Document Number: PEXMC12XSF-MC12XS6PUG
Rev. 1.0, 5/2016

Gen4eXtremeSwitch programming guide

Contents

1 General Info

Gen4eXtremeSwitch Processor Expert component is a software driver which encapsulates functionality of MC12XSF and MC12XS6 eXtreme switch families. The component creates a layer between hardware and user application and enables rapid application development by providing an interface which covers options for settings of registers, measurement and testing.

The NXP MC12XS6/F/G eXtreme switch products belong to an expanding family that can control and diagnose various types of loads, such as incandescent bulbs or light emitting diodes (LEDs), with enhanced precision. 12XS6/F/G families combine flexibility through daisy chainable SPI at 5.0 MHz, extended digital and analog feedback, which supports safety and robustness. The user can configure the initial configuration of the device in the component and generates a set of functions for operating the switch via SPI.

2 Embedded Component Description

2.1 Component API

Gen4eXtremeSwitch component provides API, which can be used for dynamic real-time configuration of device in user code. Available methods and events are listed under component selection Some of those methods/events are marked with ticks and other ones with crosses, it distinguishes which methods/events are supposed to be generated or not. You can change this setting in Processor Expert Inspector. Note that methods with grey text are always generated because they are needed for proper functionality. This forced behavior depends on various combinations of settings of component properties. For summarization of available API methods and events and their descriptions, see Table 1 Gen4eXtremeSwitch Component API

Table 1

Method	Description
Init	Initializes the device according to the component properties. This method writes the data gathered from the component properties into registers via SPI. When auto initialization is enabled, this method will be called automatically within PE initialization function PE_low_level_init().
Deinit	Deinitializes the device (puts the device in sleep mode).
GetQuickStatus	Returns current status of the device. Reads the quick status register which provides glance at failure overview. As long as no failure flag is set (logic 1), no control action by the microcontroller is necessary. Meaning of bits in quick status register can be found in datasheet or header file.
GetChannelStatus	Gets the information of the fault for all available channels. Reads information from CHx status register. Meaning of bits in CHx status register can be found in datasheet or header file.
GetDeviceStatus	Gets information on the status of the device. Reads information from device status register. Meaning of bits in device status register can be found in datasheet or header file.
GetIOStatus	Gets information on the I/O of the device. Reads information from I/O status register. Meaning of bits in I/O status register can be found in datasheet or header file.
ReadRegister	Reads value of the given register via SPI. This method allows the user to read content from a register of the device(s).
WriteRegister	Writes value to the given register via SPI. This method allows the user to set a custom value to a register of the device.
FeedWatchdog	Toggles the watchdog bit in all devices to avoid watchdog timeout. Watchdog is fed with every SPI write. If there is not SPI communication it is necessary to feed it before the timeout expires. The watchdog timeout depends on WD SEL bit in Intialization register 1. The timeout is either 32 ms or 128 ms.

SetOutputState	Turns on/off channels of the device.
SetPWMDuty	Sets PWM duty for specified channel. Calling this method will set PWM output immediately (if PWM is enabled). This method writes PWM value into CHx register.
SetPWMDutyValues	Sets PWM duty for all available channels of the device. Calling this method will set PWM output immediately (if PWM is enabled), otherwise the setting will not take effect until after PWM enabling with method SetOutputState.
SetGPWMDuty	Sets Global PWM duty cycle value. Calling this method will set PWM output immediately (if PWM is enabled). This method writes into global register.
SetPWMControl	Sets PWM output to Global or individual PWM control.
IncrementalPWMControl	This method sets LSB step according to which the duty cycle will be incremented or decremented with every call of this method.
SetOCHIOnDemand	This method enables the OCHI On Demand (over current high on demand) feature on corresponding channel. This feature enables higher tolerance of the driver to inrush current occuring when the load is reconnected after sudden power disconnection.
OLLEDTrig	This method triggers the Openload functionality for LED. The open load detection for LED is useful for detection of small load currents (e.g. LED) in on state of the switch. This feature implements a special low current detection mode. The OLLED fault is reported when the output voltage is above VPWR 0.75V after 2.0ms offtime or at each turnon command in case of offtime 2.0ms. The OLLED TRIG bit is reset after the detection.
SetOLOFF	This method enables the OLOFF (open load detection in off state) for corresponding channel. When the detection is started, the corresponding output channel is turned on with a fixed overcurrent threshold. When this overcurrent threshold is reached within the detection timeout tOLOFF, the output is turned off and the OLOFF EN bit is reset. No error is reported. If the threshold is not reached within the detection timeout tOLOFF, the output is turned off after tOLOFF and the OLOFF EN bit is reset. The open load in off state event is reported.
SetDirectInput	Sets target pin INx high/low voltage. The direct input is used to directly control corresponding channel in Fail mode. During Normal mode the control of the outputs by the control inputs is SPI programmable and the output voltage depends on set duty (global or individual). Direct inputs are only Outputs 1..4.
SetDirectInputValues	Sets target pin INx high/low voltage for all channels simultaneously. The direct input is used to directly control corresponding channel in Fail mode. During Normal mode the control of the outputs by the control inputs is SPI programmable. Direct inputs are only Outputs 1..4.
SetExtClockState	This method enables/disables external clock timer.
ConfigureMonitoring	Configures CSNS pin output. This method sets monitored parameter and synchronization.
GetSenseValue	Configures monitoring of current, voltage or temperature on the CSNS pin and returns the corresponding value from the ADC. Waits until measurement is completed.

2.2 Events

OnMeasurementSynchronization - This event is invoked when measurement synchronization trigger occurs.

*ANSIC prototype:*void OnMeasurementSynchronization(void)

2.3 Methods

Init - Initializes the device according to the component properties. This method writes the data gathered from the component properties into registers via SPI. When auto initialization is enabled, this method will be called automatically within PE initialization function - PE_low_level_init().

ANSIC prototype: TDeviceDataPtr Init(TUserDataPtr UserDataPtr)
UserDataPtr:TUserDataPtr - User data pointer
Return value:TError - Error code, ERR_OK if successful

Deinit - Deinitializes the device (puts the device in sleep mode).

ANSIC prototype: void Deinit(void)
Return value:void

GetQuickStatus - Returns current status of the device. Reads the quick status register which provides glance at failure overview. As long as no failure flag is set (logic 1), no control action by the microcontroller is necessary. Meaning of bits in quick status register can be found in datasheet or header file.

ANSIC prototype: TError GetQuickStatus(TDeviceIdx DeviceIndex,uint16_t *StatusData)
DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).
StatusData: Pointer to uint16_t - Pointer to variable that will be filled with the status information.
Return value:TError - Error code, ERR_OK if successful

GetChannelStatus - Gets the information of the fault for all available channels. Reads information from CHx status register. Meaning of bits in CHx status register can be found in datasheet or header file.

ANSIC prototype: TError GetChannelStatus(TDeviceIdx DeviceIndex,uint16_t *StatusData)
DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).
StatusData: Pointer to uint16_t - Pointer to an array that will be filled with the fault information.
Return value:TError - Error code, ERR_OK if successful

GetDeviceStatus - Gets information on the status of the device. Reads information from device status register. Meaning of bits in device status register can be found in datasheet or header file.

ANSIC prototype: TError GetDeviceStatus(TDeviceIdx DeviceIndex,uint16_t *StatusData)
DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).
StatusData: Pointer to uint16_t - Pointer to variable that will be filled with the fault information.
Return value:TError - Error code, ERR_OK if successful

GetIOStatus - Gets information on the I/O of the device. Reads information from I/O status register. Meaning of bits in I/O status register can be found in datasheet or header file.

ANSIC prototype: TError GetIOStatus(TDeviceIdx DeviceIndex,uint16_t *StatusData)
DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).
StatusData: Pointer to uint16_t - Pointer to variable that will be filled with the fault information.
Return value:TError - Error code, ERR_OK if successful

ReadRegister - Reads value of the given register via SPI. This method allows the user to read content from a register of the device(s).

ANSIC prototype: TError ReadRegister(TDeviceIdx DeviceIndex,TSORegister Register,uint16_t *RegVal)
DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).
Register:TSORegister - Address of the register.
RegVal: Pointer to uint16_t - Pointer to the variable for resulting data.
Return value:TError - Error code, ERR_OK if successful

WriteRegister - Writes value to the given register via SPI. This method allows the user to set a custom value to a register of the device.

ANSIC prototype: TError WriteRegister(TDeviceIdx DeviceIndex,TSIRegister Register,uint16_t RegVal)

DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).

Register:TSIRegister - Address of the register.

RegVal:uint16_t - Pointer to the value to be written.

Return value:TError - Error code, ERR_OK if successful

FeedWatchdog - Toggles the watchdog bit in all devices to avoid watchdog timeout. Watchdog is fed with every SPI write. If there is not SPI communication it is necessary to feed it before the timeout expires. The watchdog timeout depends on WD SEL bit in Intialization register 1. The timeout is either 32 ms or 128 ms.

ANSIC prototype: TError FeedWatchdog(void)

Return value:TError - Error code, ERR_OK if successful

SetOutputState - Turns on/off channels of the device.

ANSIC prototype: TError SetOutputState(TDeviceIdx DeviceIndex,bool *OutputStates)

DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).

OutputStates: Pointer to bool - Pointer to array containing states of available channels (TRUE = enabled, FALSE = disabled).

Return value:TError - Error code, ERR_OK if successful

SetPWMDuty - Sets PWM duty for specified channel. Calling this method will set PWM output immediately (if PWM is enabled). This method writes PWM value into CHx register.

ANSIC prototype: TError SetPWMDuty(TDeviceIdx DeviceIndex,uint8_t Channel,uint16_t Duty)

DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).

Channel:uint8_t - Output channel index.

Duty:uint16_t - PWM duty value for selected channel.

The range of the duty is 0..256, where 0 = OFF and 1 = 0.4% duty and 256 = 100%.

Return value:TError - Error code, ERR_OK if successful

SetPWMDutyValues - Sets PWM duty for all available channels of the device. Calling this method will set PWM output immediately (if PWM is enabled), otherwise the setting will not take effect until after PWM enabling with method SetOutputState.

ANSIC prototype: TError SetPWMDutyValues(TDeviceIdx DeviceIndex,uint16_t *DutyValues)

DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).

DutyValues: Pointer to uint16_t - Pointer to an array that contains the PWM duty values.

The range of the duty is 0..256, where 0 = OFF and 1 = 0.4% duty and 256 = 100%.

Return value:TError - Error code, ERR_OK if successful

SetGPWMDuty - Sets Global PWM duty cycle value. Calling this method will set PWM output immediately (if PWM is enabled). This method writes into global register.

ANSIC prototype: TError SetGPWMDuty(TDeviceIdx DeviceIndex,uint16_t Duty)

DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).

Duty:uint16_t - Global PWM duty value for selected device.

The range of the duty is 0..256, where 0 = OFF and 1 = 0.4% duty and 256 = 100%.

Return value:TError - Error code, ERR_OK if successful

SetPWMControl - Sets PWM output to Global or individual PWM control.

ANSIC prototype: TError SetPWMControl(TDeviceIdx DeviceIndex,uint8_t Channel,TChannelControl Control)

DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).

Channel:uint8_t - Output channel index.

Control:TChannelControl - Individual state of the PWM type control for each single output. The value is ccGLOBAL if a channel is supposed to be controlled by global PWM register or ccINDIVIDUAL if individually.

Return value:TError - Error code, ERR_OK if successful

IncrementalPWMControl - This method sets LSB step according to which the duty cycle will be incremented or decremented with every call of this method.

ANSIC prototype: TError IncrementalPWMControl(TDeviceIdx DeviceIndex,TIDSign Sign,TLSBStep *LSBStep)

DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).

Sign:TIDSign - idsPOSITIVE = increment, idsNEGATIVE = decrement.

LSBStep: Pointer to TLSBStep - Array for available channels with step of increment/decrement (idsNONE, ids4LSB, ids8LSB, ids16LSB).

Return value:TError - Error code, ERR_OK if successful

SetOCHIOnDemand - This method enables the OCHI On Demand (over current high on demand) feature on corresponding channel. This feature enables higher tolerance of the driver to inrush current occuring when the load is reconnected after sudden power disconnection.

ANSIC prototype: TError SetOCHIOnDemand(TDeviceIdx DeviceIndex,uint8_t Channel)

DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).

Channel:uint8_t - Index of channel to be more tolerant to inrush current. It can be individual channel or all at once.

Return value:TError - Error code, ERR_OK if successful

OLLEDTrig - This method triggers the Openload functionality for LED. The open load detection for LED is useful for detection of small load currents (e.g. LED) in on state of the switch. This feature implements a special low current detection mode. The OLLED fault is reported when the output voltage is above VPWR - 0.75V after 2.0ms off-time or at each turn-on command in case of off-time <2.0ms. The OLLED TRIG bit is reset after the detection.

ANSIC prototype: TError OLLEDTrig(TDeviceIdx DeviceIndex)

DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).

Return value:TError - Error code, ERR_OK if successful

SetOLOFF - This method enables the OLOFF (open load detection in off state) for corresponding channel. When the detection is started, the corresponding output channel is turned on with a fixed overcurrent threshold. When this overcurrent threshold is reached within the detection timeout tOLOFF, the output is turned off and the OLOFF EN bit is reset. No error is reported. If the threshold is not reached within the detection timeout tOLOFF, the output is turned off after tOLOFF and the OLOFF EN bit is reset. The open load in off state event is reported.

ANSIC prototype: TError SetOLOFF(TDeviceIdx DeviceIndex,uint8_t Channel)

DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).

Channel:uint8_t - Output channel index.

Return value:TError - Error code, ERR_OK if successful

SetDirectInput - Sets target pin INx high/low voltage. The direct input is used to directly control corresponding channel in Fail mode. During Normal mode the control of the outputs by the control inputs is SPI programmable and the output voltage depends on set duty (global or individual). Direct inputs are only Outputs 1..4.

ANSIC prototype: TError SetDirectInput(uint8_t Input,bool Value)

Input:uint8_t - Direct input number (1..4).

Value:bool - Level on INx pin, TRUE/FALSE.

Return value:TError - Error code, ERR_OK if successful

SetDirectInputValues - Sets target pin INx high/low voltage for all channels simultaneously. The direct input is used to directly control corresponding channel in Fail mode. During Normal mode the control of the outputs by the control inputs is SPI programmable. Direct inputs are only Outputs 1..4.

ANSIC prototype: TError SetDirectInputValues(bool *InputValues)

InputValues: Pointer to bool - Pointer to array with levels on IN1-4 pins (TRUE = high, FALSE = low).

Return value:TError - Error code, ERR_OK if successful

SetExtClockState - This method enables/disables external clock timer.

ANSIC prototype: TError SetExtClockState(TExtClkState ClkState)

ClkState:TExtClkState - ecsENABLED = timer is on, ecsDISABLED = timer is off.

Return value:TError - Error code, ERR_OK if successful

ConfigureMonitoring - Configures CSNS pin output. This method sets monitored parameter and synchronization.

ANSIC prototype: TError ConfigureMonitoring(TDeviceIdx DeviceIndex,TSenseMux SenseMux,TSenseSyncTrigger Trigger)

DeviceIndex:TDeviceIdx - Index of device (didxDEV1..4).

SenseMux:TSenseMux - Measured value (CSNS pin function).

Trigger:TSenseSyncTrigger - Synchronization trigger.

Return value:TError - Error code, ERR_OK if successful

GetSenseValue - Configures monitoring of current, voltage or temperature on the CSNS pin and returns the corresponding value from the ADC. Waits until measurement is completed.

ANSIC prototype: TError GetSenseValue(uint16_t *Result)

Result: Pointer to uint16_t - Pointer to variable where the measured data will be stored.

Return value:TError - Error code, ERR_OK if successful

2.4 Properties

Component Name - Name of the component.

SPI Configuration - Configuration of SPI communication with devices.

There are 2 options:

Parallel SPI: Parallel SPI (common MISO, MOSI, SCLK, separate CS pins) with common RST, CLK, CSNS and CSNS SYNC pins. For parallel SPI with these pins separated use more instances of this component.

Daisy Chain SPI: Devices connected in daisy chain (the first MISO being connected to the second MOSI etc., common CS pin) with common RST, CLK, CSNS and CSNS SYNC pins. Daisy chain SPI with these pins separated is not supported.

Global Configuration - Shared settings for all channels on all devices.

RSTB Link Linked BitIO_LDD component.

RSTB Pin This input pin is used to initialize the device configuration and fault registers (when high), as well as place the device in a low-current Sleep mode (low).

CLK Link Linked PWM_LDD component.

External Clock Frequency Reference PWM clock (in Hz) in frequency range 25.6 - 102.4 kHz which is then divided by 256 in Normal operating mode.

CLK Pin This pin used to apply reference PWM clock which is divided by 256 in Normal operating mode.

Watchdog Timeout Watchdog is used for SPI communication loss detection. SPI communication fault is detected if the WD bit is not toggled with each SPI message or WD timeout is reached. If a SPI communication error occurs, the device is switched into Fail mode.

There are 2 options:

32 ms: Short watchdog timeout.

128 ms: Long watchdog timeout.

Direct Input Control Direct input pins are used to directly control corresponding channel in Fail mode. During Normal mode the control of the outputs by the control inputs is SPI programmable and the output voltage depends on set duty (global or individual). Initial state of direct input pins can be set in corresponding BitIO_LDD component (default 0).

There are 2 modes:

Enabled - Feature is enabled. The following items are displayed in this mode:

IN1 Pin - Direct input pin. Set initial state in IN1Pin BitIO_LDD component (default 0).

IN1 Pin Link - Linked BitIO_LDD component.

IN2 Pin - Direct input pin. Set initial state in IN2Pin BitIO_LDD component (default 0).

IN2 Pin Link - Linked BitIO_LDD component.

IN3 Pin - Direct input pin. Set initial state in IN3Pin BitIO_LDD component (default 0).

IN3 Pin Link - Linked BitIO_LDD component.

IN4 Pin - Direct input pin. Set initial state in IN4Pin BitIO_LDD component (default 0).

IN4 Pin Link - Linked BitIO_LDD component.

Disabled - Feature is disabled. The following items are displayed in this mode:

Current, Voltage and Temperature Sensing The analog feedback circuit is implemented to provide load and device diagnostics during Normal mode. The feedback monitor provides a current proportional to the current of selected output, voltage proportional to the battery supply voltage or output voltage of temperature sensor monitoring avreage control die temperature.

There are 2 modes:

Enabled - Feature is enabled. The following items are displayed in this mode:

CSNS Synchronization - Sensing synchronization is provided to simplify synchronous sampling of the CSNS signal.

There are 4 options:

Off: CSNS SYNC is inactive (high).

Valid: CSNS SYNC is active (low) when CSNS is valid.

Trigger 0: CSNS SYNC is active (low) when CSNS is valid, then inactive (high) until the next PWM cycle is started.

Trigger 1/2: Pulses (active low) from the middle of the CSNS pulse to its end are generated.

CSNS Pin - ADC pin used for measuring analog value reported on feedback pin.

CSNS Function - Initial feedback monitoring selection for device 1 (selection and device can be changed while application is running by method ConfigureMonitoring).

There are 7 options:

Channel 1: Channel 1 current monitoring.

Channel 2: Channel 2 current monitoring.

Channel 3: Channel 3 current monitoring.

Channel 4: Channel 4 current monitoring.

Channel 5: Channel 5 current monitoring.

Battery voltage: Battery voltage monitoring.

Temperature: Average temperature of the control die monitoring.

ADC Conversion Time - Duration of one ADC conversion.

ADC Link - Linked ADC_LDD component.

Disabled - Feature is disabled. The following items are displayed in this mode:

Devices - Number of devices connected to MCU.

Device - Properties for device.

Device Model - Model of eXtreme Switch device (number of displayed outputs depends on this property).

There are 14 options:

MC07XSF517: Penta High-Side Switch (2 x 17 mOhm, 3 x 7 mOhm RDSon)

MC08XSF421: Quad High-Side Switch (2 x 21 mOhm, 2 x 8 mOhm RDSon)

MC17XSF400: Quad High-Side Switch (4 x 17 mOhm RDSon)

MC17XSF500: Penta High-Side Switch (5 x 17 mOhm RDSon)

MC40XSF500: Penta High-Side Switch (5 x 40 mOhm RDSon)
MC07XS6517: Penta High-Side Switch (2 x 17 mOhm, 3 x 7 mOhm RDSon)
MC08XS6421: Quad High-Side Switch (2 x 21 mOhm, 2 x 8 mOhm RDSon)
MC10XS6200: Dual High-Side Switch (2 x 10 mOhm RDSon)
MC10XS6225: Dual High-Side Switch (1 x 25 mOhm, 1 x 10 mOhm RDSon)
MC10XS6325: Triple High-Side Switch (1 x 25 mOhm, 2 x 10 mOhm RDSon)
MC17XS6400: Quad High-Side Switch (4 x 17 mOhm RDSon)
MC17XS6500: Penta High-Side Switch (5 x 17 mOhm RDSon)
MC25XS6300: Triple High-Side Switch (3 x 25 mOhm RDSon)
MC40XS6500: Penta High-Side Switch (5 x 40 mOhm RDSon)

SPI Link - Linked SPI_Device component.

SOA Mode - Reading mode of the device.
There are 2 options:

Single read: Programmed SO address is used for a single read command. After the reading, the SO address returns to quick status register.
All reads: Programmed SO address is used for the next and all further read commands until a new programming.

Overtemperature Warning Threshold - Temperature sensor is located on each power transistor to protect the transistors and provide SPI status monitoring. When overtemperature warning threshold is exceeded, the outputs remain in current state, but overtemperature warning is reported.
There are 2 options:

115 C: Low overtemperature threshold.
135 C: High overtemperature threshold.

HID Selection - Smart overcurrent window control strategy is implemented to turn on an HID (high-intensity discharge) ballast, even in the case of a long power on reset time.
There are 4 options:

All channels: Smart HID feature is available for all channels.
Channel 3: Smart HID feature is available for channel 3 only.
Channels 3 and 4: Smart HID feature is available for channels 3 and 4 only.
Disabled: Smart HID feature is not available for any channel.

OCHI Type - To minimize the electro-thermal stress inside the device in case of a short-circuit, the OCHIx levels can be dynamically adjusted in regards to the control die temperature or evaluated during the OFF-to-ON output transition depending on the output voltage.
There are 4 options:

Default: Output is protected with default OCHI levels.
Thermal: Output is protected with the OCHI level depending on the control die temperature.
Transient: Output is protected with an OCHI levels depending on the output voltage.
Both: Output is protected depending on both temperature and output voltage.

Global PWM Duty Cycle - In addition to the individual PWM register, each channel can be assigned independently to a global PWM register. When a channel is assigned to global PWM, global PWM duty cycle is applied, but the switching phase, the prescaler and the pulse skipping are according to the corresponding output channel setting.

Channels - Number of channels (fixed to 6 to cover different models).

Output - Properties for output channel. **Not all features are available for all output channels.**

PWM Output Control - Properties influencing PWM output signal.

Global PWM - In addition to the individual PWM register, each channel can be assigned independently to a global PWM register. When a channel is assigned to global PWM, global PWM duty cycle is applied, but the switching phase, the prescaler and the pulse skipping are according to the corresponding output channel setting.
There are 2 options:

Enabled: Feature is enabled.

Disabled: Feature is disabled.

Channel Duty Cycle - Duty cycle of PWM when channel is controlled individually.

Phase Selection - Phase assignment of the output channel.
There are 4 options:

0
90
180
180

Pulse Skipping - Due to the output pulse shaping feature and the resulting switching delay time of the smart switches, duty cycles close to 0 resp. 100 percent can not be generated by the device. Therefore the pulse skipping feature is integrated to interpolate this output duty cycle range in Normal mode.
There are 2 options:

Enabled: Feature is enabled.

Disabled: Feature is disabled.

Slew Rate Prescaler - Depending on the programming of the prescaler setting register, the switching speeds of the outputs are adjusted to the output frequency range of each channel.
There are 3 options:

1: Fast slew rate in high frequency range.

2: Medium slew rate in medium frequency range.

4: Slow slew rate in low frequency range.

Output Initial State - State of the output after initialization.
There are 2 options:

On: Channel is turned on after initialization.

Off: Channel is turned off after initialization.

Direct Input Control - If enabled, output is assigned to direct input pin control. If you want to use direct input pin control, enable it in global configuration part of the component and set up the pins. In Normal mode the output state is controlled by direct input, but the channel output voltage depends on set duty (global or individual).
There are 2 options:

Enabled: Feature is enabled.

Disabled: Feature is disabled.

Open Load - Open load detection feature.

Open Load LED - For detection of small load currents (e.g. LED) in on state of the switch a special low current detection mode is implemented.
There are 2 options:

Enabled: Feature is enabled.

Disabled: Feature is disabled.

OLON Deglitch Time - When an open load has been detected, the output remains in on state. The deglitch time of the open load in on state can be controlled individually for each output to be compliant with different load types.
There are 2 options:

64 us: Short deglitch time (bulb mode).

2 ms: Long deglitch time (converter mode).

Overcurrent - Overcurrent protection against ultra-low resistive short-circuit conditions due to a smart overcurrent profile and severe short-circuit protection.

OCLO Threshold - The static overcurrent threshold can be programmed individually for each output in two levels to adapt low duty cycle dimming and a variety of loads.
There are 2 options:

High: The output is protected with the higher OCLO threshold.

Low: The lower OCLO threshold is applied.

Advanced Current Sensing Mode - An advanced current sense mode (ACM) is implemented in order to diagnose LED loads in Normal mode and to improve current sense accuracy for low current loads. In the ACM mode, the offset sign of current sense amplifier

is toggled on every CSNS SYNCB rising edge. The error amplifier offset contribution to the CSNS error can be fully eliminated from the measurement result by averaging each two sequential current sense measurements. ACM mode affects maximum current of the output.
There are 2 options:

Enabled: Feature is enabled.

Disabled: Feature is disabled.

Short OCHI - The length of the OCHI windows can be shortened by a factor of 2, to accelerate the availability of the CSNS diagnosis and to reduce the potential stress inside the switch during an overload condition.
There are 2 options:

Enabled: Feature is enabled.

Disabled: Feature is disabled.

No OCHI - The switch on process of an output can be done without an OCHI window, to accelerate the availability of the CSNS diagnosis.
There are 2 options:

Enabled: Feature is enabled.

Disabled: Feature is disabled.

Auto Initialization - When auto initialization is enabled, Init method will be called automatically within PE initialization function - PE_low_level_init().

3 Typical Usage

Examples of typical settings and usage of Gen4eXtremeSwitch component

Initialization of eSwitch.

Getting of eSwitch status information.

Setting channel to use individual PWM.

Setting channel to use global PWM.

Usage of incremental PWM feature.

Usage of open load detection feature.

Usage of over current protection feature.

Channel control with use of direct inputs.

Measurement of output current, battery voltage and die temperature.

Note: All used methods take as parameter either number of specific channel or array of values regarding used channels. It is important to realize that only enabled and configured channels in Processor Expert are relevant and unused channels are skipped. In terms of indexing this means that index passed to various methods or size of array corresponds to those used channels and appropriate index shift is applied.

Example: Channel 1 and 3 is enabled in component. Channel 1 - index 0, channel 3 - index 1 and so on.

Initialization of eSwitch.

This example shows how to handle device initialization when auto-initialization feature is disabled.

Required component setup and dependencies:

Properties: *Auto initialization*: Disabled

Methods: Init

Content of main.c:

Listing 1: Source code

```
void main(void)
{
  G4XS1_TError Error;
```

```
  uint16_t UserData = 1;

  /* Calling Init method more then once in user code will restore initial
   Processor Expert setting. */

  Error = G4XS1_Init(&UserData); /* It is possible to pass pointer to your
   own data, which is then stored in device data structure as TUserDataPtr.
   */
  if (Error != ERR_OK) {
    /* Initialization was successful. */
  } else {
    /* Initialization was not successful. */
  }

  uint16_t *MyData = (uint16_t *)(G4XS1_DeviceDataPtr->UserDataPtr); /* You
   can access your data later. Explicit conversion is needed because
   UserDataPtr is just typedef of (void *). */
}
```

Getting of eSwitch status information.

This example shows how to get and interpret eSwitch status information. This includes summary quick status, individual channel status and common device and I/O status.

Required component setup and dependencies:

Properties: *None*

Methods: GetQuickStatus GetChannelStatus GetDeviceStatus GetIOStatus

Content of main.c:

Listing 2: Source code

```
void main(void)
{
  G4XS1_TError Error;
  uint16_t StatusData;

  /* Reads quick status register of device 1. */
  Error = G4XS1_GetQuickStatus(0, &StatusData);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }
  else { /* Interpretation of status data. */
    if (G4XS_GET_QSFx(StatusData)) { /* Quick status flag for channel x (OC
   | OTW | OTS | OLON | OLOFF) */
      /* Take some action. */
    }
    if (G4XS_GET_CLKF(StatusData)) { /* PWM clock fail flag. */
      /* Take some action. */
    }
    if (G4XS_GET_RCF(StatusData)) {  /* Register clear flag. */
      /* Take some action. */
    }
    if (G4XS_GET_CPF(StatusData)) {  /* Charge pump flag. */
      /* Take some action. */
    }
  }

  /* The same procedure can be applied for Channel, Device and I/O status
   information. See header file for other useful macros. */
}
```

Setting channel's output state.

This example shows how to set channel output state. This can be used to change initial channel state without change to duty value.

Required component setup and dependencies:

Properties: *None*

Methods: SetOutputState

Content of main.c:

Listing 3: Source code

```
void main(void)
{
  G4XS1_TError Error;
  bool OutputStates[5] = {FALSE, TRUE, FALSE, TRUE, FALSE}; /* Explicit
    specification of output states for device with 5 channels. */

  Error = G4XS1_SetOutputState(0, OutputStates);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }
}
```

Setting channel to use individual PWM.

This example shows how to set channel to use individual PWM.

Required component setup and dependencies:

Properties: *None*

Methods: SetPWMControl SetPWMDuty SetPWMDutyValues

Content of main.c:

Listing 4: Source code

```
void main(void)
{
  G4XS1_TError Error;

  /* Sets control of selected channels to individual. */
  for (uint8_t i = 1; i < 6; i++) {
    Error = G4XS1_SetPWMControl(0, i, ccINDIVIDUAL);
    if (Error != ERR_OK) {
      /* Something went wrong. */
    }
  }

  uint16_t DutyValues[5] = {256, 256, 128, 128, 0};

  /* Note that methods also handle channel's output state. */

  ..... CHANGE DUTY FOR EACH CHANNEL SEPARATELY .....

  /* Sets PWM duty value for each channel separately. */
  for (uint8_t ch = 1; ch < 6; ch++) {
    Error = G4XS1_SetPWMDuty(0, ch, DutyValues[ch - 1]);
    if (Error != ERR_OK) {
      /* Something went wrong. */
    }
  }

  ..... CHANGE DUTY FOR ALL CHANNELS AT ONCE .....
```

```
  /* Sets PWM duty values for all channels at once. */
  Error = G4XS1_SetPWMDutyValues(0, DutyValues);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }
}
```

Setting channel to use global PWM.

This example shows how to set channel to use global PWM.

Required component setup and dependencies:

Properties: *None*

Methods: SetPWMControl SetGPWMDuty

Content of main.c:

Listing 5: Source code

```
void main(void)
{
  G4XS1_TError Error;

  /* Sets control of selected channels to global */
  for (uint8_t i = 1; i < 6; i++) {
    Error = G4XS1_SetPWMControl(0, i, ccGLOBAL);
    if (Error != ERR_OK) {
      /* Something went wrong. */
    }
  }

  uint16_t Duty = 128;

  /* Note that method also handles channel's output state. */

  /* Sets GPWM duty, this applies for all channels with global PWM control.
   */
  Error = G4XS1_SetGPWMDuty(0, Duty);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }
}
```

Usage of incremental PWM feature.

This example shows how to use incremental PWM feature. It lowers overhead of SPI communication.

Required component setup and dependencies:

Properties: *None*

Methods: IncrementalPWMControl

Content of main.c:

Listing 6: Source code

```
void main(void)
{
  G4XS1_TError Error;

  /* Sets control of selected channels to individual, this feature does not
   work for global PWM control. */
  for (uint8_t i = 1; i < 6; i++) {
```

```
    Error = G4XS1_SetPWMControl(0, i, ccINDIVIDUAL);
    if (Error != ERR_OK) {
      /* Something went wrong. */
    }
  }

  /* Predefined steps of increment/decrement for each enabled channel. */
  G4XS1_TLSBStep LSBStep[5] = {idsNONE, ids4LSB, ids8LSB, ids16LSB, idsNONE
    };

  while (1) { /* increments to maximum duty */
    Error = G4XS1_IncrementalPWMControl(0, idsPOSITIVE, LSBStep);
    if ((Error != ERR_OK) && (Error != ERR_MAX_DUTY)) {
      /* something went wrong */
    }
    for (uint8_t j = 0; j < 10; j++) { /* waits for 200 ms */
      WaitMS(20);
      G4XS1_FeedWatchdog(); /* It is necessary to feed watchdog during delay
    . */
    }
    /* Note that ERR_MAX_DUTY is returned when any of channels reach this
    threshold. */
    if (Error == ERR_MAX_DUTY) {
      break;
    }
  }

  while (1) { /* decrements to minimum duty */
    Error = G4XS1_IncrementalPWMControl(0, idsNEGATIVE, LSBStep);
    if ((Error != ERR_OK) && (Error != ERR_MIN_DUTY)) {
      /* something went wrong */
    }
    for (uint8_t j = 0; j < 10; j++) { /* waits for 200 ms */
      WaitMS(20);
      G4XS1_FeedWatchdog();
    }
    /* Note that ERR_MIN_DUTY is returned when any of channels reach this
    threshold. */
    if (Error == ERR_MIN_DUTY) {
      break;
    }
  }
}
```

Usage of open load detection feature.

This example shows how to use open load detection feature.

Required component setup and dependencies:

Properties: *None*

Methods: SetOLOFF OLLEDTrig

Content of main.c:

Listing 7: Source code

```
void main(void)
{
  G4XS1_TError Error;
  uint16_t StatusData;

  /* Enables open load detection in OFF state. */
```

```
  for (uint8_t i = 1; i < 6; i++) { /* This feature can be used also
   individually. */
    Error = G4XS1_SetOLOFF(0, i);
    if (Error != ERR_OK) {
      /* Something went wrong. */
    }
  }

  /*
    The detection result is reported in:
          the corresponding QSFx bit in the quick status register #1
          the global open load flag OLF (register #1:#7)
          the OLOFF bit of the corresponding channel status register #2:#6
  */

  for (uint8_t i = 1; i < 6; i++) {
    Error = G4XS1_GetChannelStatus(0, i, &StatusData);
    if (G4XS_GET_OLOFFX(StatusData)) {
      /* Take some action. */
    }
  }

  /*
    Triggers open load detection in ON state of the switch when following
   condition is met.
    When a special low current detection mode is used (OLLED) and output is
   in fully ON operation (100% PWM):
          the detection on all outputs is triggered by setting the OLLED
   TRIG bit inside the LED control register #13-2
          at the end of detection time, the current source (IOLLED) is
   disabled 100   sec (typ.) after the output reactivation.
  */

  Error = G4XS1_OLLEDTrig(0);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }

  /*
    The detection result is reported in:
          the corresponding QSFx bit in the quick status register #1
          the global open load flag OLF (register #1:#7)
          the OLON bit of the corresponding channel status register #2:#6
  */

  for (uint8_t i = 1; i < 6; i++) {
    Error = G4XS1_GetChannelStatus(0, i, &StatusData);
    if (G4XS_GET_OLONX(StatusData)) {
      /* Take some action. */
    }
  }
}
```

Usage of over-current protection feature.

This example shows how to use over-current protection feature.

Required component setup and dependencies:

Properties: *None*

Methods: SetOCHIOnDemand

Content of main.c:

Listing 8: Source code

```
void main(void)
{
  G4XS1_TError Error;

  /* In some instances, a lamp might be de-powered when its supply is
   interrupted by the opening of a switch (as in a door),
     or by disconnecting the load (as in a trailer harness).
     In these cases, the driver should be tolerant of the inrush current
   occurring when the load is reconnected. */

  for (uint8_t i = 1; i < 6; i++) { /* This feature can be used also
   individually. */
    Error = G4XS1_SetOCHIOnDemand(0, i);
    if (Error != ERR_OK) {
      /* Something went wrong. */
    }
  }
}
```

Channel control with use of direct inputs.

This example shows how to control channels with direct inputs. This method can be used to preserve functionality in fail mode. It can be used also in normal mode when direct input control is enabled, but channel operates on predefined PWM duty value.

Required component setup and dependencies:

Properties: *None*

Methods: SetDirectInput SetDirectInputValues

Content of main.c:

Listing 9: Source code

```
void main(void)
{
  G4XS1_TError Error;
  bool InputValues[5] = {TRUE, FALSE, TRUE, FALSE, TRUE};

  /* Sets direct input value for each channel separately. */
  for (uint8_t ch = 1; ch < 6; ch++) {
    Error = G4XS1_SetDirectInput(0, ch, InputValues[ch - 1]);
    if (Error != ERR_OK) {
      /* Something went wrong. */
    }
  }

  ..... OR .....

  /* Sets direct input values for all channels at once. */
  Error = G4XS1_SetDirectInputValues(0, InputValues);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }
}
```

Measurement of output current, battery voltage and die temperature.

This example shows how to measure feedback from eSwitch.

Required component setup and dependencies:

Properties: *Current, Voltage and Temperature Sensing*: Enabled

Methods: ConfigureMonitoring GetSenseValue

Events: OnMeasurementSynchronization

Content of Events.c:

Listing 10: Source code

```
void OnMeasurementSynchronization(void)
{
  trigger_flag = TRUE;
}
```

Content of main.c:

Listing 11: Source code

```
void main(void)
{
  G4XS1_TError Error;
  uint16_t Result;

  for (uint8_t mux = smOUT1_CURRENT; mux <= smCTRL_DIE_TEMP; mux++) {
    Error = G4XS1_ConfigureMonitoring(0, mux, trigger);
    if (Error != ERR_OK) {
      /* Something went wrong. */
    }

    /* After configuring of monitored output sense value has to be measured
   in valid interval.
       This can be achieved by usage of measurement synchronization trigger
   or by explicitly specified delay. */

    ..... SYNCHRONIZATION TRIGGER .....

    trigger_flag = FALSE;
    while(!trigger_flag);

    ..... USER SPECIFIED DELAY .....

    /* It is up to user to find correct delay value to hit valid measurement
     point. */
    WaitMS(?);

    ......

    Error = G4XS1_GetSenseValue(&Result);
    if (Error != ERR_OK) {
      /* something went wrong */
    }
  }
}
```

4 User Types

TDeviceDataPtr = *Device data pointer*

TUserDataPtr = *User data pointer*

***ComponentName*_TDeviceIdx** = enum { didxDEV1, didxDEV2, didxDEV3, didxDEV4} *Device index*

***ComponentName*_TSIRegister** = enum { sirINIT1, sirINIT2, sirCH1_CTRL, sirCH2_CTRL, sirCH3_CTRL, sirCH4_CTRL, sirCH5_CTRL, sirCH6_CTRL, sirOUT_CTRL, sirGPWM_CTRL1, sirGPWM_CTRL2, sirOC_CTRL1, sirOC_CTRL2, sirIE, sirPRS1, sirPRS2, sirOL_CTRL, sirOLLED_CTRL, sirINC_DEC} *Device serial input registers*

***ComponentName*_TSORegister** = enum { sorQUICK_STATUS, sorCH1_STATUS, sorCH2_STATUS, sorCH3_STATUS, sorCH4_STATUS, sorCH5_STATUS, sorDEVICE_STATUS, sorIO_STATUS, sorDEVICE_ID} *Device serial output registers*

***ComponentName*_TChannelControl** = enum { ccINDIVIDUAL, ccGLOBAL} *Channel control types*

***ComponentName*_TIDSign** = enum { idsPOSITIVE, idsNEGATIVE} *Incremental PWM control sign*

***ComponentName*_TLSBStep** = enum { idsNONE, ids4LSB, ids8LSB, ids16LSB} *Steps for incremental PWM control*

***ComponentName*_TExtClkState** = enum { ecsENABLED, ecsDISABLED} *External clock states*

***ComponentName*_TSenseMux** = enum { smOFF, smOUT1_CURRENT, smOUT2_CURRENT, smOUT3_CURRENT, smOUT4_CURRENT, smOUT5_CURRENT, smVBAT_MONITOR, smCTRL_DIE_TEMP} *Possible outputs for monitoring*

***ComponentName*_TSenseSyncTrigger** = enum { sstOFF, sstVALID, sstTRIG0, sstTRIG12} *Current sense synchronization values*

***ComponentName*_TError** = *Error codes*

How to Reach Us:

Home Page:
NXP.com

Web Support:
http://www.nxp.com/support

Document Number: PEXMC12XSF-MC12XS6PUG
Rev. 1.0
5/2016