Hysteretic Controller Datasheet HYSTCTRL V 1.0

Copyright © 2009-2014 Cypress Semiconductor Corporation. All Rights Reserved.

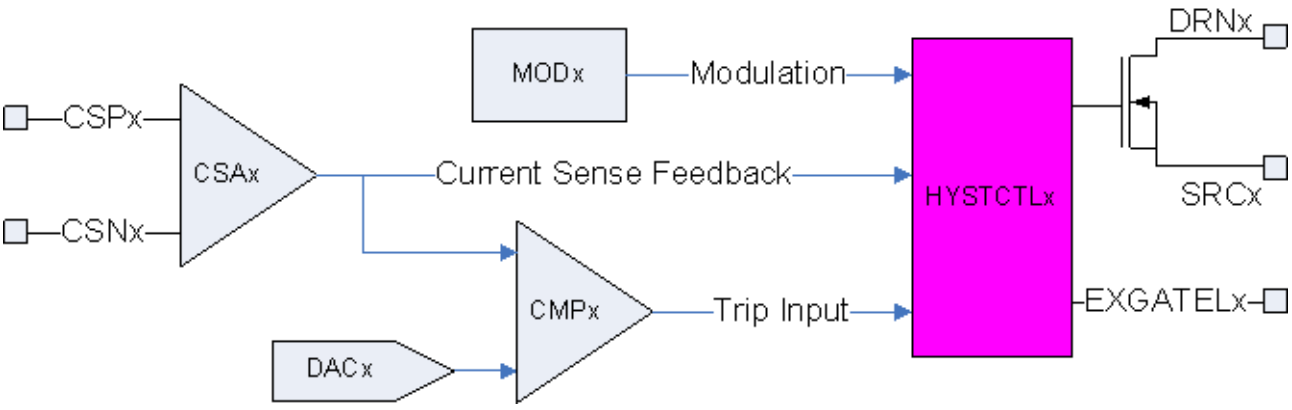
Resources	PSoC® Blocks	API Memory (Bytes)		Pins (per External I/O)
	HYSTCTRL + FET	Flash	RAM	
CY8CLED0xD, CY8CLED0xG	1	166	0	3-6

Features and Overview

- DAC configurable thresholds
- 20 kHz to 2 MHz effective switching frequency range
- Programmable Minimum on and off time
- Selectable drive strength

The HYSTCTRL User Module is intended for the use in LED applications as an intelligent controller for high brightness LEDs. It provides cycle-by-cycle switch control with a fast transient response. The hysteretic controller simplifies system design because it requires no external compensation. The gate drivers are used to drive either internal or external power FETs.

Figure 1. Hysteretic Controller Block Diagram



Functional Description

The HYSTCTRL User Module is a typical switching regulator. It provides output current in specified ranges and performs one of the three assignable kinds of pulse-width modulation. It also features an over current detection mechanism that disables the hysteretic controller if the input current exceeds the allowable maximum.

The HYSTCTRL User Module consists of two hardware blocks; a Hysteretic Controller and a Gate Driver that drives a FET.

The Gate Driver is capable of driving either the internal power FET or an external power FET. The driver strength for both gate drivers is configurable.

The current monitoring is done with voltage tracking. The voltage operation range is also configurable and can be 1.3V or 2.6V.

The HYSTCTRL User Module gets feedback input from either the current sense amplifier output or from an external source via FN_0_x.

The block diagram shows the internal structure of the HYSTCTRL User Module. It is driven by three functions:

- Main Control Loop Function
- DIM Function
- Trip Function

Main Control Loop Function

Hysteretic Control Loop consists of two DACs, two comparators, two timers, an RS-trigger, and some logic.

The 8-bit DACs that are internal to this user module are used to set the reference for the lower and upper hysteretic comparators. Refer to the Dual 8-bit Hardware DAC datasheet for more information about their characteristics.

The timers are retriggerable monoshots that are triggered at the rising edge of the input clock.

Output is generated by comparing the feedback value to the two thresholds. Going below the lower threshold turns the switch on, and exceeding the upper threshold turns the switch off. Programmable minimum on-time and off-time timing circuits prevent high frequency oscillations that can be destructive to the output switches.

DIM Function

The DIM input is a separate input signal from a modulation block (MOD) or from an external source (FN_0_x port). The composite signal is generated by ANDing the hysteretic controller outputs with the modulation scheme. This is useful for driving high brightness LED circuits, where a high frequency control must be gated by a low frequency pulse-width modulation for dimming the visible brightness of the LEDs.

In the HYSTCTRL User Module the following modulations are provided:

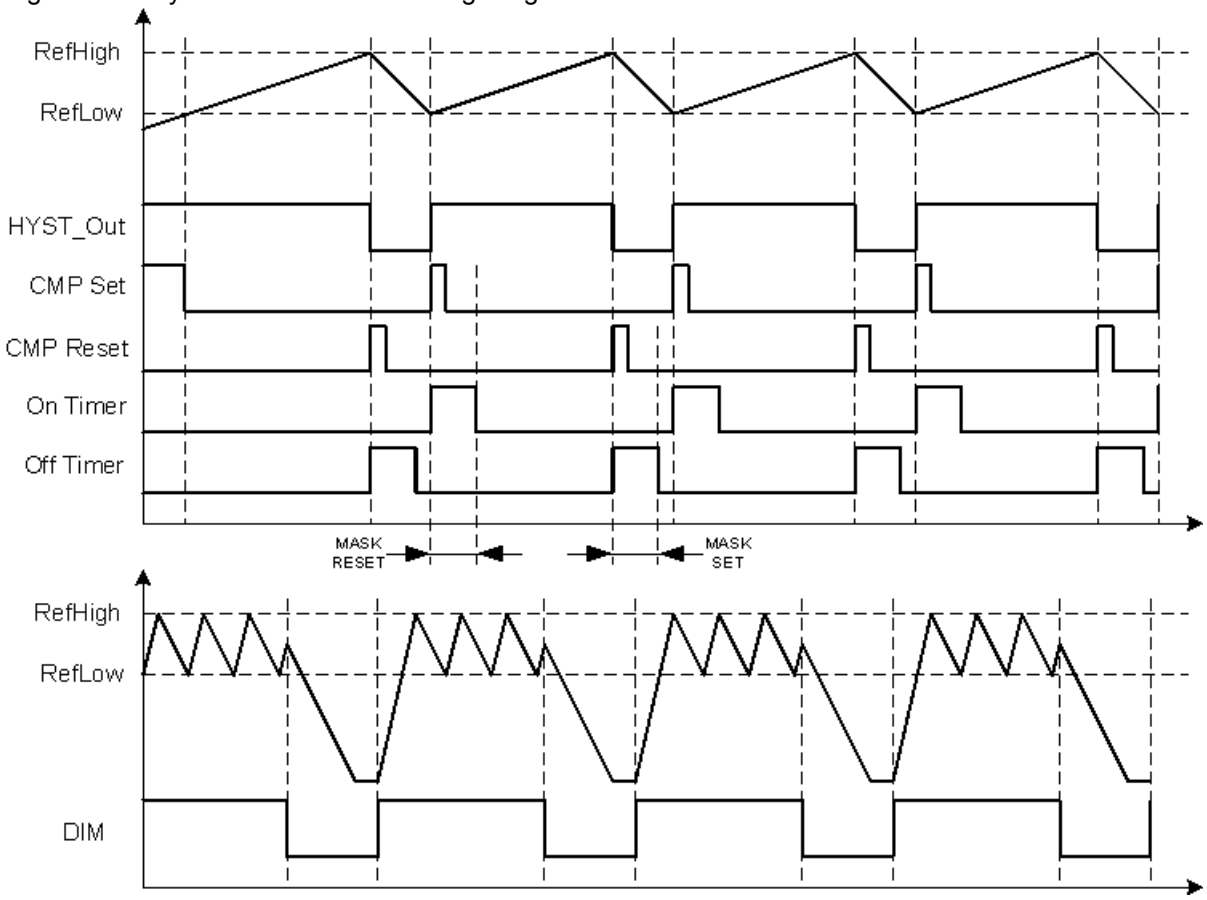
- Pulse width modulation (PWM mode)
- Density modulation (DMM mode)
- Stochastic signal density modulation (PrISM™ mode)

For information about these modulation mechanisms, refer to the PWM16HW, DMM16HW or PrISM16HW User Module datasheets.

Trip Function

The trip function is generated from the comparator bank or can be taken from FN_0_x source. It notifies the hysteretic controller that an over current condition has occurred at the output. Trip, like DIM, overrides normal hysteretic controller main loop operation. This is implemented by ANDing Trip with DIM, main loop output and hysteretic controller enable signal.

Figure 2. Hysteretic Controller Timing Diagram



The DIM waveform (Figure 2) turns the switch off to control brightness (dimming) and is independent of the hysteretic control loop.

DC and AC Electrical Characteristics

See the device characterization data in the DC and AC Electrical Characteristics section of the device datasheet.

Placement

The HYSTCTRL User Module can be placed on any of the HYSTCTRL blocks. HYSTCTRL also uses a FET from the same row as the HYSTCTRL block:

- HYSTCTRL0 + FET0
- HYSTCTRL1 + FET1
- HYSTCTRL2 + FET2
- HYSTCTRL3 + FET3

Parameters and Resources

The following parameters configure HYSTCTRL User Module.

FeedbackInput

Defines the HYSTCTRL User Module Feedback Input connection. The choices are one of the CSA block outputs or FN0 pins. The specific CSA block or FN_0_x pin available depends on placement of the user module.

DACVoltageRange

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

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 1

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

Equation 2

$$SenseVoltage = PeakCurrent \times SenseResistor$$

Equation 3

$$DACVoltage = SenseVoltage \times CSAGain$$

Equation 4

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

In the last formula the VoltageRange is the value of DACVoltageRange parameter (1.3V or 2.6V). For example, a LED application intended to drive 350 mA of constant current through the LEDs using a 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 RefHigh DAC value is computed as follows:

Equation 5

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

Equation 6

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

Equation 7

$$DACVoltage(V) = 80.85 \times 20 = 1.617$$

Equation 8

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

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

RefLow

Configures 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 9

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

Equation 10

$$SenseVoltage = ValleyCurrent \times SenseResistor$$

Equation 11

$$DACVoltage = SenseVoltage \times CSAGain$$

Equation 12

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

For example, an LED application intended to drive 350 mA of constant current through the LEDs using a 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 RefLow DAC value is computed as follows:

Equation 13

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

Equation 14

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

Equation 15

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

Equation 16

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

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

DimInput

Selects the HYSTCTRL User Module DIM input connection. It is possible to connect one of the MOD block outputs or FN_0_x. A MOD block can provide a PrISM, PWM, or DMM modulated signal as the DIM input.

TripInput

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

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.

GateDriver

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

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 HYSTCTRL_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 HYSTCTRL 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.

HYSTCTRL_Start

Description:

Starts the HYSTCTRL User Module. The output FET is open until HYSTCTRL_Start is called.

C Prototype:

```
void HYSTCTRL_Start(void);
```

Assembler:

```
lcall HYSTCTRL_Start
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

HYSTCTRL_Stop

Description:

Stops the HYSTCTRL User Module. The output FET is open when HYSTCTRL_Stop is called. When the FET is off, the HYSTCTRL User Module output is low.

C Prototype:

```
void HYSTCTRL_Stop();
```

Assembler:

```
lcall HYSTCTRL_Stop
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

HYSTCTRL_SetDACVoltageRange

Description:

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

C Prototype:

```
void HYSTCTRL_SetDACVoltageRange(BYTE bVoltageRange);
```

Assembler:

```
mov   A, bVoltageRange
lcall  HYSTCTRL_SetDACVoltageRange
```

Parameters:

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
HYSTCTRL_DAC_1_3V	0x02	Sets the reference DAC voltage range from 0V to 1.3V.
HYSTCTRL_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.

HYSTCTRL_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 HYSTCTRL_SetRefHigh(BYTE bData);
```

Assembler:

```
mov   A, bData
lcall  HYSTCTRL_SetRefHigh
```

Parameters:

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.

HYSTCTRL_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 HYSTCTRL_SetRefLow(BYTE bData);
```

Assembler:

```
mov   A, bData
lcall  HYSTCTRL_SetRefLow
```

Parameters:

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.

HYSTCTRL_EnableHystTripping

Description:

Enables the hysteretic tripping operation. The trip function is generated from the comparator bank through a digital mux. It notifies the hysteretic controller that an over current condition has occurred at the output.

C Prototype:

```
void HYSTCTRL_EnableHystTripping(void);
```

Assembler:

```
lcall HYSTCTRL_EnableHystTripping
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

HYSTCTRL_DisableHystTripping

Description:

Disables the hysteretic tripping operation.

C Prototype:

```
void HYSTCTRL_DisableHystTripping(void);
```

Assembler:

```
lcall HYSTCTRL_DisableHystTripping
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

HYSTCTRL_ResetTrip

Description:

Resumes Hysteretic operation after Trip condition.

C Prototype:

```
void HYSTCTRL_ResetTrip(void);
```

Assembler:

```
lcall HYSTCTRL_ResetTrip
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

HYSTCTRL_SetGateDriver

Description:

Configures the operation of the internal and external gate drivers.

C Prototype:

```
void HYSTCTRL_SetGateDriver(BYTE bSelect);
```

Assembler:

```
mov   A, bSelect
lcall  HYSTCTRL_SetGateDriver
```

Parameters:

bSelect - specifies the enabled gate driver. Symbolic names provided in C and assembly, and their associated values, are given in the following table:

Symbolic Name	Value	Description
HYSTCTRL_GDSEL_NONE	0x00	Disable both drivers.
HYSTCTRL_GDSEL_EXTERNAL	0x01	Enable the external driver only.
HYSTCTRL_GDSEL_INTERNAL	0x02	Enable the internal driver only.

Return Value:

None.

Side Effects:

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

HYSTCTRL_SetGateDriverStrength

Description:

Sets the drive strength of the gate driver.

C Prototype:

```
void HYSTCTRL_SetGateDriverStrength(BYTE bStrength);
```

Assembler:

```
mov   A, bStrength
lcall  HYSTCTRL_SetGateDriverStrength
```

Parameters:

bStrength - specifies the gate driver strength. Symbolic names provided in C and assembly, and their associated values, are given in the following table:

Symbolic Name	Value	Description
HYSTCTRL_STRENGTH_DEFAULT	0x00	Default drive strength.
HYSTCTRL_STRENGTH_75PC	0x04	75% of the default strength.
HYSTCTRL_STRENGTH_50PC	0x08	50% of the default strength.
HYSTCTRL_STRENGTH_25PC	0x0C	25% of the default strength.

Return Value:

None.

Side Effects:

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

HYSTCTRL_SetTimerDelay

Description:

Sets the monoshot timer delay for both the on and off timers.

C Prototype:

```
void HYSTCTRL_SetTimerDelay(BYTE bDelay);
```

Assembler:

```
mov   A, bDelay
lcall  HYSTCTRL_SetTimerDelay
```

Parameters:

bDelay - specifies the timer delay. Symbolic names provided in C and assembly, and their associated values, are given in the following table:

Symbolic Name	Value	Description
HYSTCTRL_DELAY_OFF	0x0C	No delay from either timer.
HYSTCTRL_DELAY_10_25NS	0x00	10 to 25 ns timer delay for both the on and off timers.
HYSTCTRL_DELAY_20_50NS	0x04	20 to 50 ns timer delay for both the on and off timers.
HYSTCTRL_DELAY_40_100NS	0x08	40 to100 ns timer delay for both the on and off timers.

Return Value:

None.

Side Effects:

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

Sample Firmware Source Code

The C code illustrated here shows you how to use the HYSTCTRL User Module.

```
WORD i;
   HYSTCTRL_Start();

   while(1)
   {
     HYSTCTRL_SetRefHigh(95);
   HYSTCTRL_SetRefLow(80);

   for(i = 0; i < 20000; i++);  //Delay

   HYSTCTRL_SetRefLow(20);
   HYSTCTRL_SetRefHigh(30);

     for(i = 0; i < 20000; i++);  //Delay
```

```
    }
```

The same code in assembly is:

```
    call HYSTCTRL_Start
.loop:
    mov  A, 95
    call HYSTCTRL_SetRefHigh
    mov  A, 80
    call HYSTCTRL_SetRefLow

  ;Delay
    mov X, FFh
.Ext_Loop:
    mov A, FFh
.Int_Loop:
  dec A
    jnz .Int_Loop
    dec X
    jnz .Ext_Loop

  mov  A, 20
    call HYSTCTRL_SetRefLow
    mov  A, 30
    call HYSTCTRL_SetRefHigh

  ;Delay
    mov X, FFh
.Ext_Loop2:
    mov A, FFh
.Int_Loop2:
  dec A
    jnz .Int_Loop2
    dec X
    jnz .Ext_Loop2

  jmp .loop
```

Configuration Registers

The HYSTCTRL User Module is personalized and parameterized through the registers. The following tables shows the register values as constants and the parameters as named bit fields with brief descriptions.

Table 1. HYSTCTRL_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 HYSTCTRL_Start or HYSTCTRL_Stop.

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

Table 2. HYSTCTRL_VDAC_DATA0_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 RefLow parameter in the Device Editor. The value can be changed at runtime with the HYSTCTRL_SetRefLow API.

Table 3. HYSTCTRL_VDAC_DATA1_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 RefHigh parameter in the Device Editor. The value can be changed at runtime with the HYSTCTRL_SetRefHigh API.

Table 4. HYSTCTRL_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 HYSTCTRL_Start or HYSTCTRL_Stop.

Trip determines the hysteretic controller tripping ability and can be changed at runtime with the HYSTCTRL_EnableHystTripping API.

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 TimerDelay parameter in the Device Editor. The value can be changed at runtime with the HYSTCTRL_SetTimerDelay API.

Table 5. HYSTCTRL_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 HYSTCTRL_Start or HYSTCTRL_Stop.

Table 6. HYSTCTRL_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 GateDriver parameter in the Device Editor. The value can be changed at runtime with the HYSTCTRL_SetGateDriver API.

GateDriverStrength selects the FET gate driver strength. The initial value of this bit is set with the value of the GateDriverStrength parameter in the Device Editor. The value can be changed at runtime with the HYSTCTRL_SetGateDriverStrength API.

Version History

Version	Originator	Description
1.0	DHA	Added ResetTrip API to HYSTCTRL User Module.

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

Copyright © 2009-2014 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.