# UM11182

## Q100 (MC33XS2410) Extreme switch software driver user guide

**Rev. 1.1 — 5 September 2022** **User manual**

**Document information**

| Information | Content |
|---|---|
| Keywords | Q100 eSwitch extreme switch |
| Abstract | This documentation describes how to install and use the Q100 Extreme Switch software driver. |

**Revision history**

| Rev | Date | Description |
|---|---|---|
| v1.1 | 20220905 | Section 2.2: Corrected the following entry:<br>• 12 bits ADC:<br>– Current from 5.0 mA to 5.0 A with ± 6 % above 1 A<br>– Voltage from 0.5 to 65 V with ± 6.5 % above 5.0 V<br>– Temperature warning for each channel plus central die monitoring<br>• Revision history relocated |
| v.1.0 | 20190830 | Initial version |

# 1 Overview

This documentation describes how to install and use the Q100 Extreme Switch software driver (driver).

The Q100 Extreme Switch software driver encapsulates the functionality of the Q100 (MC33XS2410) device. The driver acts as an API layer between the microcontroller low-level drivers, e.g. SDK, and the user application, allowing you to perform the following:

- Setting and reading device registers (control and diagnostic bank)
- Initializing device with default or custom register values
- Controlling the OUTx channel by Direct Input or SPI signal, PWM or without PWM
- Reading fault, warning and channel status
- Measurements of voltage, current and temperature
- Enabling watchdog, transit to safe mode
- Setting PWM PI regulation

# 2 MCU compatibility

The driver implementation is generic; there is no dependency on a specific MCU. Virtually any MCU with the required peripherals should be able to use the driver.

## 2.1 Peripheral requirements

The driver needs the following MCU peripherals for its function:

- **SPI Module** is required for communication (MOSI_M, MISO_M, SCLK_M, CSB_M).
- **GPIO** is required for controlling RESET_B pin or optionally used for software controlled SPI chip select (CSB_M) instead of HW chip select.
- **Interrupt** pin is optionally required for use with the FAULT_B pin – interrupt implementation is up to user.
- **Timer** is optionally required for generating external clock signal for Pulse-Width Modulation (PWM) of the Q100 device - implementation is up to user

Depending on the user application, other resources may be required. See the provided example projects.

## 2.2 Supported devices

**Q100 (MC33XS2410)**

- Four fully-protected 100 mΩ / dual 50 mΩ (at 25 °C) high-side switches
- 4 x 1.8 A DC (Pd 2.5 W @ TJ 150 °C) or 2 x 3.6 A DC in parallel mode configuration
- Floating power output architecture to drive all types of loads
- 16-bit SPI port communication 3.3 V / 5.0 V compatible with daisy chain capability
- Outputs controllable via SPI-bus or direct inputs
- Diagnostic status reported via SPI-bus
- Watchdog for invalid commands or inactive SPI, with programmable timeout
- Programmable interrupt generator that reports to FAULT pin or SPI-bus
- Four independent PWM modules programmable from 0.5 Hz to 2.0 kHz
- Protection for battery transient overvoltage and reversed polarity battery connection
- Configurable safe mode

- Standby mode with very low power consumption
- 10 mA open load detection in ON state
- Latch off with configurable auto retry
- Severe short-circuit and overload protection
- Programmable active current limit threshold to minimize short-circuit effect
- 12 bits ADC:
  - Current from 5.0 mA to 5.0 A with ± 6 % above 1 A
  - Voltage from 0.5 to 65 V with ± 6.5 % above 5.0 V
  - Temperature warning for each channel plus central die monitoring
- Qualified in accordance with AEC Q100 grade 1
- Electrical transient disturbance immunity according of ISO 7637-2 and ISO 16750-2

## 2.3 Supported MCUs

The current implementation of the Q100 software driver is generic, such that any suitable 32 b microcontroller with SPI module and other necessary peripheries can be used. See Section 2.1 for peripheral requirements.

| Board name | MCU Board | Description |
|---|---|---|
| FRDM-XS2410EVB | FRDM-KL25Z | Q100 board with FRDM-KL25 |
| FRDM-XS2410EVB | S32K144EVB-Q100 | Q100 board with S32K144EVB |

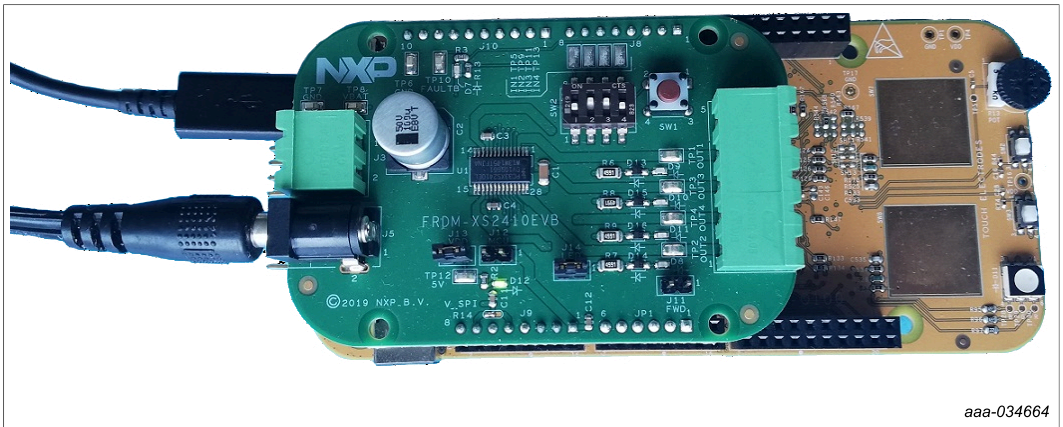
The driver was tested with an S32K144 MCU and S32K14x EAR SDK 0.8.6. Figure 1 shows a HW setup of S32K144EVB-Q100 and Q100 EVB.

The FRDM-XS2410EVB (Q100) evaluation board is directly compatible with the FRDM KL25Z board. See Table 1 for used pin compatibility between FRDM-XS2410EVB (Q100) evaluation board and S32K144EVB-Q100 and FRDM KL25Z.

**Table 1. Q100 EVB pin compatibility with S32K144EVB-Q100**

| Pin Function (Q100 EVB) | MOSI | MISO | CLK | CS | RST_B | FAULT_B | LHM | IN1 | IN2 | IN3 | IN4 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| S32K144EVB-Q100 | PTB4 | PTB3 | PTB2 | PTB5 | PTD13 | PTB8 | PTB9 | PTC11 | PTC10 | PTB11 | PTB10 |
| FRDM KL25Z | PTD2 | PTD3 | PTD1 | PTD0 | PTA13 | PTD4 | PTA12 | PTC9 | PTC8 | PTA5 | PTA4 |

When other MCU boards or other Q100 EVB are used, refer to the provided user guides and schematics of the respective boards.




aaa-034664

**Figure 1. Set of S32K144 and FRDM-XS2410EVB (Q100) evaluation boards**

# 3 Q100 Extreme switch software driver

This section provides an overview of the functionality, settings and usage of the driver (configuration and functions). For additional information, see the *API Programmer's Guide* (included in the Q100 software driver zip file) and the comments embedded in the code.

The MSDI software driver consists of three files. The main functionality is contained in **Q100.c** and **Q100.h** whereas **Q100_regmap.h** contains register map of device. As the driver is not platform specific, several functions (marked as *external*in Q100.h) need to be implemented by the user.

## 3.1 Configuring the driver

The configuration structure shown in Figure 2 is the user interface available for configuring the driver and its behavior. The user must set appropriate Q100 device type and instance of the driver (in case more instances will be used) into the driver configuration structure (*Q100_driver_t*). The driver configuration structure is passed to all API functions of the driver as a parameter. Members *qSpiStatus*, *regMap* and *toggleBit* are initialized automatically in the *Q100_Init* function.

As the driver is not platform specific, the user needs to implement several low-level functions. See Section 3.2. *Instance* member in *Q100_driver_t* structure is not modified by the driver itself. This member is passed to user-defined functions in order to differentiate between Q100 driver instances in case that more Q100 devices are connected to the MCU.

```
/*!
 * @brief Q100 driver structure.
 */
typedef struct {
    uint8_t instance;                       /*!< Instance of driver. It is used to recognize drivers e.g.
                                                 for different SPI communication for each driver instance. */
    Q100_device_ctrl_data_t devCtrlData;    /*!< Control settings which correspond to Control register bank. */
    Q100_device_diag_data_t devDiagData;    /*!< Diagnostic data which correspond to Diagnostic register bank. */
    Q100_quick_spi_status_t qSpiStatus;     /*!< Quick status which is provided by each read SPI command. */

    Q100_reg_map_t regMap;                  /*!< Register map where are all raw values stored. */
    bool toggleBit;                         /*!< Actual value of toggle bit. Is no allowed to modify it. */
} Q100_driver_t;
```

aaa-032698

**Figure 2. Driver configuration**

For a more detailed description of the user configuration structure, refer to the API programmer's guide.

## 3.2 Driver API

This Q100 software driver provides API that can be used for dynamic real-time configuration of a device in user code. For a summary of available functions, see Table 2. As the Q100 SW driver is platform independent, functions for SPI transfer and GPIO control need to be implemented by the user. There are helper functions that can be used for converting user-friendly float values into raw values. These values can be filled in the driver structure.

**Table 2. Q100 software driver API**

| Function | Description |
|---|---|
| **External function** | |
| Q100_SPI_Transfer | External defined function for SPI transfer. |
| **SPI communications functions** | |
| Q100_SpiWriteCtrl | Used for writing control registers via SPI |
| Q100_SpiReadCtrl | Used for reading control registers via SPI |
| Q100_SpiReadDiag | Used for reading diagnostic registers via SPI |
| **APP control helpers configuration functions** | |
| Q100_UserCtrlPwmToRaw | Used for conversion user PWM (float) value to raw (uint8_t) value |
| Q100_RawCtrlPwmToUserVal | Used for conversion PWM raw (uint8_t) value to user (float) |
| Q100_UserCtrlCurrToRaw | Used for conversion user Current (float) value to raw (uint8_t) value |
| Q100_RawCtrlCurrToUserVal | Used for conversion Current raw (uint8_t) value to user (float) value |
| Q100_UserCtrlVoltToRaw | Used for conversion user Voltage (float) value to raw (uint8_t) value |
| Q100_RawCtrlVoltToUserVal | Used for conversion Voltage raw (uint8_t) value to user (float ) value |
| Q100_UserCtrlTempToRaw | Used for conversion user Temperature (float) to raw (uint8_t) value |
| Q100_RawCtrlTempToUserVal | Used for conversion Temperature raw (uint8_t) to user (float) value |
| **APP control configuration functions** | |
| Q100_WriteGlobalControlSettings | Writes global control settings (reg: Q100_GLB_CTRL #00h → Q100_READBACK #01h) |
| Q100_ReadGlobalControlSettings | Reads global control settings (reg: Q100_GLB_CTRL #00h → Q100_READBACK #01h) |
| Q100_WriteInputControlSettings | Writes input control settings (reg: Q100_OUT1_4_CTRL #02h → Q100_IN_CTRL2 #04h) |
| Q100_ReadInputControlSettings | Reads input control settings (reg: Q100_OUT1_4_CTRL #02h → Q100_IN_CTRL2 #04h) |
| Q100_WritePwmSettings | Writes PWM control settings (reg: Q100_PWM_CTRL1 #05h → Q100_PWM_DC4 #0Fh) |
| Q100_ReadPwmSettings | Reads PWM control settings (reg: Q100_PWM_CTRL1 #05h → Q100_PWM_DC4 #0Fh) |
| Q100_WriteIrqWarningSettings | Writes interrupt and warnings configuration for SPI and FAULT_B pin (reg: Q100_EN_IRQ_SPI #10h → Q100_EN_WARN_PIN #13h) |
| Q100_ReadIrqWarningSettings | Reads interrupt and warnings configuration for SPI and FAULT_B pin (reg: Q100_EN_IRQ_SPI #10h → Q100_EN_WARN_PIN #13h) |
| Q100_WriteWatchdogSettings | Writes watchdog configuration (reg: Q100_WDT_REG #14h) |
| Q100_ReadWatchdogSettings | Reads watchdog configuration (reg: Q100_WDT_REG #14h) |
| Q100_WriteMeasurementsSettings | Writes measurements configuration (reg: Q100_M_SETUP #15h → Q100_C_CTRL #16h) |
| Q100_ReadMeasurementsSettings | Reads measurements configuration (reg: Q100_M_SETUP #15h → Q100_C_CTRL #16h) |
| Q100_WriteUnderOverCurrentSettings | Writes undercurrent and overcurrent configurations (reg: Q100_WC_CTRL #17h → Q100_UCW_OUT4 #1Fh) |
| Q100_ReadUnderOverCurrentSettings | Reads undercurrent and overcurrent configurations (reg: Q100_WC_CTRL #17h → Q100_UCW_OUT4 #1Fh) |
| Q100_WriteUnderOverVoltageSettings | Writes undervoltage and overvoltage configurations (reg: Q100_WV_CTRL #20h → Q100_UVW_OUT4 #28h) |
| Q100_ReadUnderOverVoltageSettings | Reads undervoltage and overvoltage configurations (reg: Q100_WV_CTRL #20h → Q100_UVW_OUT4 #28h) |
| Q100_WriteTemperatureSettings | Writes common temperature warning threshold (reg: Q100_TEMP_WT #29h) |

**Table 2. Q100 software driver API***...continued*

| Function | Description |
|---|---|
| Q100_ReadTemperatureSettings | Reads common temperature warning threshold (reg: Q100_TEMP_WT #29h) |
| Q100_WriteVBATinOFFstateSettings | Writes short to VBAT in OFF state settings (reg: Q100_BV_STVB #2Ah → Q100_BT_STVB #2Bh) |
| Q100_ReadVBATinOFFstateSettings | Reads short to VBAT in OFF state settings (reg: Q100_BV_STVB #2Ah → Q100_BT_STVB #2Bh) |
| Q100_WriteOpenLoadSettings | Writes open load settings (reg: Q100_OPD_CTRL1 #2Ch → Q100_I_OLDx #31h) |
| Q100_ReadOpenLoadSettings | Reads open load settings (reg: Q100_OPD_CTRL1 #2Ch → Q100_I_OLDx #31h) |
| Q100_WriteActiveCurrentLimitSettings | Writes active current limit settings (reg: Q100_ACL_CTRL1 #32h → Q100_ACL_CTRL2 #33h) |
| Q100_ReadActiveCurrentLimitSettings | Reads active current limit settings (reg: Q100_ACL_CTRL1 #32h → Q100_ACL_CTRL2 #33h) |
| Q100_WriteSevereShortCircuitSettings | Writes severe short circuit settings (reg: Q100_SSC_CTRL #34h) |
| Q100_ReadSevereShortCircuitSettings | Reads severe short circuit settings (reg: Q100_SSC_CTRL #34h) |
| Q100_WriteOverloadProtectionSettings | Writes over load protection settings (reg: Q100_OLP_CTRL #35h → Q100_OCL_OUT4 #39h) |
| Q100_ReadOverloadProtectionSettings | Reads over load protection settings (reg: Q100_OLP_CTRL #35h → Q100_OCL_OUT4 #39h) |
| Q100_WritePwmRegulationSettings | Writes Proportional-integral regulation compensation settings (reg: Q100_PI_CTRL1 #3Ah → Q100_I_SET4 #3Fh) |
| Q100_ReadPwmRegulationSettings | Reads Proportional-integral regulation compensation settings (reg: Q100_PI_CTRL1 #3Ah → Q100_I_SET4 #3Fh) |
| Q100_FillDataStructureBy | Gets settings for driver structure depend on parameter filledBy |
| Q100_Init | Initializes device by values stored in driver data structure |
| **Runtime control functions** | |
| Q100_EnableNonPwmOutput | Controls specified non-PWM output |
| Q100_EnablePwmOutput | Controls specified PWM output |
| Q100_SynchronizePwmOutputs | Synchronize all PWM outputs |
| **APP diagnostic status functions** | |
| Q100_ReadGlobalStatus | Reads global status from diagnostic register (Q100_GLB_STA #00h) |
| Q100_ReadInputOutputState | Reads input output state from diagnostic register (Q100_IN_OUT_STA #01h) |
| Q100_ReadChannelStatus | Reads channel status from diagnostic register (Q100_OUT1_STAx #02h → #05h) |
| Q100_GetSpecificChannelStatus | Helps to find specific event in channel status register |
| Q100_ReadIrqStatus | Reads IRQ interrupt status from diagnostic register (Q100_ISR_IRQ #06h) |
| Q100_GetSpecificIrqStatus | Helps to find specific event in IRQ status register |
| Q100_ReadWarningStatus | Reads Warning interrupt status from diagnostic register (Q100_ISR_WARN #07h) |
| Q100_GetSpecificWarnStatus | Helps to find specific event in WARN status register |
| **APP diagnostic helpers configuration functions** | |
| Q100_RawDiagCurrToUserVal | Helper conversion function from diagnostic raw Current (uint16_t) to user value (float) |
| Q100_RawDiagVoltToUserVal | Helper conversion function from diagnostic raw Voltage uint16_t) to user value (float) |
| Q100_RawDiagTempToUserVal | Helper conversion function from diagnostic raw Temperature (uint16_t) to user value (float) |
| Q100_RawDiagPiDutyToUserVal | Helper conversion function from diagnostic raw PI duty cycle (uint16_t) to user value (float) |
| Q100_RawDiagFBCurrToUserVal | Helper conversion function from diagnostic raw Feedback Current in Ton/2 (uint16_t) to user value (float) |
| **APP diagnostic measurement functions** | |
| Q100_ReadDiagnosticCurrent | Reads current from device diagnostic register |
| Q100_ReadDiagnosticVoltage | Reads voltage from device diagnostic register |
| Q100_ReadTemperature | Reads central temperatures from device diagnostic register |
| Q100_ReadPiPwmDutyCycle | Reads device duty cycle of proportional-integral controller |
| Q100_ReadFeedbackCurrentTon2 | Reads device feedback current in Ton/2 |

For a more detailed description of software driver API (function signatures, parameters) refer to the programmer's guide, included in the Q100 software driver zip file, or to the comments embedded in the Q100.h file.

### 3.3 Required driver setup

In order to execute correctly, the Q100 software driver requires the following:

1. Fill in the driver configuration structure (Q100_driver_t).
2. Implement the external functions of **Q100/Q100.h**.

Moreover, the driver requires correctly pin-muxed pins and correctly initialized SPI periphery, which is handled by the external function *Q100_SPI_Transfer*:

- **SPI:** 16 bits/frame, MSB first, clock polarity: active high, clock phase: capture on the 2nd edge, active low chip select.
- SW controlled **CSB_M** chip select (if HW chip select is not possible): GPIO output pin, active low, initial value: high.
- SW controlled **RESET_B** pin, which is active in low, initial value high.

Additionally, some applications may require another set of correctly pin-muxed Q100 pins:

- **FAULTB_M:** Interrupt pin
- **INx:** Direct input pins for all channels

For more details refer to the provided example project.

### 3.4 Implementation notes

Q100 Extreme Switch SW driver is based on variables of *bool*, *uint8_t*, *uint16_t* and *float* types and requires standard *stdbool.h*, *stdint.h* and *stddef.h* libraries.

Each data received via MISO pin is parsed in order to get *QuickSpi* status flags. Fault or warning status flags are always saved into *qSpiStatus* member of the *Q100_driver_t* structure and can be read directly from the structure. There are three different SPI transfer functions, depending on R/W action on *Control register bank* and R *Diagnostic register bank*. Q100 driver automatically toggles the bit by each SPI transfer action. The user does not need to care about it.

## 4 Installing the software

This section describes installation of S32 Design Studio for ARM and shows how to use this SW driver with S32K144 and S32K14x SDK for application development. A process of adding the Q100 SW driver to an existing project in different IDEs or with use of different MCUs should be analogical. Most likely, the addition of a low-level SDK driver will vary.

### 4.1 Installing IDE

This procedure explains how to obtain and install the latest version of S32 Design Studio for ARM (2018.R1).

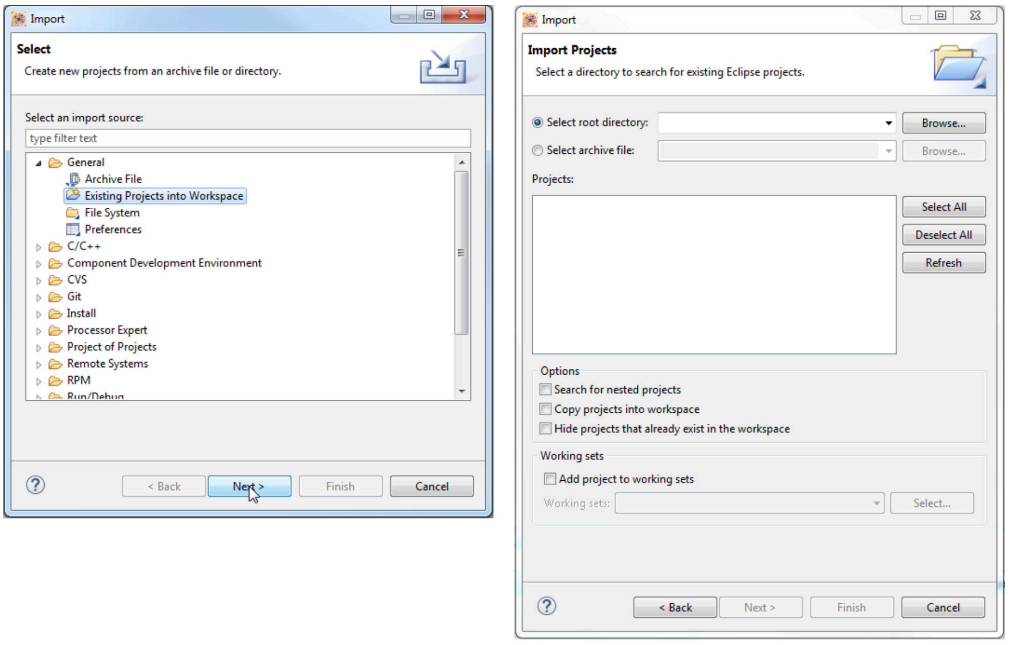
***Note:*** *The example in the driver package is intended for S32 Design Studio for ARM 2018.R1. If the selected IDE is already installed on the system, skip this section.*

1. Obtain the latest S32 Design Studio for ARM 2018.R1 installer file from the NXP website here: www.nxp.com/S32DS
2. Run the executable file and follow the instructions.

## 4.2 Import an example project into IDE

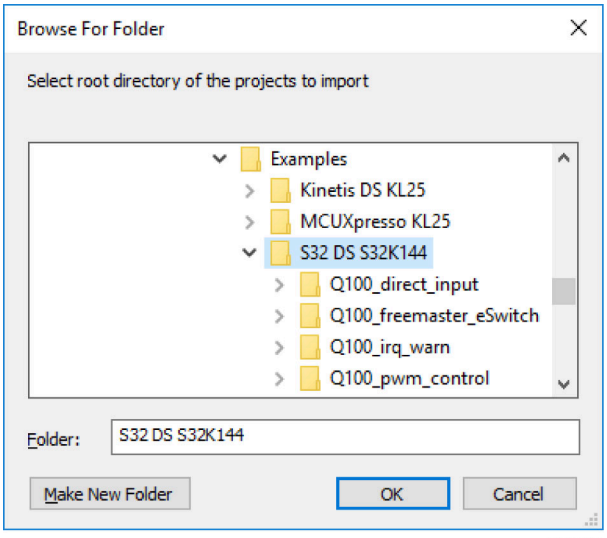
The following steps show how to import an example from the downloaded zip file into S32 Design Studio for ARM.

1. In the S32 Design Studio menu bar, click **File** → **Import…**. In the pop-up window, select **General** → **Existing Projects into Workspace** and click **Next**.



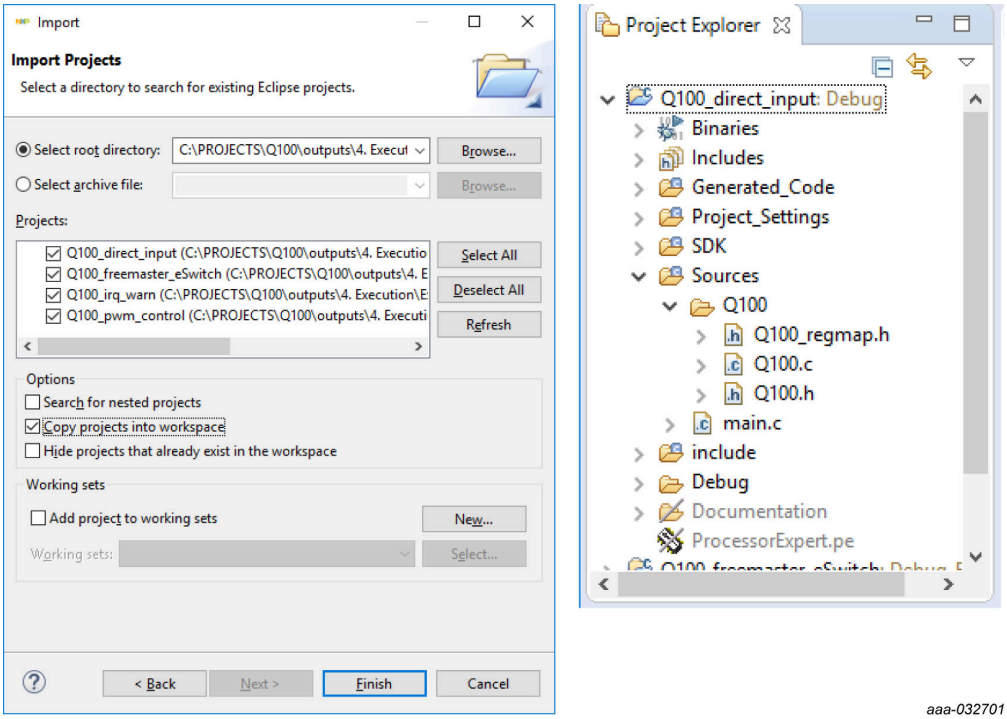

aaa-032699

2. Click **Browse** and locate the folder where you unzipped the downloaded example files. Find the folder **S32DS_Examples** and select a project to import. Then click **OK**.




aaa-032700

3. With your project now loaded in the **Select root directory** box, click on the **Copy projects into workspace** checkbox. Then click **Finish**. The project is now in the S32 Design Studio workspace where you can build and run it.



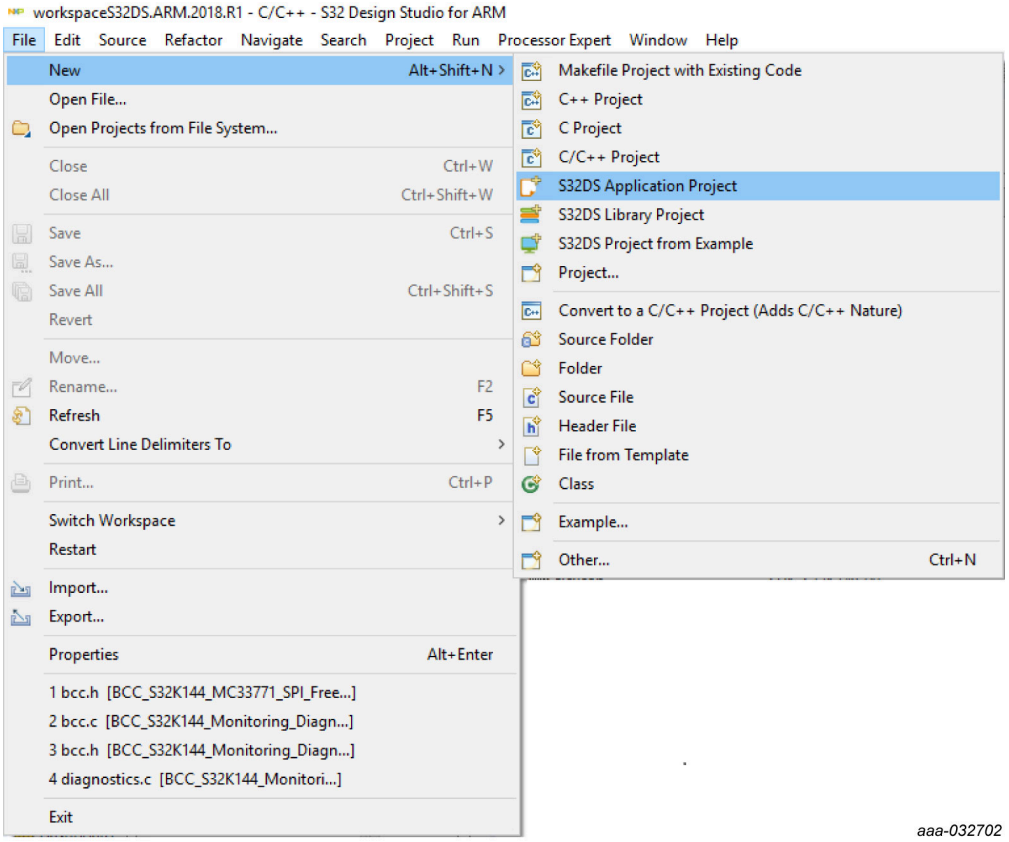

aaa-032701

## 4.3 Creating a new project with an MSDI software driver

If you choose not to use the example project, the following instructions describe how to create and set up a new project for S32K144 MCU that uses the MSDI SW driver.

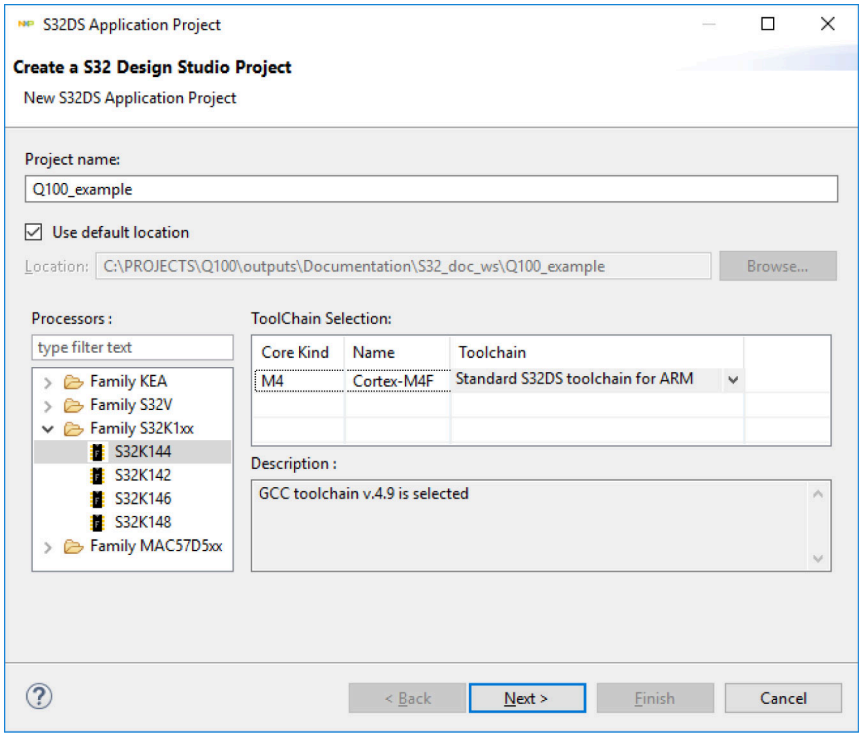
To create a new project in S32 Design Studio for ARM, do the following:

1. In the S32 Design Studio menu bar, select **File → New → S32DS Application Project**.



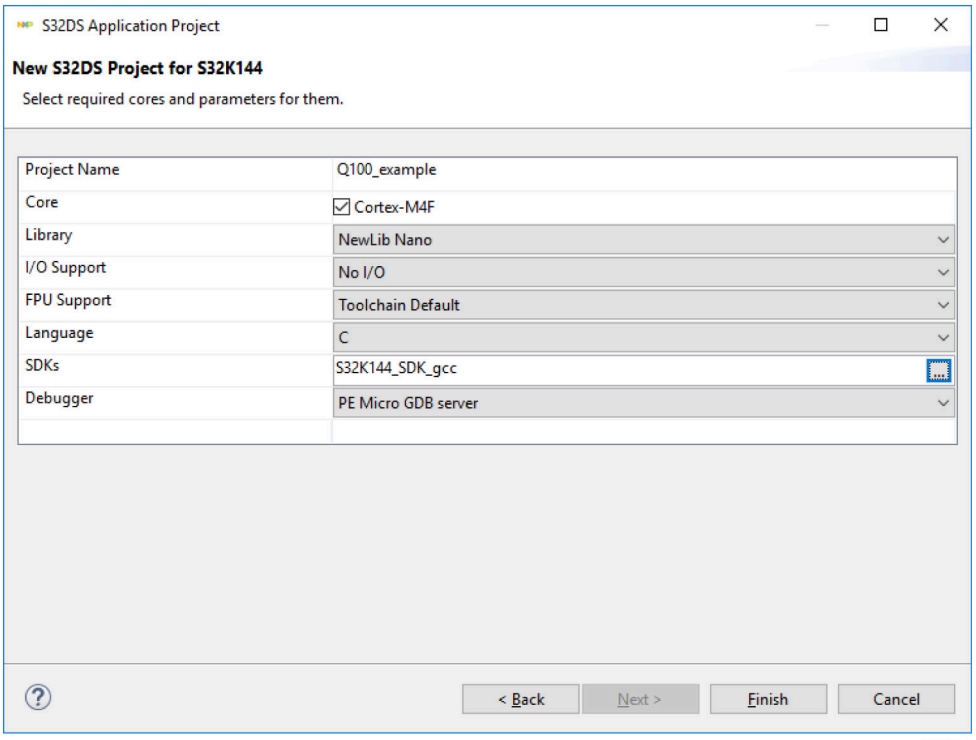

aaa-032702

2. When the **S32DS Application Project** box opens, enter a project name into the text box, choose **S32K144** processor in the *Processors* tab, **Standard S32DS toolchain for ARM** in *ToolChain Selection* and click **Next**.



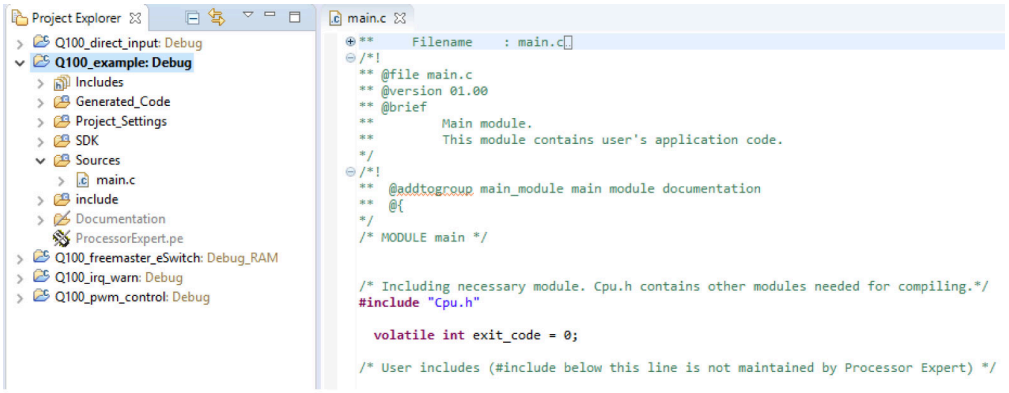

aaa-032703

3. Select **NewLib Nano** library, **S32K144_SDK (version 0.8.6)** SDK and click **Finish**.




aaa-032704

4. The Project Explorer panel and a part of the **main.c** content after creation of new project is shown. This project includes only startup code and minimal driver set from S32K144 SDK.

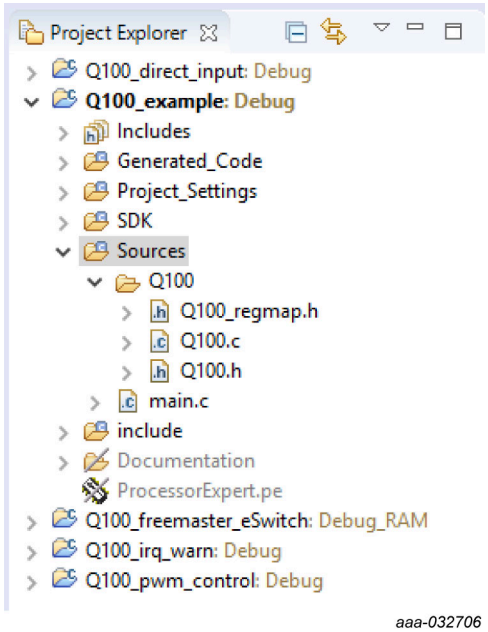



aaa-032705

### 4.3.1 Adding the Q100 eSwitch software driver to the project

This section describes how to add the Q100 software driver to the project.

1. Copy the content of **SDK_SW_Driver** to the **Sources** folder in your newly created project as shown.




aaa-032706

2. Include the *Q100.h* header file in **main.c** in order to get access to the Q100 software driver in the user code.

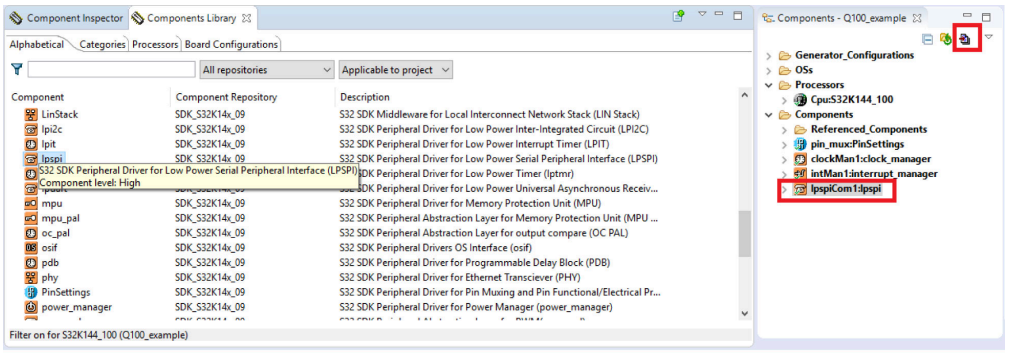
```
/* Including necessary module. Cpu.h contains other modules needed for compiling.*/
#include "Cpu.h"

  volatile int exit_code = 0;

/* User includes (#include below this line is not maintained by Processor Expert) */
#include "Q100/Q100.h"

/*!
  \brief The main function for the project.
  \details The startup initialization sequence is the following:
```

aaa-032707

### 4.3.2 Setting up the project

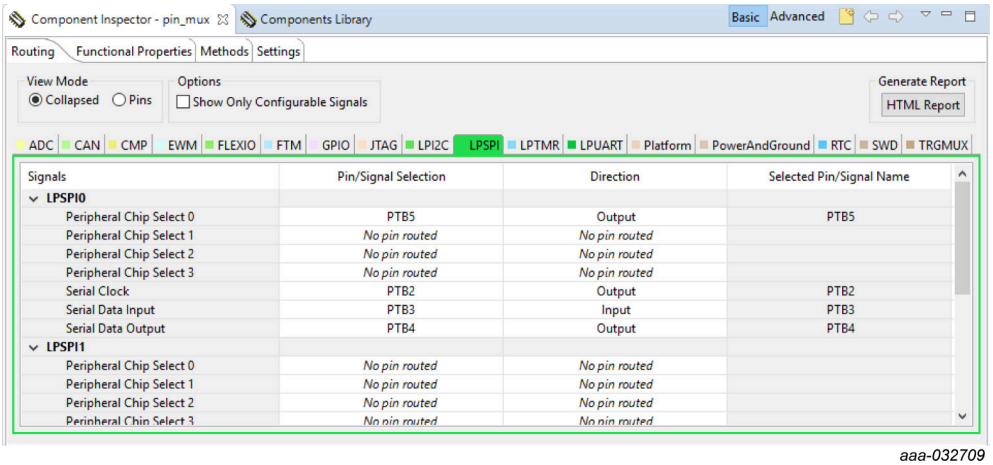
Once the new project has been created and the Q100 software driver has been added into it, the project must be set up.

1. In order to implement the platform specific (external) functions of the Q100 software driver, LPSPI and GPIO S32K14x SDK drivers are required. GPIO driver is already attached to the project. In order to generate the LPSPI driver into the project, click twice on the **lpspi** component in **Components Library** window. If the **Components Library** window is hidden, open it by clicking **Processor Expert** → **Show Views**. When the lpspi component is included in the PEx file, it is shown in the **Components** window.



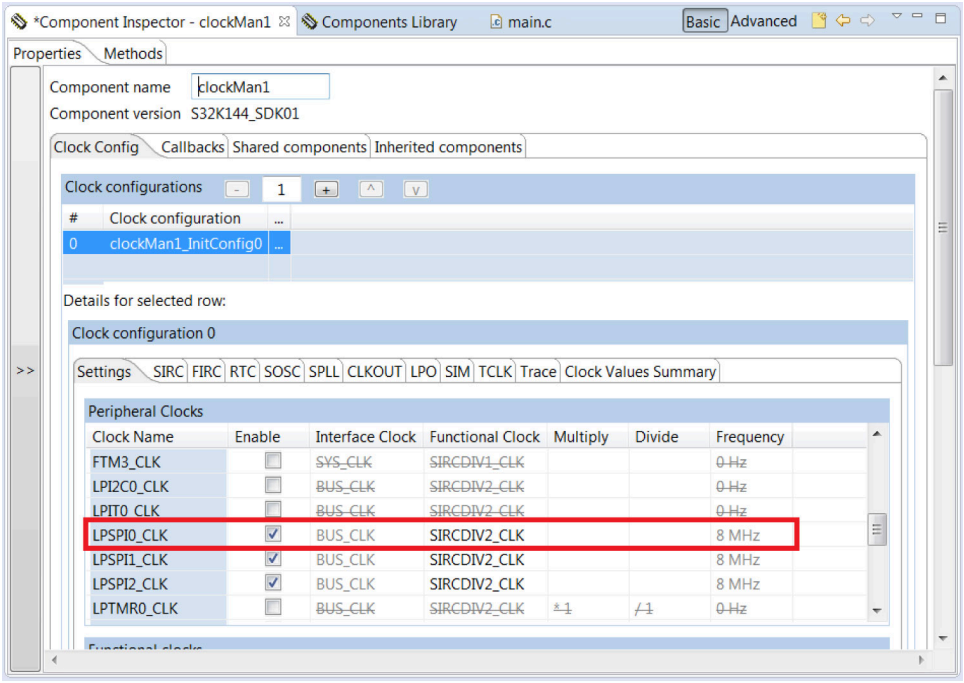

aaa-032708

2. In order to get the Q100 software driver to run on S32K144, the **PinSettings** component must be edited to configure the LPSPI and GPIO pins being used and the RESET_B signal. This entails making the correct MCU pin selections and then muxing them as needed. See Section 2.3. The following image is an example of LPSPI0 pin muxing. You should also set the correct GPIO pin directions and initial values in the **PinSettings** component.




aaa-032709

3. In order to get the Q100 software driver to run on an S32K144, check the **clock_manager** component settings for the following:
   - The peripheral clock to LPSPI and PORT peripheries must be enabled AND
   - The frequency for theses peripheries must be in the allowed range




aaa-032710

4. In the **Components** window, click the **Generate Processor Expert Code** icon to generate the component settings into the **Generated_Code** folder.
   In order to set the clock configuration and mux the pins according to the settings generated from the **Processor Expert** components, add the following lines to the beginning of the main() function:

```
CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
```

```
CLOCK_SYS_UpdateConfiguration(0U,
CLOCK_MANAGER_POLICY_FORCIBLE);
PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);
```

5. Create a variable of type **Q100_driver_t** that will be passed to all used functions. This variable stores Q100 software driver configuration and its internal data. This variable must be accessible during run time and should be declared either in the **main()** function or as a global variable.
6. Configure the Q100 as shown. You may change all individual items as needed.

```
int main(void)
{
    lpspi_state_t lpspiState;
    lpuart_state_t lpUartState;

    /* Main driver structure where are all configurations stored. */
    Q100_driver_t q100Drv;

    /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
    #ifdef PEX_RTOS_INIT
    PEX_RTOS_INIT();                 /* Initialization of the selected RTOS. Macro is defined b
    #endif
    /*** End of Processor Expert internal initialization.                    ***/

    /* Initialization of board. */
    CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
                   g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
    CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_FORCIBLE);

    PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);

    /* Initialization of SPI instance 0. In this example is only one device and SPI used. */
    LPSPI_DRV_MasterInit(Q100_SPI_INSTACE, lpspiState , &lpspiCom1_MasterConfig0);

    /* Next configure peripheral delay between active level of chip select signal and clock.
     * Next delay between CLK and de-active CS signal.*/
    LPSPI_DRV_MasterSetDelay(Q100_SPI_INSTACE, 5, 5, 5);

    /* Filling driver data structure by default values, disabled
     * transition to save mode if watchdog timeout event occur. */
    Q100_FillDataStructureBy(Q100_INSTAMCE_0, &q100Drv, Q100_DEFAULT_SETTINGS, false);

    /* Reset device. */
    PINS_DRV_ClearPins(Q100_RSTB_PORT, (1 << Q100_RSTB_PIN));
    OSIF_TimeDelay(1);
    /* Activate device. */
    PINS_DRV_SetPins(Q100_RSTB_PORT, (1 << Q100_RSTB_PIN));
    OSIF_TimeDelay(1);

    q100Drv.devCtrlData.wtchdgConf.enable = false;
    q100Drv.devCtrlData.unOvVoltConf.overVoltage[Q100_CHANNEL_0] = Q100_UserCtrlVoltToRaw(15.5);

    /* Initialize device by configuration stored in driver structure. */
    Q100_Init(&q100Drv);
```

Q100 driver structure

SPI Initialization

Default structure configuration

User drv config modification

Initialization of Q100 device with driver data configurations

*aaa-032711*

7. Set up the LPSPI peripherals that will be used by the Q100 software driver. The easiest way is to set the LPSPI configuration in the **lpspi** Processor Expert component. After clicking the **Generate Processor Expert Code** icon, the lpspi configuration is generated in the Generated_Code folder. This configuration can be passed as a parameter of *LPSPI_DRV_MasterInit* SDK function in **main()**. In addition, it is recommended to set the Chip Select To Clock Delay (CSTCD) configuration and Delay Between Frames (DBF) configuration by LPSPI_DRV_MasterSetDelay when using the HW chip select.
8. The Q100 initialization function should be called. User must pass reference to driver configuration structure.

9. Except for the peripheral and Q100 initialization function, the external function (listed in Q100.h and Table 2) used by Q100 driver need to be implemented. See the *S32 DS S32K144* folder in the provided example project as a template of its implementation for S32K144.

### 4.3.3 Writing your application code

All of your application code must reside in the **Sources** folder in your project directory. You may modify the code in **main.c** but you must retain the original comments related to usage directions.

When the Q100 SW Driver and utilized peripherals are configured properly and external functions are implemented, you can use all of the prepared API functions to construct your own application.

See the *API Programmer's Guide* (included in the Q100 software driver zip file) for function signatures and required parameters. Also review the **Q100.h** headerfile, which contains prototypes and explanation for all available functions.

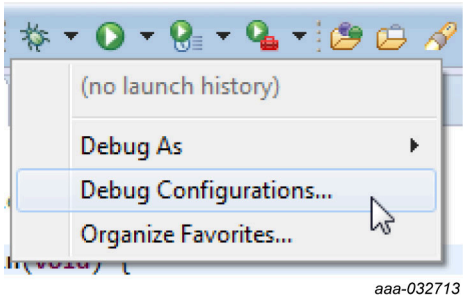
### 4.3.4 Compiling, downloading and debugging

To compile a project, click the compile icon in the toolbar.

aaa-032712

The process for downloading an application on board in S32 Design Studio for ARM may differ according to used MCU board. If you have any question, please see S32 Design Studio for ARM user's guide. To download and debug on S32K144EVB-Q100 MCU board, do the following:

1. Click the arrow next to the debug icon in the toolbar and select **Debug Configurations…**.




aaa-032713

2. In the **Debug Configurations** dialog box, select one of the existing configurations with a project name under **GDB PEMicro Interface Debugging**.
3. Make sure that the **C/C++ Application** contains a path to the .elf file of the project in the **Main** tab of the Debug Configuration window.
4. Pick up proper **debug interface** and **USB port** in the **Debugger** tab of Debug Configuration window.

5. Apply changes and then click **Debug**. S32 Design Studio for ARM will download and launch the program on board.

# 5 References


| | | | |
|---|---|---|---|
| [1] | **FRDM-XS2410EVB** | Product Summary Page | http://www.nxp.com/FRDM-XS2410EVB |
| [2] | **Q100 eSwitch software driver** | Tool Summary Page | http://www.nxp.com/Q100-ESWITCH-DRIVER |
| [3] | **S32 Design Studio IDE** | Tool Summary Page | http://www.nxp.com/S32DS |

# 6 Legal information

## 6.1 Definitions

**Draft** — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

## 6.2 Disclaimers

**Limited warranty and liability** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

**Right to make changes** — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Export control** — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

**Evaluation products** — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

**Translations** — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

## 6.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

**NXP** — wordmark and logo are trademarks of NXP B.V.

# Tables



# Figures

# Contents





For more information, please visit: http://www.nxp.com

**Date of release: 5 September 2022**
**Document identifier: UM11182**