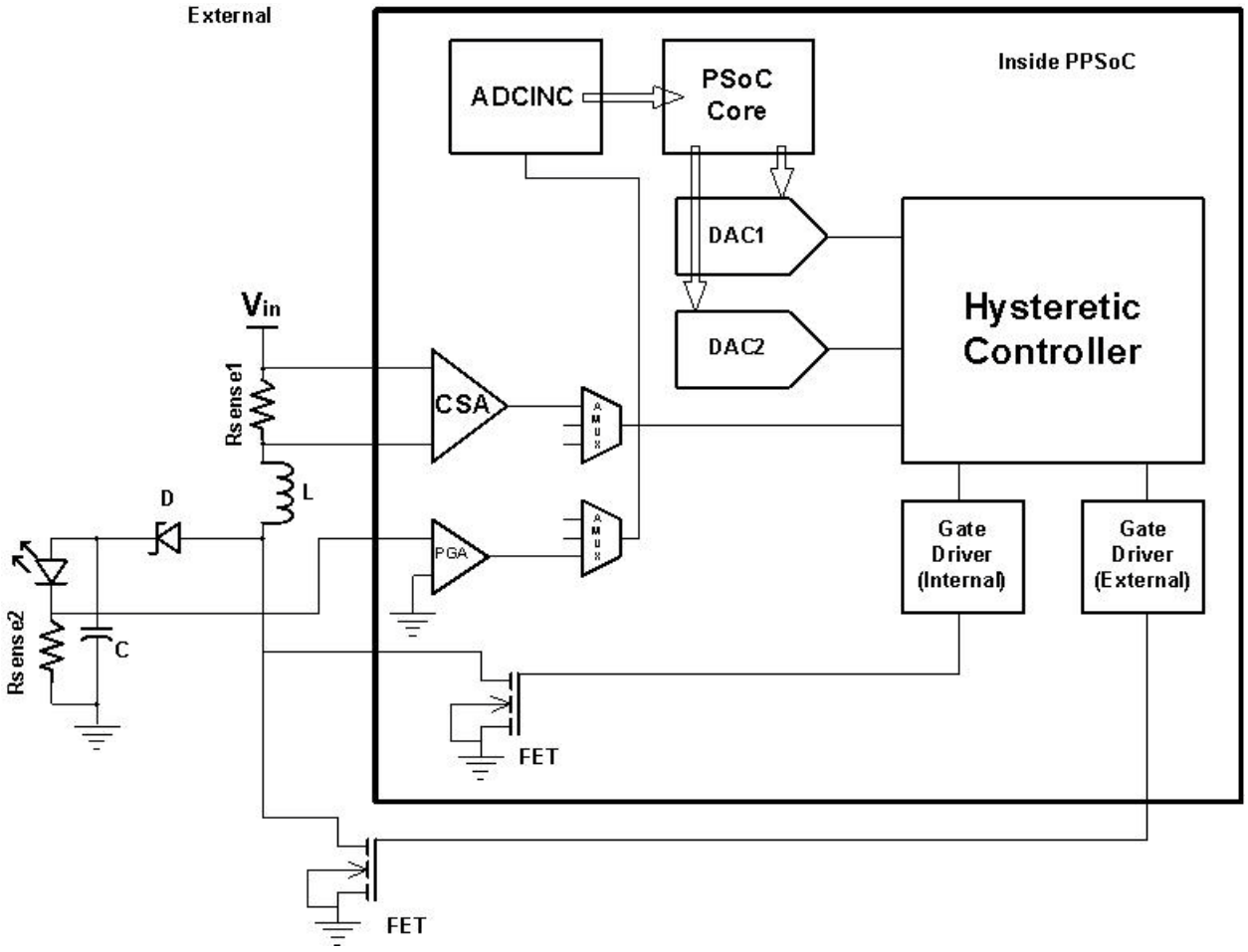
PSoC® Blocks			PowerPSoC Blocks			API Memory (Bytes)		
Digital	Analog CT	Analog SC	CSA	HYSTCTRL	Power FET	flash	RAM	Pins (per External I/O)
CY8CLED0xD, CY8CLED0xG								
1	1	2	4	4	4	980 bytes	17 bytes	4 to 6

Features and Overview

- Highly efficient boost control operation
- Fast hysteretic control loop
- Software integral loop for current regulation
- Easy-to-implement optional current protection

The Boost Regulator User Module is a current-to-direct direct current (DC) converter that steps up the source DC voltage from a lower level to a desired higher level.The Boost Regulator User Module may be used for driving High Brightness LEDs and other applications.

Figure 1. BOOST Regulator Block Diagram



Functional Description

The Boost Regulator User Module is a complete current regulator that performs an output voltage step-up power conversion. It provides output current stabilization in specified ranges and overcurrent protection. The user module topology allows it to work with four channels at the same time.

The user module is designed to work with internal and high power N-Channel FETs. The main current feedback control may come from either the integrated current sense amplifier or from external circuitry through the FN_0 port. The internal ADC is used to sense the current through the load. It can sample from any one of four analog pins at a time.

The Boost Regulator topology requires an energy storage element (inductor L), two sense resistors (R_{SENSE1} and R_{SENSE2}), a Shottky diode (diode D), and a filter (capacitor C) that are external to the PSoC device. Internally, the User Module sets up an incremental two-stage ADC, two amplifiers (PGA and CSA) and a Hysteretic Controller. The two DACs are dedicated to the Hysteretic Controller (HCT) and serve as thresholds. Additionally, the Boost regulator requires a switch (FET) that can be either internal or external.

This switching regulator keeps track of current changes on the load (LEDs) by monitoring the voltage drop on the R_{SENSE2} resistor. The Boost scheme does not allow you to use R_{SENSE1} for this purpose because

the current flowing through R_{SENSE1} differs from the current running through the load and R_{SENSE2}. The Current Sense Amplifier (CSA) is used to sense across the R_{SENSE1} resistor and Programmable Gain Amplifier (PGA) is used for the R_{SENSE2} resistor.

The CSA along with the HCT and two DACs provides a hardware based feedback loop. It controls the inductor current. A software based integral control loop controls the PGA, the two DACs, the HCT, the PSoC core and increments the ADC. It performs load current control. The PSoC core computes the error and corrects it.

The BOOST Regulator User Module operates in this method:

- When the FET is on the current runs through the inductor to ground. The inductor accumulates energy in this cycle.
- When FET is off the inductor current runs through the diode, LEDs, and the capacitor. The accumulated energy is transferred to the load.

The inductor current for the open FET state can be calculated as:

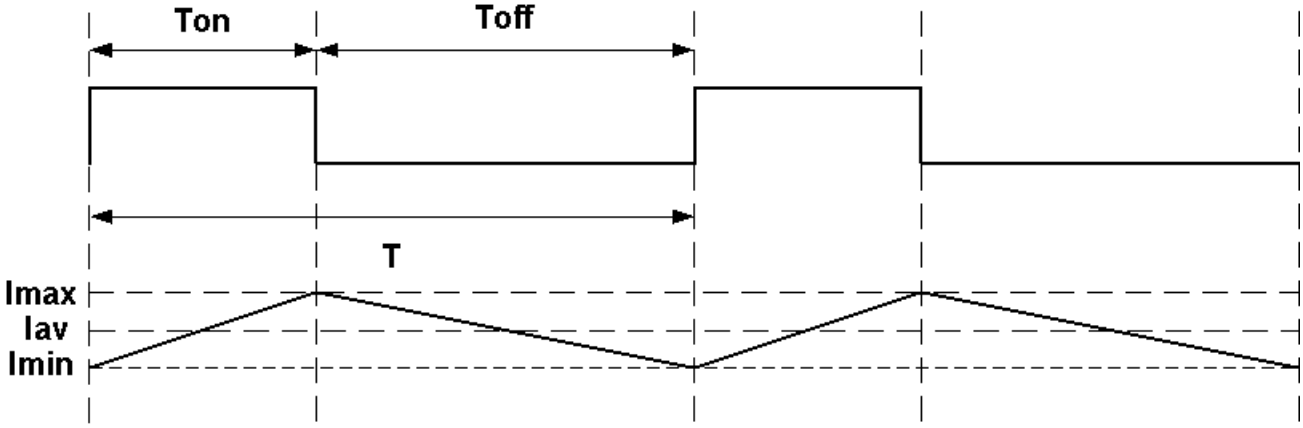
Equation 1

$$\Delta I_{L_{on}} = \frac{V_i \cdot Ton}{L}$$

In this equation, Vi is the input voltage, and Ton is the commutation period when the FET is on.

This figure represents the BOOST Regulator operation diagram.

Figure 2. BOOST Operation Diagram



The equation for the closed FET state inductor current is:

Equation 2

$$\Delta I_{L_{off}} = \frac{(V_i - V_o) \cdot Toff}{L}$$

The duty cycle is defined by this equation:

Equation 3

$$D = \frac{Ton}{T} = 1 - \frac{Vi}{Vo}$$

The output current:

Equation 4

$$I_o = \frac{Vi^2 \cdot D^2 \cdot T}{2 \cdot L \cdot (Vo - Vi)}$$

The boost regulator trip function notifies the software that a current value exceeds the specified threshold. It overrides the normal HCT main loop operation.

The Boost Regulator User Module has a wizard that helps you set the user module parameters properly for your application. It provides flexible tuning for internal hardware and gives you an opportunity to compute efficiency and other factors.

DC and AC Electrical Characteristics

See the device datasheet for your PowerPSoC device for electrical characteristics.

Placement

The Boost User Module occupies any available digital block, the entire second analog column, and a set of power blocks (CSA, HYSTCTRL, FET).

Wizard

The BOOST wizard assists you with setting the setting the parameters of the BOOST user module to match your chosen application for the BOOST User Module.

Accessing the Wizard

To access the wizard, right click the **Boost Regulator User Module** in the Workspace Explorer and select the **BOOST Wizard**.

Working with the Wizard

The first page of the BOOST wizard allows you to select an approach to configuration. The possible choices are "Configure all channels with the same hardware settings" and "Configure each channel independently." The default selection is "Configure all channels with the same hardware settings."

Figure 3. The First Wizard Page

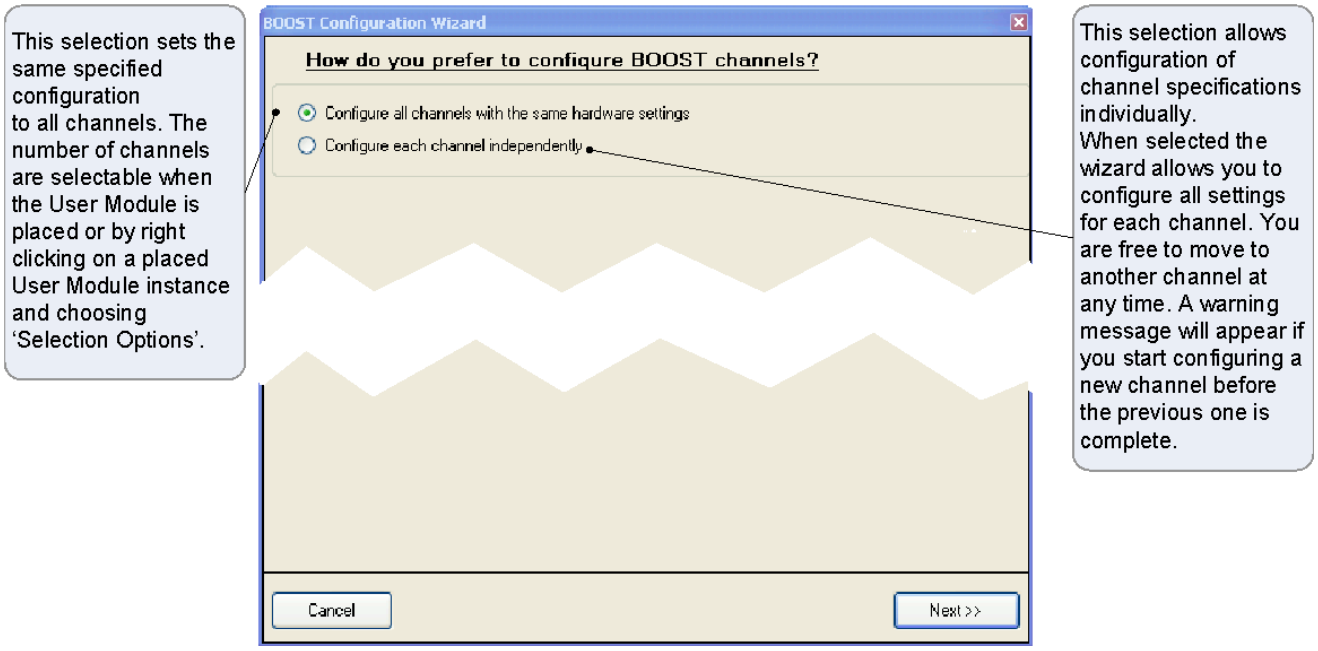
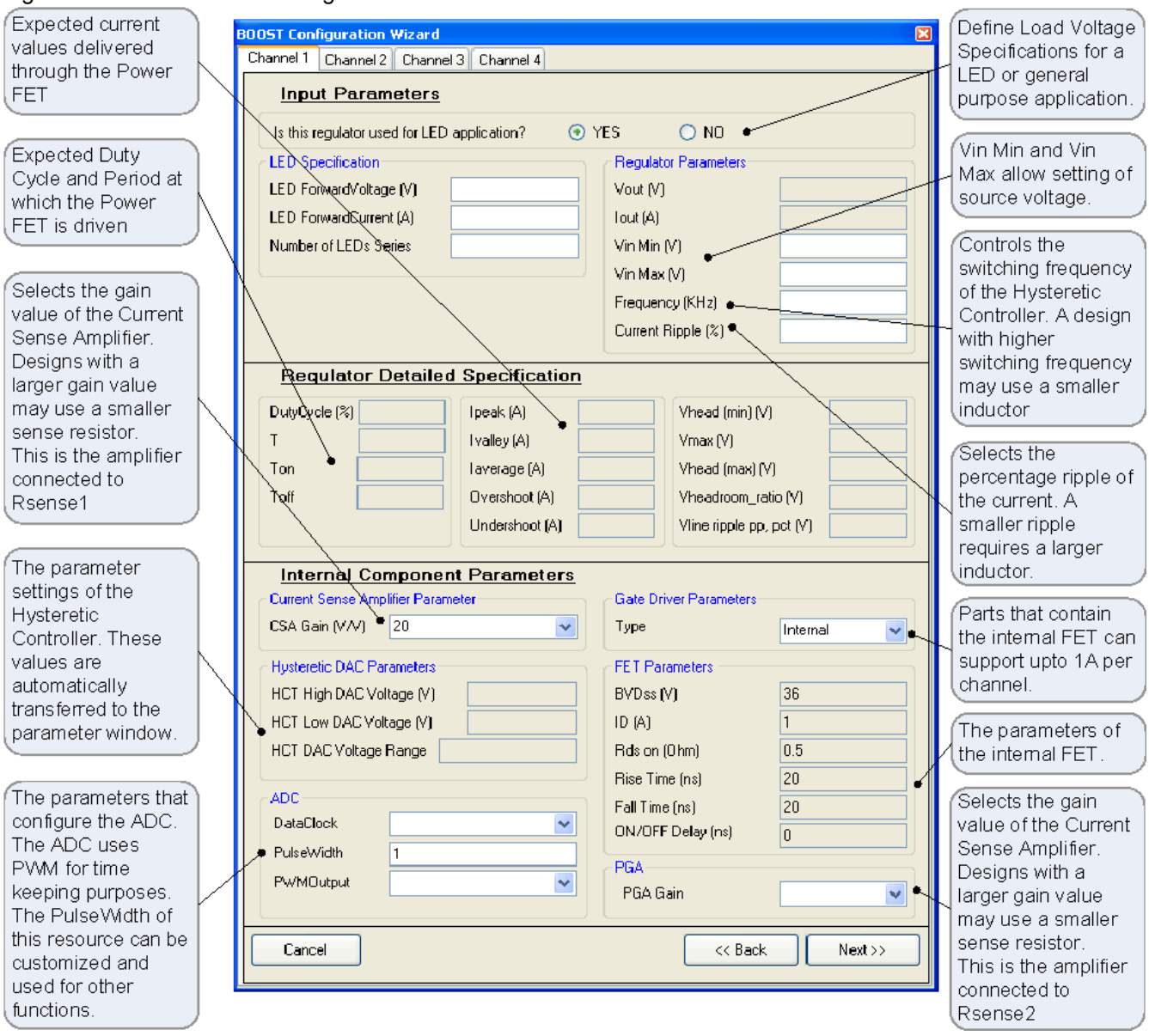
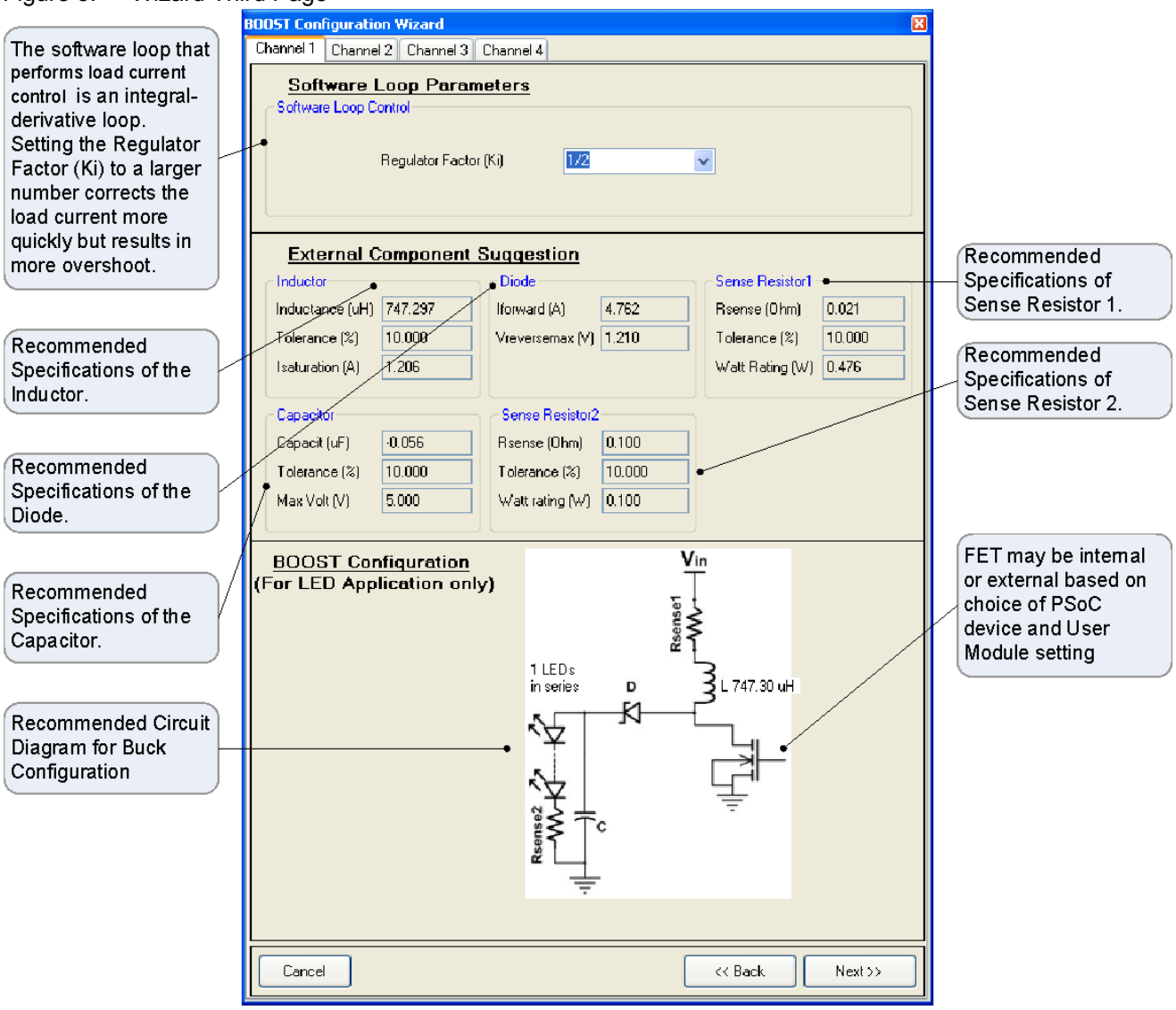


Figure 4. Second Wizard Page



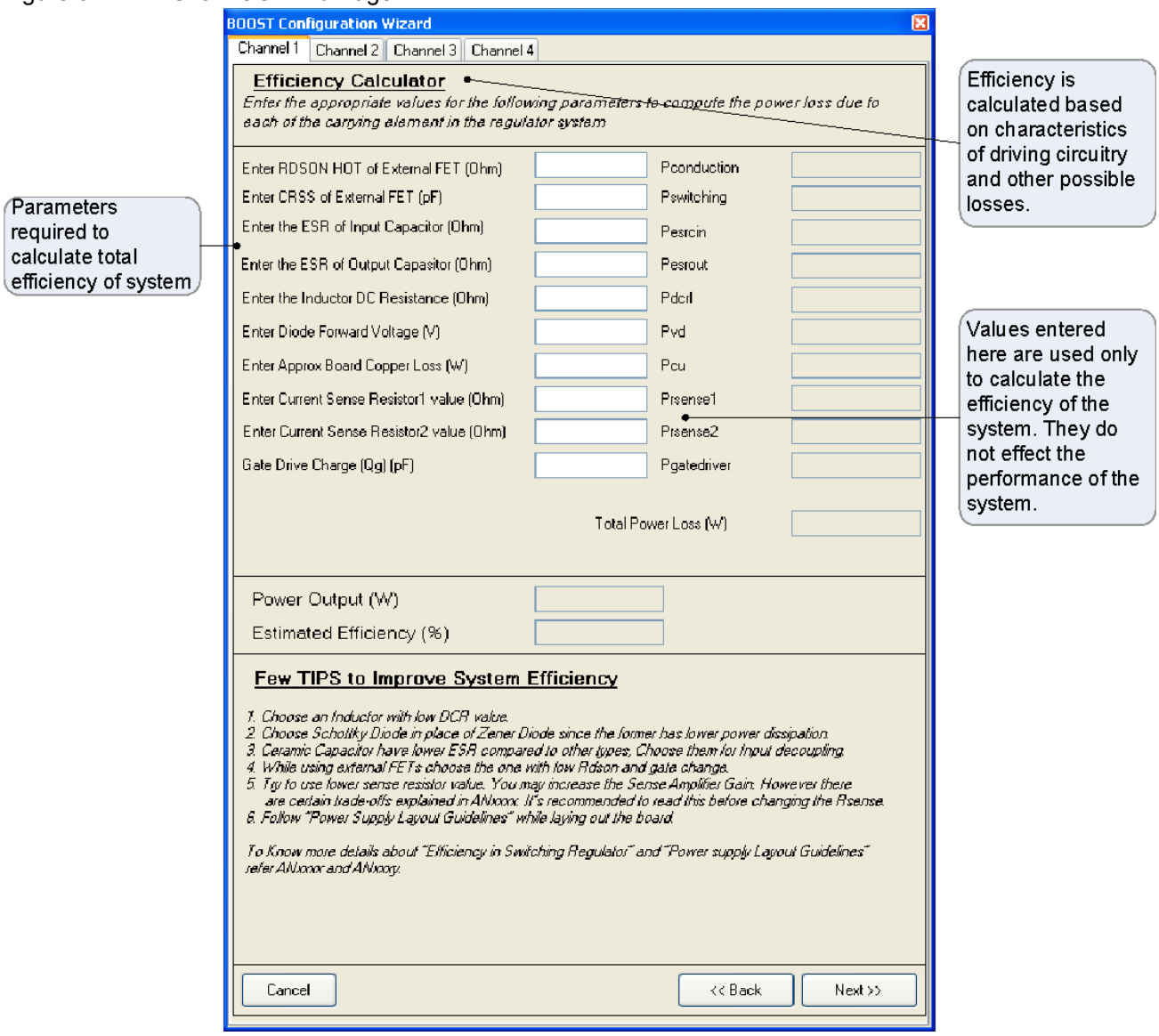
The second page is used to enter general characteristics. The middle section shows values calculated based on your inputs.

Figure 5. Wizard Third Page



The third page is divided into three sections. The first section allows you to tune the software loop. The second section shows the calculations for external components. The third section shows you a schematic of the Boost components.

Figure 6. All Channels Third Page



The fourth page is an efficiency calculator. It calculates the power loss for each external component and the total power loss.

Parameters and Resources

The parameters section uses the following abbreviations:

Abbreviation	Meaning
CSA	Current sense amplifier block
HCT	Hysteretic controller block
PWM	Digital block
GAIN, ADC1, ADC2	The analog blocks

PGA_Gain

The PGA_Gain is selected from values ranging from 1.00 to 48.00. The values can also be set at run time using the BOOST_SetGain() routine provided in the API.

ADC_DataClock

The data clock determines the sample rate. This clock goes to all four PSoC blocks (PWM, GAIN, ADC1, ADC2).

It is imperative that the same clock be used for both the digital block and the analog column clock or this user module does not function correctly.

The Data Clock should not be set to less than 250 kHz when the CPU is running at 24 MHz. Otherwise, it may be set as low as 125 kHz. The Data Clock may not exceed the CPU clock, it must always be equal to or less than CPU Clock. The PWM is set to provide an interrupt every 256 counts of the Data Clock.The counter integrates the signal for 16 cycles. An additional cycle is required to reset the integrator and process the data. The sample rate is defined as:

Equation 5

$$SampleRate = \frac{DataClock}{256 \cdot 17} = \frac{DataClock}{4352}$$

The sample window determines the normal mode frequencies the ADC rejects. It is defined as:

Equation 6

$$SampleWindow = \frac{1024}{DataClock}$$

To reject a higher frequency and its harmonics, select the sample window such that it is an even multiple of the frequency-to-reject.

ADC_PulseWidth

Sets pulse width from a value 1 to 255 counts. If no value is set then the user module automatically sets the PWM pulse width to 1 when the BOOST_GetSamples() function is called.

ADC_PWMOutput

The Output parameter may be disabled or routed to one of four row outputs.

CSAx_Gain

This parameter sets the current sense amplifier gain. It controls both Stage 1 gain and Stage 2 bypass. The following options are provided:

Stage 1 Gain	Description
3	Sets Stage 1 amplifier gain 3 and bypasses Stage 2.
4	Sets Stage 1 amplifier gain 4 and bypasses Stage 2.
15	Sets Stage 1 amplifier gain 3 and uses Stage 2 amplifier with gain 5.
20 (Default)	Sets Stage 1 amplifier gain 4 and uses Stage 2 amplifier with gain 5.

CSAx_Bandwidth

The Bandwidth parameter controls the capacitance load at the output of the Stage 1 amplifier and provides bandwidth adjustment capability, allowing trade-offs in bandwidth, time delay, and power supply rejection ratio. The bandwidth can be selected from one of the following values:

Bandwidth	Description
Highest (Default)	Highest bandwidth, no capacitance added to Stage 1 output.
MediumHigh	Medium high bandwidth.
MediumLow	Medium low bandwidth.
Lowest	Lowest bandwidth, most capacitance added.

HCTx_FeedbackInput

Defines the HCT block Feedback Input connection. The choices are the CSAx block output or FN_0_x pin.

HCTx_DACVoltageRange

Selects the voltage range of the reference DACs. The choices are 1.3V and 2.6V.

HCTx_RefHigh

Configures the high reference of the DAC output voltage. The DAC reference dictates the output of the hysteretic controller block and it is important to choose the correct value for this parameter. Use the following formulas:

Equation 7

$$PeakCurrent = Current + \frac{Ripple \times Current}{2}$$

Equation 8

$$SenseVoltage = PeakCurrent \times SenseRegister$$

Equation 9

$$DACVoltage = SenseVoltage \times CSAGain$$

Equation 10

$$DACValue = \frac{DACVoltage \times 255}{VoltageRange}$$

In the last formula VoltageRange is the value of HCT_DACVoltageRange parameter (1.3V or 2.6V). Consider an example LED application intended to drive 350 mA of constant current through the LEDs using Floating Load Buck topology. Assume this circuit uses an external sense resistor of 0.22W, the internal current sense amplifier gain is set to 20, the desired ripple is 10% with 1 MHz switch frequency, and the DAC is set in the 2.6V range. For an ideal selection of external components, the HCT_RefHigh DAC value is computed as:

$$PeakCurrent(mA) = 350 + \frac{0.1 \times 350}{2} = 367.5$$

$$SenseVoltage(mV) = 367.5 \times 0.22 = 80.85$$

$$DACVoltage(V) = 73.15 \times 20 = 1.463$$

$$DACValue \text{ to be loaded (approximate HEX)} = \frac{1.617 \times 255}{2.6} = 9C$$

Actual voltage on the DAC output depends on this parameter and the HCT_DACVoltageRange parameter value.

HCTx_RefLow

Configures the low reference DAC output voltage. The DAC reference dictates the output of the hysteretic controller block and it is important to choose the correct value for this parameter. Use the following formulas:

Equation 11

$$ValleyCurrent = Current - \frac{Ripple \times Current}{2}$$

Equation 12

$$SenseVoltage = ValleyCurrent \times SenseRegister$$

Equation 13

$$DACVoltage = SenseVoltage \times CSAGain$$

Equation 14

$$DACValue = \frac{DACVoltage \times 255}{VoltageRange}$$

Consider an example LED application intended to drive 350 mA of constant current through the LEDs using Floating Load Buck topology. Assume this circuit uses an external sense resistor of 0.22W, the internal current sense amplifier gain is set to 20, the desired ripple is 10% with 1 MHz switch frequency, and the DAC is set in the 2.6V range. For an ideal selection of external components, the HCT_RefLow DAC value is computed as:

$$ValleyCurrent(mA) = 350 - \frac{0.1 \times 350}{2} = 332.5$$

$$SenseVoltage(mV) = 332.5 \times 0.22 = 73.15$$

$$DACVoltage(V) = 73.15 \times 20 = 1.463$$

$$DACValue \text{ to be loaded (approximate HEX)} = \frac{1.463 \times 255}{2.6} = 8F$$

Actual voltage on the DAC output depends on this parameter in conjunction with the HCT_DACVoltageRange parameter value.

HCTx_TripInput

Selects the TRIP input connection to the HCT block. It is possible to connect one of the hardware comparators, one of the FN_0_x pins or any of Vgnd (Vgnd0 or Vgnd1) as input.

HCTx_TimerDelay

Sets the monoshot timer delay for both on and off timers. Possible choices are no delay, 10-25 ns delay, 20-50 ns delay or 40-100 ns for both on and off timers.

HCTx_GateDriver

Configures the operation of the gate drivers. Possible choices are disable, internal driver enabled or external driver enabled.

HCTx_GateDriverStrength

Configures the strength of the gate drivers. Possible choices are default, 75% of default, 50% of default, and 25% of default.

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the include files.

Each time a user module is placed, it is assigned an instance name. By default, PSoC Designer assigns the BOOST_1 to the first instance of this user module in a given project. It can be changed to any unique value that follows the syntactic rules for identifiers. The assigned instance name becomes the prefix of every global function name, variable, and constant symbol. In the following descriptions the instance name has been shortened to BOOST for simplicity.

Note

** In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X before the call if those values are required after the call. This "registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API functions may leave A and X unchanged, there is no guarantee they may do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

BOOST_Start

Description:

Starts the BOOST operation for specified channel or channels.

C Prototype:

```
void BOOST_Start(BYTE bChannel);
```

Assembler:

```
mov  A, bChannel
lcall BOOST_Start
```

Parameters:

bChannel - one of the following constants:

Channel Number	Channel Mask	Value	Description
1st	BOOST_CHANNEL1	0x00	Selects first channel
2nd	BOOST_CHANNEL2	0x01	Selects second channel
3rd	BOOST_CHANNEL3	0x02	Selects third channel
4th	BOOST_CHANNEL4	0x03	Selects fourth channel
All	BOOST_ALL_CHANNELS	0x04	Selects all channels

Return Value:

None.

Side Effects:

See Note** at the beginning of the API section.

BOOST_Stop

Description:

Stops the BOOST operation for the specified channel or channels.

C Prototype:

```
void BOOST_Stop(BYTE bChannel);
```

Assembler:

```
mov  A, bChannel
lcall BOOST_Stop
```

Parameters:

bChannel - one of the following constants:

Channel Number	Channel Mask	Value	Description
1st	BOOST_CHANNEL1	0x00	Selects first channel
2nd	BOOST_CHANNEL2	0x01	Selects second channel
3rd	BOOST_CHANNEL3	0x02	Selects third channel
4th	BOOST_CHANNEL4	0x03	Selects fourth channel
All	BOOST_ALL_CHANNELS	0x04	Selects all channels

Return Value:

None.

Side Effects:

See Note** at the beginning of the API section.

BOOST_GetSamples

Description:

Runs the ADC for the specified number of samples.

C Prototype:

```
void BOOST_GetSamples(BYTE bNumSamples);
```

Assembler:

```
mov  A, bNumSamples
lcall BOOST_GetSamples
```

Parameters:

bNumSamples - an 8-bit value that sets the number of samples to be converted. A value of ‘0’ causes the ADC to run continuously.

Return Value:

None.

Side Effects:

See Note** at the beginning of the API section.

BOOST_StopADC

Description:

Immediately stops the ADC. The PWM continues to run.

C Prototype:

```
void BOOST_StopADC(void);
```

Assembler:

```
lcall BOOST_StopADC
```

Parameters:

None

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

BOOST_fIsDataAvailable

Description:

Checks the availability of sampled data.

C Prototype:

```
BYTE BOOST_fIsDataAvailable(void);
```

Assembler:

```
lcall BOOST_fIsDataAvailable
;now A contains a returned value (sampled data availability)
```

Parameters:

None

Return Value:

Returns a nonzero value if data has been converted and is ready to read.

Side Effects:

See Note** at the beginning of the API section.

BOOST_wClearFlagGetData

Description:

Clears the data ready flag and gets converted data as unsigned integer. Checks to see that data-flag is still reset. If not the data is retrieved again. This makes sure that the ADC interrupt routine did not update the answer while it was being collected.

C Prototype:

```
WORD BOOST_wClearFlagGetData(void);
```

Assembler:

```
lcall BOOST_wClearFlagGetData  ;Data will be in A and X upon return
```

Parameters:

None

Return Value:

Returns the converted 16-bit unsigned data sample. X contains MSB, A - LSB of result.

Side Effects:

See Note** at the beginning of the API section.

BOOST_wGetData

Description:

Returns converted data as an unsigned integer. BOOST_fIsDataAvailable() should be called to verify that the data sample is ready.

C Prototype:

```
WORD BOOST_wGetData(void);
```

Assembler:

```
lcall BOOST_wGetData ;Data will be in A and X upon return
```

Parameters:

None

Return Value:

Returns the converted 16-bit unsigned data sample. X contains MSB, A - LSB of result.

Side Effects:

See Note** at the beginning of the API section.

BOOST_fClearFlag

Description:

Clears the data ready flag and return previous status.

C Prototype:

```
BYTE BOOST_fClearFlag(void);
```

Assembler:

```
lcall BOOST_fClearFlag
;now A contains a returned value (status)
```

Parameters:

None

Return Value:

This returns the value of the status register.

Side Effects:

See Note** at the beginning of the API section.

BOOST_WritePulseWidth

Description:

Changes the pulse width of the PWM.

C Prototype:

```
void BOOST_WritePulseWidth(BYTE bPulseWidthValue);
```

Assembler:

```
mov  A, bPulseWidthValue
lcall BOOST_WritePulseWidth
```

Parameters:

bPulseWidthValue - PWM pulse width. This value must not be zero. The ADC stops functioning if the pulse width is set to zero.

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

BOOST_SetCSAGain

Description:

Sets the current sense amplifier gain for given channel. Controls both Stage 1 gain and Stage 2 bypass.

C Prototype:

```
void BOOST_SetCSAGain(BYTE bChannel, BYTE bGain);
```

Assembler:

```
mov  X, bGain
mov  A, bChannel
lcall BOOST_SetCSAGain
```

Parameters:

bChannel - one of the following constants:

Channel Number	Channel Mask	Value	Description
1st	BOOST_CHANNEL1	0x00	Selects first channel
2nd	BOOST_CHANNEL2	0x01	Selects second channel
3rd	BOOST_CHANNEL3	0x02	Selects third channel
4th	BOOST_CHANNEL4	0x03	Selects fourth channel
All	BOOST_ALL_CHANNELS	0x04	Selects all channels

bGain - is the gain value. Symbolic names are provided in C and assembly. Their associated values are given in the following table:

Symbolic Name	Value	Description
BOOST_GAIN_3	0x0A	Sets Stage 1 amplifier gain 3 and bypasses Stage 2.
BOOST_GAIN_4	0x08	Sets Stage 1 amplifier gain 4 and bypasses Stage 2.
BOOST_GAIN_15	0x02	Sets Stage 1 amplifier gain 3 and uses Stage 2 amplifier with gain 5.
BOOST_GAIN_20	0x00	Sets Stage 1 amplifier gain 4 and uses Stage 2 amplifier with gain 5.

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

BOOST_SetDACVoltageRange

Description:

Sets the range of hysteretic DAC output voltage. This single parameter sets the range for both reference DACs.

C Prototype:

```
void BOOST_SetDACVoltageRange(BYTE bChannel, BYTE bVoltageRange);
```

Assembler:

```
mov  X, bVoltageRange
mov  A, bChannel
lcall BOOST_SetDACVoltageRange
```

Parameters:

bChannel - one of the following constants:

Channel Number	Channel Mask	Value	Description
1st	BOOST_CHANNEL1	0x00	Selects first channel
2nd	BOOST_CHANNEL2	0x01	Selects second channel
3rd	BOOST_CHANNEL3	0x02	Selects third channel
4th	BOOST_CHANNEL4	0x03	Selects fourth channel
All	BOOST_ALL_CHANNELS	0x04	Selects all channels

bVoltageRange - indicates the output range of the reference DACs. Symbolic names provided in C and assembly, and their associated values, are given in the following table:

Symbolic Name	Value	Description
BOOST_DAC_1_3V	0x02	Sets the reference DAC voltage range from 0V to 1.3V.
BOOST_DAC_2_6V	0x00	Sets the reference DAC voltage range from 0V to 2.6V.

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

BOOST_SetRefHigh

Description:

Writes an 8-bit data code to the high reference DAC data register. This changes the high reference DAC output voltage setup.

C Prototype:

```
void BOOST_SetRefHigh(BYTE bChannel, BYTE bData);
```

Assembler:

```
mov  X, bData
mov  A, bChannel
lcall BOOST_SetRefHigh
```

Parameters:

bChannel - one of the following constants:

Channel Number	Channel Mask	Value	Description
1st	BOOST_CHANNEL1	0x00	Selects first channel
2nd	BOOST_CHANNEL2	0x01	Selects second channel
3rd	BOOST_CHANNEL3	0x02	Selects third channel
4th	BOOST_CHANNEL4	0x03	Selects fourth channel
All	BOOST_ALL_CHANNELS	0x04	Selects all channels

bData - defines the 8-bit data code to be loaded in the high reference DAC.

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

BOOST_SetRefLow

Description:

Writes an 8-bit data code to low reference DAC data register. This changes the low reference DAC output voltage setup.

C Prototype:

```
void BOOST_SetRefLow(BYTE bChannel, BYTE bData);
```

Assembler:

```
mov  X, bData
mov  A, bChannel
lcall BOOST_SetRefLow
```

Parameters:

bChannel - one of the following constants:

Channel Number	Channel Mask	Value	Description
1st	BOOST_CHANNEL1	0x00	Selects first channel
2nd	BOOST_CHANNEL2	0x01	Selects second channel
3rd	BOOST_CHANNEL3	0x02	Selects third channel
4th	BOOST_CHANNEL4	0x03	Selects fourth channel
All	BOOST_ALL_CHANNELS	0x04	Selects all channels

bData - defines the 8-bit data code to be loaded in the low reference DAC.

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

BOOST_SetPGAGain

Description:

Sets the gain for output current monitoring system.

C Prototype:

```
void BOOST_SetPGAGain(BYTE bGain);
```

Assembler:

```
mov  A, bGain
lcall BOOST_SetPGAGain
```

Parameters:

bGain - programmable gain. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value	Symbolic Name	Value
BOOST_PGA_G16_0	0x08	BOOST_PGA_G1_78	0x88
BOOST_PGA_G8_00	0x18	BOOST_PGA_G1_60	0x98
BOOST_PGA_G5_33	0x28	BOOST_PGA_G1_46	0xA8
BOOST_PGA_G4_00	0x38	BOOST_PGA_G1_33	0xB8
BOOST_PGA_G3_20	0x48	BOOST_PGA_G1_23	0xC8
BOOST_PGA_G2_67	0x58	BOOST_PGA_G1_14	0xD8
BOOST_PGA_G2_27	0x68	BOOST_PGA_G1_06	0xE8
BOOST_PGA_G2_00	0x78	BOOST_PGA_G1_00	0xF8

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

BOOST_SelectChannel

Description:

This function sets ADC to given channel or appropriate external pin

C Prototype:

```
void BOOST_SelectChannel(BYTE bChannel);
```

Assembler:

```
mov  A, bChannel
lcall BOOST_SelectChannel
```

Parameters:

bChannel - number of necessary channel.

Channel Number	Channel Mask	Appropriate external pin	Value	Description
1st	BOOST_CHANNEL1	P0[3]	0x00	Selects first channel
2nd	BOOST_CHANNEL2	P0[5]	0x01	Selects second channel
3rd	BOOST_CHANNEL3	P0[7]	0x02	Selects third channel
4th	BOOST_CHANNEL4	P0[4]	0x03	Selects fourth channel

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

BOOST_Regulate

Description:

Trims output current setting for given channel(s).

C Prototype:

```
void BOOST_Regulate(BYTE bChannel);
```

Assembler:

```
mov  A, bChannel
lcall BOOST_Regulate
```

Parameters:

bChannel - one of the following constants:

Channel Number	Channel Mask	Value	Description
1st	BOOST_CHANNEL1	0x00	Selects first channel

Channel Number	Channel Mask	Value	Description
2nd	BOOST_CHANNEL2	0x01	Selects second channel
3rd	BOOST_CHANNEL3	0x02	Selects third channel
4th	BOOST_CHANNEL4	0x03	Selects fourth channel
All	BOOST_ALL_CHANNELS	0x04	Selects all channels

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

BOOST_SetChannelCurrent

Description:

Sets channel current.

C Prototype:

```
void BOOST_SetChannelCurrent(BYTE bChannel, WORD wCurrentValue);
```

Assembler:

```
mov  A, [wCurrentValue + 0]
push A
mov  A, [wCurrentValue + 1]
push A
mov  A, bChannel
push A
lcall BOOST_SetChannelCurrent
add SP, -3
```

Parameters:

wCurrentValue - defines the 16-bit data code to be set for bChannel.

bChannel - one of the following constants:

Channel Number	Channel Mask	Value	Description
1st	BOOST_CHANNEL1	0x00	Selects first channel
2nd	BOOST_CHANNEL2	0x01	Selects second channel
3rd	BOOST_CHANNEL3	0x02	Selects third channel
4th	BOOST_CHANNEL4	0x03	Selects fourth channel

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

BOOST_wGetChannelCurrent

Description:

Gets channel current.

C Prototype:

```
WORD BOOST_wGetChannelCurrent(BYTE bChannel);
```

Assembler:

```
mov  A, bChannel
lcall BOOST_wGetChannelCurrent
;Current value is stored now in X(MSB) and A(LSB)
```

Parameters:

bChannel - one of the following constants:

Channel Number	Channel Mask	Value	Description
1st	BOOST_CHANNEL1	0x00	Selects first channel
2nd	BOOST_CHANNEL2	0x01	Selects second channel
3rd	BOOST_CHANNEL3	0x02	Selects third channel
4th	BOOST_CHANNEL4	0x03	Selects fourth channel

Return Value:

Returns the current value through X (MSB) and A (LSB)

Side Effects:

See Note** at the beginning of the API section.

Sample Firmware Source Code

The code illustrated here shows you how to use the BOOST Regulator User Module.

This example lights up the LED(s) changing output current. The second channel is used here.

The code in C language is:

```
#include <m8c.h>      // part specific constants and macros
#include "PSoCAPI.h"  // PSoC API definitions for all User Modules

WORD delay;
BYTE count;
WORD crnt;

void main(void)
{
M8C_EnableGInt;
BOOST_Start(BOOST_CHANNEL2);
```

```
BOOST_SetChannelCurrent(BOOST_CHANNEL2, 400);
count = 0;

while(1)
{
BOOST_SelectChannel(BOOST_CHANNEL2);
BOOST_GetSamples(1);
while(!BOOST_fIsDataAvailable());
crnt=BOOST_wClearFlagGetData();

for(delay=2200; delay; delay--);
BOOST_Regulate(BOOST_CHANNEL2);

count++;
if(count == 200)
{
count=0;
crnt = BOOST_wGetChannelCurrent(BOOST_CHANNEL2);
if(crnt == 400) BOOST_SetChannelCurrent(BOOST_CHANNEL2, 100);
else if(crnt == 100) BOOST_SetChannelCurrent(BOOST_CHANNEL2, 400);
}
}
}
```

The same code in assembly is:

```
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

AREA bss

count: blk 1
crnt:  blk 2

AREA text
_main:

    M8C_EnableGInt

 ;BOOST_Start(BOOST_CHANNEL2);
    mov A, BOOST_CHANNEL2
    call BOOST_Start

 ;BOOST_SetChannelCurrent(BOOST_CHANNEL2, 400);
    mov A, 01h
    push A
    mov A, 90h
    push A
    mov A, BOOST_CHANNEL2
    push A
    call BOOST_SetChannelCurrent
    add SP, -3
```

```
 ;count = 0;
    RAM_SETPAGE_CUR >count
    mov [count], 0

.BOOST_CYCLE:  ;while(1)
    ;BOOST_SelectChannel(BOOST_CHANNEL2);
    mov A, BOOST_CHANNEL2
    call BOOST_SelectChannel

 ;BOOST_GetSamples(1);
    mov A, 1
    call BOOST_GetSamples

.DataIsAvailable: ;while(!fIsDataAvailable());
    call BOOST_fIsDataAvailable
    cmp A, 0
    jz .DataIsAvailable

 ;crnt=BOOST_wClearFlagGetData();
    call BOOST_wClearFlagGetData
    ; A & X - wChannelCurrent

 ;for(delay=2200; delay; delay--);
    mov X, 10
.ForCycle:
     mov A, FFh
.InternalCycle:
     nop
     nop
     nop
     dec A
     jnz .InternalCycle
     dec X
     jnz .ForCycle

  ;BOOST_Regulate(BOOST_CHANNEL2);
     mov A, BOOST_CHANNEL2
     call BOOST_Regulate

  ;count++
     RAM_SETPAGE_CUR >count
     inc [count]

  ;if (count == 200)
     cmp [count], 200
  jnz .BOOST_CYCLE

  ;count=0;
     mov [count], 0

  ;crnt=BOOST_wGetChannelCurrent(BOOST_CHANNEL2);
     mov A, BOOST_CHANNEL2
     call BOOST_wGetChannelCurrent

  ;if(crnt == 400)
```

```
     sub A, 0x90
     swap A, X
     sbb A, 0x01
     jc .Set400

.Set100:  ;BOOST_SetChannelCurrent(BOOST_CHANNEL2, 100);
     mov A, 00h
     push A
     mov A, 64h
     push A
     mov A, BOOST_CHANNEL2
     push A
     call BOOST_SetChannelCurrent
     add SP, -3
     jmp .BOOST_CYCLE

.Set400:  ;BOOST_SetChannelCurrent(BOOST_CHANNEL2, 400);
     mov A, 01h
     push A
     mov A, 90h
     push A
     mov A, BOOST_CHANNEL2
     push A
     call BOOST_SetChannelCurrent
     add SP, -3
  jmp .BOOST_CYCLE
```

Configuration Registers

The following registers are used by the Application Programming Interface.

CSA Block Register

Table 1. BOOST_CSA_CONTROL_REG

Bit	7	6	5	4	3	2	1	0
Value	0	0	Bandwidth		Gain			Enable

The Enable bit is modified by calling the BOOST_Start() or BOOST_Stop() API routine.

The Gain bits determine the Stage 1 gain and control the Stage 2 amplifier bypass. The value of these bits is determined by the choice made for the CSA_Gain parameter under user module parameters in the Device Editor. The value can also be changed by calling the BOOST_SetCSAGain() API.

The Bandwidth bits determine the CSA_Bandwidth. The value of these bits is determined by the choice made for the parameter under user module parameters in the Device Editor.

Hysteretic Controller Registers

Table 2. BOOST_VDAC_CONTROL_REG

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	Mode	Enable

Enable enables or disables the VDAC. Disabling powers down the VDAC and all of its output references go to 0V. It is modified by calling BOOST_Start() or BOOST_Stop().

Mode sets the VDAC output range and step size. The initial value of this bit is set with the value of the HCT_DACVoltageRange parameter in the Device Editor. The value can be changed at runtime with the BOOST_SetDACVoltageRange() API.

Table 3. BOOST_VDAC_DATA0_REG

Bit	7	6	5	4	3	2	1	0
Value	RefHigh							

RefHigh determines the reference high DAC output voltage. The initial value of this bit is set with the value of the HCT_RefHigh parameter in the Device Editor. The value can be changed at runtime with the BOOST_SetRefHigh() API.

Table 4. BOOST_VDAC_DATA1_REG

Bit	7	6	5	4	3	2	1	0
Value	RefLow							

RefLow determines the reference low DAC output voltage. The initial value of this bit is set with the value of the HCT_RefLow parameter in the Device Editor. The value can be changed at runtime with the BOOST_SetRefLow() API.

Table 5. BOOST_HYST_CONTROL_REG

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	TimerDelay		Trip	Enable

Enable enables or disables the user module. It is modified by calling BOOST_Start() or BOOST_Stop().

Trip determines the hysteretic controller tripping ability, also can be changed with HCT_TripInput.

TimerDelay determines the monoshot timer delay for both on and off timers. The initial value of this bit is set with the value of the HCT_TimerDelay parameter in the Device Editor..

Table 6. BOOST_CMP_CONTROL_REG

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	Enable	0	0	0	Enable

Enable enables or disables the comparator block (even and odd) in the hysteretic channel. It is modified by calling BOOST_Start() or BOOST_Stop().

Table 7. BOOST_GATE_CONTROL_REG

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	GateDriverStrength		GateDriver	

GateDriver selects the active FET gate driver. The initial value of this bit is set with the value of the HCT_GateDriver parameter in the Device Editor.

GateDriverStrength selects the FET gate driver strength. The initial value of this bit is set with the value of the HCT_GateDriverStrength parameter in the Device Editor.

PGA Registers

Table 8. BOOST_GAIN_CR0

Bit	7	6	5	4	3	2	1	0
Value	Gain					1	Reference	

Gain sets the gain value. The initial value of this bit is set with the value of the PGA_Gain parameter in the Device Editor. Also the value can be changed at runtime with the BOOST_SetPGAGain() API. Reference sets the reference point (effective "ground") for gain..

Table 9. BOOST_GAIN_CR1

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	0	0	Input		

Input is the value selected in PSoC Designer (AnalogColumn_InputSelect_1 or AnalogMUXBus1)..

Table 10. BOOST_GAIN_CR2

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	0	0	0	0	0

Power is set to `Off' following device reset and configuration. It is modified by calling BOOST_Start() or BOOST_Stop() functions in the API..

Table 11. BOOST_GAIN_CR3

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	EXGAIN

The EXGAIN bit is automatically set when ever a gain of 24 or 48 is selected.

ADC Block Registers.

Table 12. BOOST_ADC1_CR0

Bit	7	6	5	4	3	2	1	0
Value	1	0	0	0	1	0	0	0

Table 13. BOOST_ADC2_CR0

Bit	7	6	5	4	3	2	1	0
Value	1	0	0	1	0	0	0	0

Table 14. BOOST_ADC1_CR1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 15. BOOST_ADC2_CR1

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 16. BOOST_ADC1_CR2

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	0	0	0	0	0

Table 17. BOOST_ADC2_CR2

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	0	0	0	0	0

Table 18. BOOST_ADC1_CR3

Bit	7	6	5	4	3	2	1	0
Value	1	1	1	1	1	0	0	0

Table 19. BOOST_ADC2_CR3

Bit	7	6	5	4	3	2	1	0
Value	1	1	1	1	1	0	0	0

PWM Block Registers

Table 20. BOOST_PWM_DR2

Bit	7	6	5	4	3	2	1	0
Value	PulseWidth							

PulseWidth sets the PWM pulse width. The initial value of this bit is set with the value of the ADC_PulseWidth parameter in the Device Editor. The value can be changed at runtime with the BOOST_WritePulseWidth() API.

Table 21. BOOST_PWM_FN

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	1	0	0	0	1

Version History

Version	Originator	Description
1.0	FRE	Changed P-Channel N-Channel in the sentence: "The user module is designed to work with internal and high power P-Channel FETs."
1.0.b	DHA	Added wizard help file.
1.0.c	DHA	Moved BOOST User Module to Legacy status.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2009-2012 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.