CANopenNode

Generated by Doxygen 1.8.13

Contents

Chapter 1

CANopen Sensor

Application description

This application simulates the remote sensor using the two buttons as temperature control and displays the state of unit using LEDs.

- Button 1 (**SW3**) decreases the temperature by 2 degrees each push;
- Button 2 (**SW2**) increases the temperature by 2 degrees each push;
- GREEN LED is on when the Unit is idle;
- CYAN LED is on when the Cooling unit is on;
- ORANGE LED is on when the Heating unit is on.

The application initializes the clock, GPIO pins and moves on to initializing the CANopenNode protocol:

- Firstly the FlexCAN user configuration is sent to the CANopen driver;
- Then CANopenNode objects are intiliazed and the drivers takes care of setting up the mailboxes for the FlexCAN;
- LPIT is set up and started to run the timer thread of the CANopenNode protocol.
- This loop is where all other code that needs to execute on the CPU goes or in the timer thread if the code needs to be executed in real time.

The node configuration is done using the Object Dictionary(OD) wich is specific for each node on the network:

- The node ID is set in the main routine to 0x06.
- Two variables are created in the OD for storing the sensor temperature and unit state.
- At index 0x3000 is stored the variable **Sensor_Temperature** as int16 which can be mapped by an RPDO.
- At index 0x3100 is stored the variable **Unit_State** as int8 which can be mapped by an TPDO.
- Heartbeat feature is activated by seeting a value of 5000 to the index entry 0x1017, the heartbeat message is sent once every 5 seconds.
- The second TPDO(TPDO_1) at index 0x1801(Communication object) and 0x1A01(Mapping object) is set to send one variable stored at 0x3000 with lenght 16bits once every second.
- The first RPDO(RPDO_0) at index 0x1400(Communication object) and 0x1600(Mapping object) is set to receive asynchronously one variable with lenght 8bits from the network node with ID 0x01 (the messageID is 0x181) and store it at 0x3100.

Prerequisites

To run the example you will need to have the following items:

- 1 S32K144 boards
- 1 Power Adapter 12V
- 3 Dupont female to female cable
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

Boards supported

The following boards are supported by this application:

- S32K144EVB-Q100

Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K144EVB-Q100
CAN HIGH (∗)	J13.1 - J13.1 on the second board
CAN LOW (∗)	J13.2 - J13.2 on the second board
BUTTON 1 (**PTC13**)	SW3 - wired on the board
BUTTON 2 (**PTC12**)	SW2 - wired on the board
RED_LED (**PTD15**)	RGB_RED - wired on the board
GREEN_LED (**PTD16**)	RGB_GREEN - wired on the board
BLUE_LED (**PTD0**)	RGB_BLUE - wired on the board
GND (**GND**)	J13.4 - J13.4 on the second board

(∗) Those lines must be modulated using a transceiver, if it is not specified the boards already include the CAN transceiver

Note

For the CAN transceiver to work the board must be powered using a 12V power adapter and the jumper J107 must be in position 1-2

How to run

1. Importing the projects into the workspace

After opening S32 Design Studio, go to **File** -> **New S32DS Project From**... and select **CANopen_Sensor_↩ s32k144**. Then click on **Finish**.
The projects should now be copied into you current workspace. If a message is displayed about the project being created for another SDK click **Continue loading**. A second message box about SDK reference not being correct should appear. Please click **Yes** and in the opened windows click **Browse** and go to the SDK instalation directory.

2. Generating the Processor Expert configuration

First go to **Project Explorer** View in S32 DS and select the project(**CANopen_Sensor_s32k144**). Then go to **Project** and click on **Generate Processor Expert Code**
Wait for the code generation to be completed before continuing to the next step.

3. Building the project

Before building select CANopen_Sensor_s32k144 project and go to **Project** -> **Properties**. Under **Resources** -> **Linked Resources** please make sure that the path variable **CANopenNode_PATH** points to CANopenNode git repository and the **S32_SDK_PATH** point to the SDK instalation folder. Select the **FLASH** (Debug_FLASH) configuration to be built by left clicking on the downward arrow corresponding to the **build** button(.
Wait for the build action to be completed before continuing to the next step.

4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
CANopen_Sensor_s32k144_Debug_FLASH_Jlink	Debug the FLASH configuration using Segger Jlink debuggers
CANopen_Sensor_s32k144_Debug_FLASH_PE↩ Micro	Debug the FLASH configuration using PEMicro debuggers

Select either **Debug_FLASH Jlink** or **Debug_FLASH PEMicro** debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.
Use the controls to control the program flow.

Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

Chapter 2

CANopen Unit

Application description

This application simulates the air conditioning unit using the two buttons as user temperature control and displays the range of the temperature received from sensor using LEDs.

- Button 1 (**SW3**) decreases the temperature by 2 degrees each push;
- Button 2 (**SW2**) increases the temperature by 2 degrees each push;
- RED LED is on when the temperature received from the sensor is above 50 degrees;
- ORANGE LED is on when the temperature received from the sensor is between 31 and 49 degrees;
- GREEN LED is on when the temperature received from the sensor is between 21 and 30 degrees;
- CYAN LED is on when the temperature received from the sensor is between 1 and 20 degrees;
- BLUE LED is on when the temperature received from the sensor is under 0 degrees.

The application initializes the clock, GPIO pins and moves on to initializing the CANopenNode protocol:

- Firstly the FlexCAN user configuration is sent to the CANopen driver;
- Then CANopenNode objects are intiliazed and the drivers takes care of setting up the mailboxes for the FlexCAN;
- LPIT is set up and started to run the timer thread of the CANopenNode protocol.
- Then the main enters a loop until the connection is reset or the device needs to shut down.
- This loop is where all other code that needs to execute on the CPU goes or in the timer thread if the code needs to be executed in real time.

The node configuration is done using the Object Dictionary(OD) wich is specific for each node on the network:

- The node ID is set in the main routine to 0x01.
- Two variables are created in the OD for storing the sensor temperature and unit state.
- At index 0x3000 is stored the variable **Sensor_Temperature** as int16 which can be mapped by an TPDO.

- At index 0x3100 is stored the variable **Unit_State** as int8 which can be mapped by an RPDO.
- Heartbeat feature is deactivated. But the monitoring of heartbeat messages from network nod with ID 0x01 is set to be done once every 7,5 seconds (1,2∗message send time). This setting is done in index 0x1016 sub index 0x01 by setting bits 0-15 with the consumer time in ms and bits 16-23 with the node id monitored.
- The first TPDO(TPDO_0) at index 0x1800(Communication object) and 0x1A00(Mapping object) is set to send one variable stored at 0x3100 with lenght 8bits every time its value gets changed.
- The second RPDO(RPDO_1) at index 0x1401(Communication object) and 0x1601(Mapping object) is set to receive asynchronously one variable with lenght 16bits from the network node with ID 0x06 (the messageID is 0x286) and store it at 0x3000.

Prerequisites

To run the example you will need to have the following items:

- 1 S32K144 boards
- 1 Power Adapter 12V
- 3 Dupont female to female cable
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

Boards supported

The following boards are supported by this application:

- S32K144EVB-Q100

Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K144EVB-Q100
CAN HIGH (∗)	J13.1 - J13.1 on the second board
CAN LOW (∗)	J13.2 - J13.2 on the second board
BUTTON 1 (**PTC13**)	SW3 - wired on the board
BUTTON 2 (**PTC12**)	SW2 - wired on the board
RED_LED (**PTD15**)	RGB_RED - wired on the board
GREEN_LED (**PTD16**)	RGB_GREEN - wired on the board
BLUE_LED (**PTD0**)	RGB_BLUE - wired on the board
GND (**GND**)	J13.4 - J13.4 on the second board

(∗) Those lines must be modulated using a transceiver, if it is not specified the boards already include the CAN transceiver

Note

For the CAN transceiver to work the board must be powered using a 12V power adapter and the jumper J107 must be in position 1-2

How to run

1. Importing the projects into the workspace

After opening S32 Design Studio, go to **File** -> **New S32DS Project From**... and select **CANopen_Unit_s32k144**. Then click on **Finish**.
The project should now be copied into you current workspace. If a message is displayed about the project being created for another SDK click **Continue loading**. A second message box about SDK reference not being correct should appear. Please click **Yes** and in the opened windows click **Browse** and go to the SDK instalation directory.

2. Generating the Processor Expert configuration

First go to **Project Explorer** View in S32 DS and select the project(**CANopen_Unit_s32k144**). Then go to **Project** and click on **Generate Processor Expert Code**
Wait for the code generation to be completed before continuing to the next step.

3. Building the project

Before building select CANopen_Sensor_s32k144 project and go to **Project** -> **Properties**. Under **Resources** -> **Linked Resources** please make sure that the path variable CANopenNode_PATH points to **CANopenNode** git repository and the **S32_SDK_PATH** point to the SDK instalation folder. Select the **FLASH** (Debug_FLASH) configuration to be built by left clicking on the downward arrow corresponding to the **build** button(.
Wait for the build action to be completed before continuing to the next step.

4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
CANopen_Unit_s32k144_Debug_FLASH_Jlink	Debug the FLASH configuration using Segger Jlink debuggers
CANopen_Unit_s32k144_Debug_FLASH_PEMicro	Debug the FLASH configuration using PEMicro debuggers

Select either **Debug_FLASH Jlink** or **Debug_FLASH PEMicro** debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.
Use the controls to control the program flow.

Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.