G4XS_K64F_XSG_ParseStatusData

Processor Expert Example Project

Date: *21. 11. 2016*

Revision: 2.0

Overview

The purpose of this example project is to show how to parse and interpret status data with use of the Gen4eXtremeSwitch component. The result is printed to the console. Such an implementation can be used to real time status reporting of device condition.

Requirements

Kinetis Design Studio 3.2.0 and newer

FRDM-K64F MCU board

FRDM-32XSG-EVB Gen4 eXtremeSwitch board

USB cable

Setting up hardware

Target platform for this example is FRDM-K64F MCU board with FRDM-32XSG-EVB eXtremeSwitch board.

Chip select on FRDM-32XSG-EVB board has to be properly configured in such way that CSB0 pin is used. This can be achieved by switching ***number 0 on SW2 switch*** on the eSwitch board to ***ON position***. The rest of CSB routes should remain in OFF position.

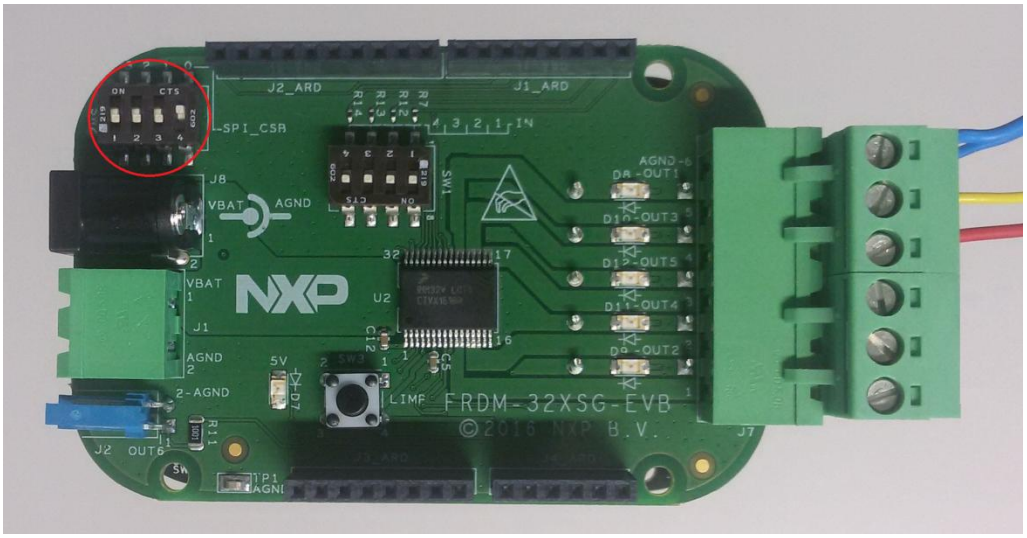

Figure 1 CSB Route Selection

In order to get the project successfully running you need to connect some load to channels 1 and 3 on your FRDM eSwitch board.

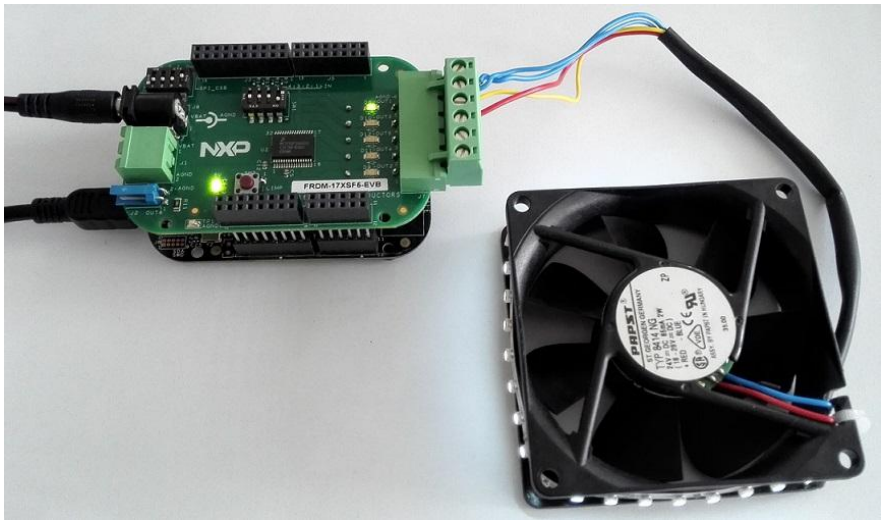

Figure 2 Channel Load

Set up OpenSDA connection between PC and MCU board with USB cable.

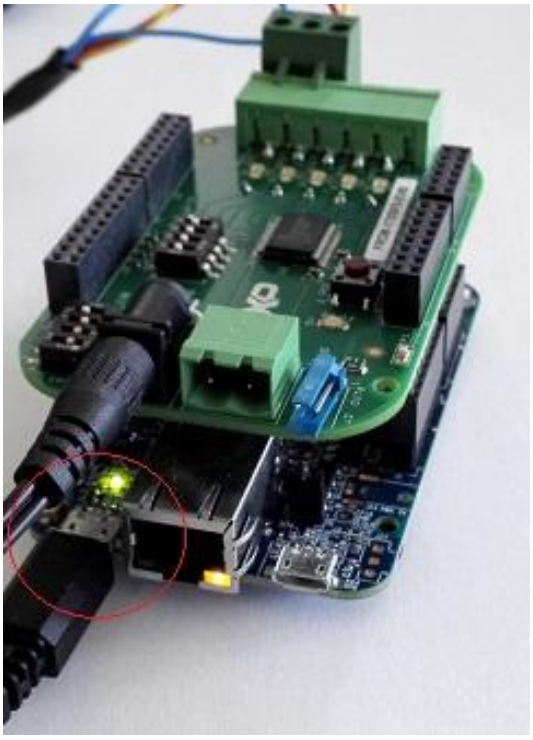

Figure 3 OpenSDA Connection

Setting up software

Make sure you have installed KDS 3.2.0 or newer which comes with Processor Expert support.

These components must be imported to the Processor Expert Component Library.

- **SPI_Device** component (encapsulates SPI communication)
- **Gen4eXtremeSwitch** component (encapsulates eSwitch configuration)

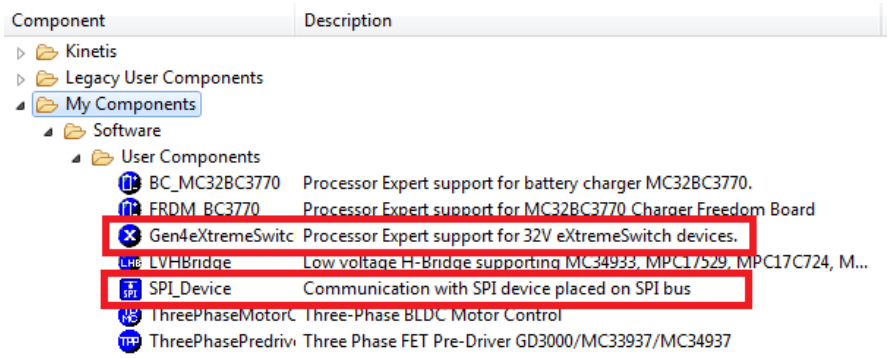

Figure 4 Processor Expert Component Library

Check that your OpenSDA connection has been set up.

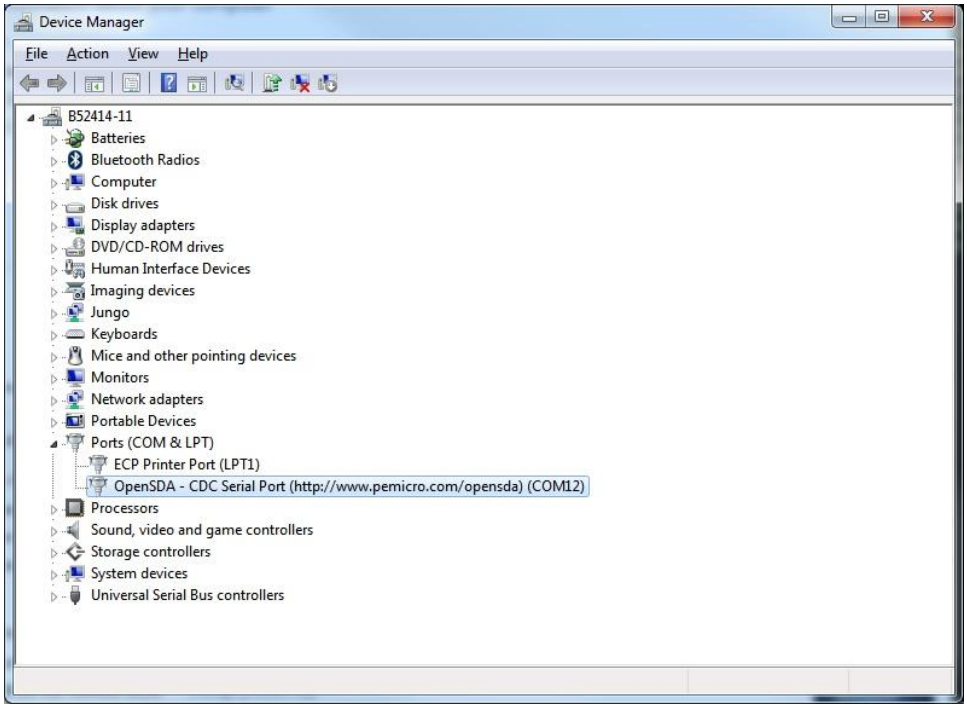

Figure 5 OpenSDA Virtual Port

Description

The example is preconfigured in the following way. FRDM-32XSG-EVB board is configured in **Gen4eXtremeSwitch** component, see Figure 7. SPI settings are configured separately in **SPIMaster_LDD** component, see Figure 8 and in inherited **CSpin1** component, see Figure 9. All components can be accessed in component tree, see Figure 6.

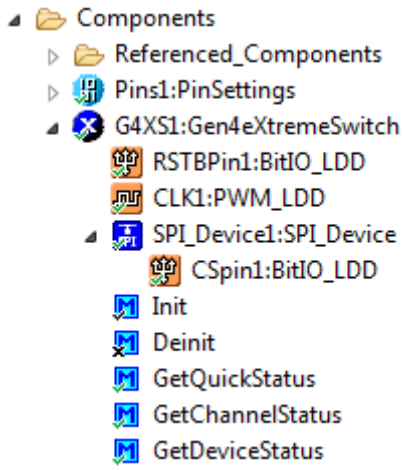

Figure 6 Component Tree

Channels 1 and 3 are enabled and configured in Gen4eXtremeSwitch component properties, see Figure 7. Both channels are enabled by default with PWM duty set to maximum value (256). Watchdog timeout period is set to 128ms. Make sure that if you modify project, you still hold this watchdog period. All other features are set to default values except pin selection which follows **Error! Reference source not found.**.

Component Name	G4XS1	
SPI Configuration	Parallel SPI	
Global Configuration		
RSTB Pin	PTE26/ENET_1588_CLKIN/UART4_...	
External Clock Frequency	50000 D	
CLK Pin	ADC0_SE15/PTC1/LLWU_P6/SPI0_...	ADC0_SE15/PTC1/LLWU_P6/SPI0_...
Watchdog Timeout	128 ms	
Direct Input Control	Disabled	
Current, Voltage and Tempe	Disabled	
Devices	1	
Device1		
Device Model	MC17XSF500	
SOA Mode	Single read	
Overtemperature Warning	115 °C	
HID Selection	Disabled	
OCHI Type	Default	
Global PWM Duty Cycle	0 D	
Channels	6	
Output1	Enabled	Corresponds to channel with index...
PWM Output Cont		
Global PWM	Disabled	
Channel Duty C	256 D	
Phase Selectior	0°	
Pulse Skipping	Disabled	
Slew Rate Presc	1	
Output Initial S	On	
Direct Input Contr	Disabled	
Open Load		
Open Load LED	Disabled	
OLON Deglitch	64 us	
Overcurrent		
OCLO Threshol	Low	
Advanced Curr	Disabled	
Short OCHI	Disabled	
No OCHI	Disabled	
Output2	Disabled	Channel disabled.
Output3	Enabled	Corresponds to channel with index...
Output4	Disabled	Channel disabled.
Output5	Disabled	Channel disabled.
Output6	Disabled	Channel disabled.
Auto Initialization	yes	

Figure 7 G4XS Component Settings

SPI setting involves pin selection (MISO, MOSI, CLK and CSB). The first three pins are configured in **SPIMaster_LDD**, see Figure 8 and the last one is configured in **CSPin** component, see Figure 9, which is inherited by **SPI_Device** component.

Device	SPI0
Interrupt service/event	Enabled
Settings	
Input pin	Enabled
Pin	PTD3/SPI0_SIN/UART2_TX/FTM3_CH3/FB_AD3/I2C0_SDA
Output pin	Enabled
Pin	PTD2/LLWU_P13/SPI0_SOUT/UART2_RX/FTM3_CH2/FB_AD4/I2C0_...
Clock pin	
Pin	ADC0_SE5b/PTD1/SPI0_SCK/UART2_CTS_b/FTM3_CH1/FB_CS0_b
Chip select list	0
Attribute set list	1
Attribute set 0	
Width	8 bits
MSB first	yes
Clock polarity	Low
Clock phase	Change on leading edge
Parity	None
Chip select toggling	no
Clock rate index	0
Delay after transfer index	0
CS to CLK delay index	0
CLK to CS delay index	0
Clock rate	0.667572 µs
Delay after transfer	0.190735 µs
CS to CLK delay	0.190735 µs
CLK to CS delay	0.190735 µs
Initialization	
Auto initialization	yes

Figure 8 SPI Master Component Settings

Pin for I/O	PTD0/LLWU_P12/SPI0_PCS0/UART2_RTS_b/FTM3_CH0/FB_ALE/FB_...
Direction	Output
Initialization	
Init. direction	Output
Init. value	0
Auto initialization	yes

Figure 9 Chip Select Pin Settings

In order to be able to print status information, **ConsoleIO** component is used. This component encapsulates UART communication protocol. UART settings can be accessed through inherited **Serial_LDD** component, see Figure 10.

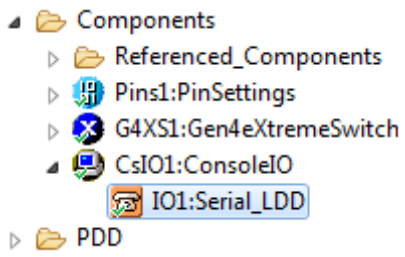

Figure 10 ConsoleIO Component Tree

Receiver and transmitter port selection and baud rate value is on Figure 11.

Device	UART0
Interrupt service/event	Disabled
Settings	
Data width	8 bits
Parity	None
Baud rate	115200 baud
Receiver	Enabled
RxD	PTB16/SPI1_SOUT/UART0_RX/FTM_CLKIN0/FB_AD17/EWM_IN
Transmitter	Enabled
TxD	PTB17/SPI1_SIN/UART0_TX/FTM_CLKIN1/FB_AD16/EWM_OUT_b
Initialization	
Auto initialization	yes

Figure 11 ConsoleIO Component Settings

Implementation itself is quite straightforward. All of the implemented functions in **main.c** (Figure 12) are responsible for parsing selected status register. The result is printed to the console. Functions implement the following logic.

1. ***Fetch*** content of the selected status register.
2. ***Parse*** obtained value with use of predefined macros.
3. ***Take action*** (print to console or something else) if some flag is or is not present.

```c
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  printf("32eXtremeSwitch\n");

  if (G4XS1_Init(NULL) == ERR_OK) {
    printf("Initialization was successful.\n");
  }
  else {
    printf("Initialization failed.\n");
  }

  ParseDeviceID();
  ParseIOStatusData();

  while (1) {
    printf("\n<G4XS Status Report>\n\n");

    PrintStatusRegisters();

    ParseQuickStatusData();
    ParseChannelStatusData();
    ParseDeviceStatusData();

    for (uint8_t i = 0; i < 40; i++) { /* waits for 2 sec */
      WaitMS(10);
      G4XS1_FeedWatchdog();
    }
  }
```

Figure 12 Content of main.c

Functions **ParseDeviceID()** and **ParseIOStatusData()** are called just once. The first one gives information about device type, family and development design status. The second gives information whether channel is on or off.

The rest of functions are called in infinite loop to provide real-time status report (for all enabled channels). Function **ParseQuickStatusData()** gives overall information for all channels, **ParseChannelStatusData()** expands this information with specific status for each channel and **ParseDeviceStatusData()** informs about device state and condition.

Import the example project

Pre-requisites such as needed components are assumed to be imported already. If that is correct, you can approach to importing of the example project.

1. In KDS click on the ***File / Import***.
2. Choose ***General / Existing Projects into Workspace***.
3. Click ***Browse to select root directory*** with your downloaded example projects.
4. ***Select project*** named **G4XS_K64F_XSG_ParseStatusData** and click ***Finish*** to complete process.
5. Now the example project should be copied to your workspace and ready to run.

Build and Debug

In order to build and run the project you need to ***generate code*** at first. Then you can ***build*** the project usual way. If the build is successful, ***debug and run*** the project.