TLE9893_2QKW62S_UART_TRANSCEIVER

About this document

Scope and purpose

The aim of this guide is to present the scope, the implementation, the algorithm and a demonstration of the **TLE9893_2QKW62S_UART_TRANSCEIVER** example code for the TLE989x Infineon Embedded Power ICs based on Arm® Cortex® M3. This example code can be found in the Keil µVision Pack Installer.

The full functionalities and characteristics of the embedded power devices are described in the datasheets and user's manual. Please refer to these documents for more detailed information. Furthermore, a low level (line-by-line) description of the code is not the aim of this document, although occasionally some codeblocks might be reported if necessary to the comprehension.

Note: *The following information is given as a hint for the implementation of the system only and shall not be regarded as a description or warranty of a certain functionality, condition or quality of the referred devices or presented software example.*

Intended audience

Design engineers, system engineers, embedded power designers

Table of contents

About this document **1**
Table of contents **1**
1 Introduction **2**
2 Hardware **4**
3 Implementation **6**
3.1 Get the example via the Pack Installer for Keil 6
3.2 Configuration 8
3.3 Sample code 10
References **13**
Revision history **14**

1 Introduction

A command menu is written to the standard output (see Figure 1). A user-defined stdout target as well as a user-defined stdin target is implemented in the UART module. The redirection is established via the compiler-I/O-STDOUT RTE and compiler-I/O-STDIN RTE settings.

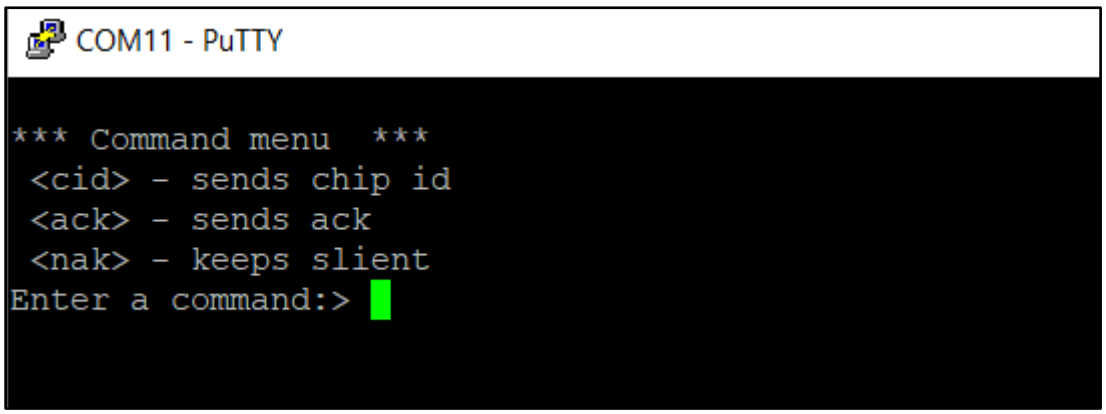

Figure 1 Command menu

The pin P1.1 used as output is one debug pin on the USB device. The pin P1.2 used as input is another debug pin on the USB device.

Figure 2 shows the transmission of the ack command including the echo which can be seen on the UART transmit and receive buffer registers. The ACK response can be seen on the transmit buffer register immediately after the ack command is received.

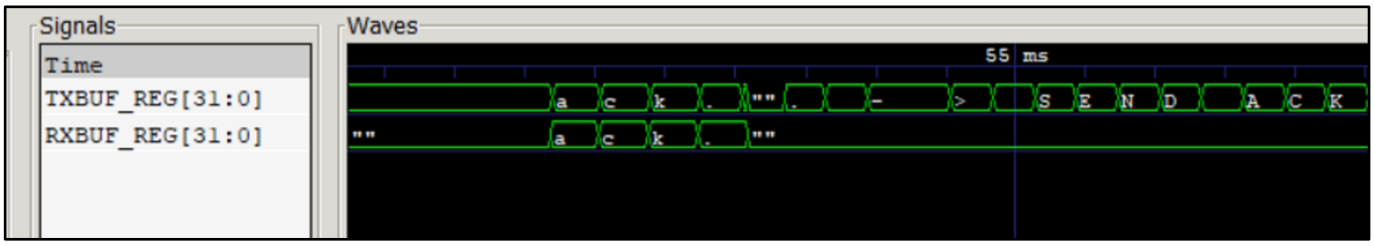

Figure 2 Transmission of ack command

The TLE9893_2QKW62S_UART_TRANSCEIVER example has implemented three commands for demonstration purpose:

- cid – returns the chip ID (demonstrates scenario request & response with individual data)
- ack – returns an acknowledgement (demonstrates scenario request & acks request without data)
- nak – keeps silent (demonstrates scenario request & no response)
- any other command will be ignored and an error response is sent out.

Figure 3 shows the execution of all those described commands and their appropriate expected response.

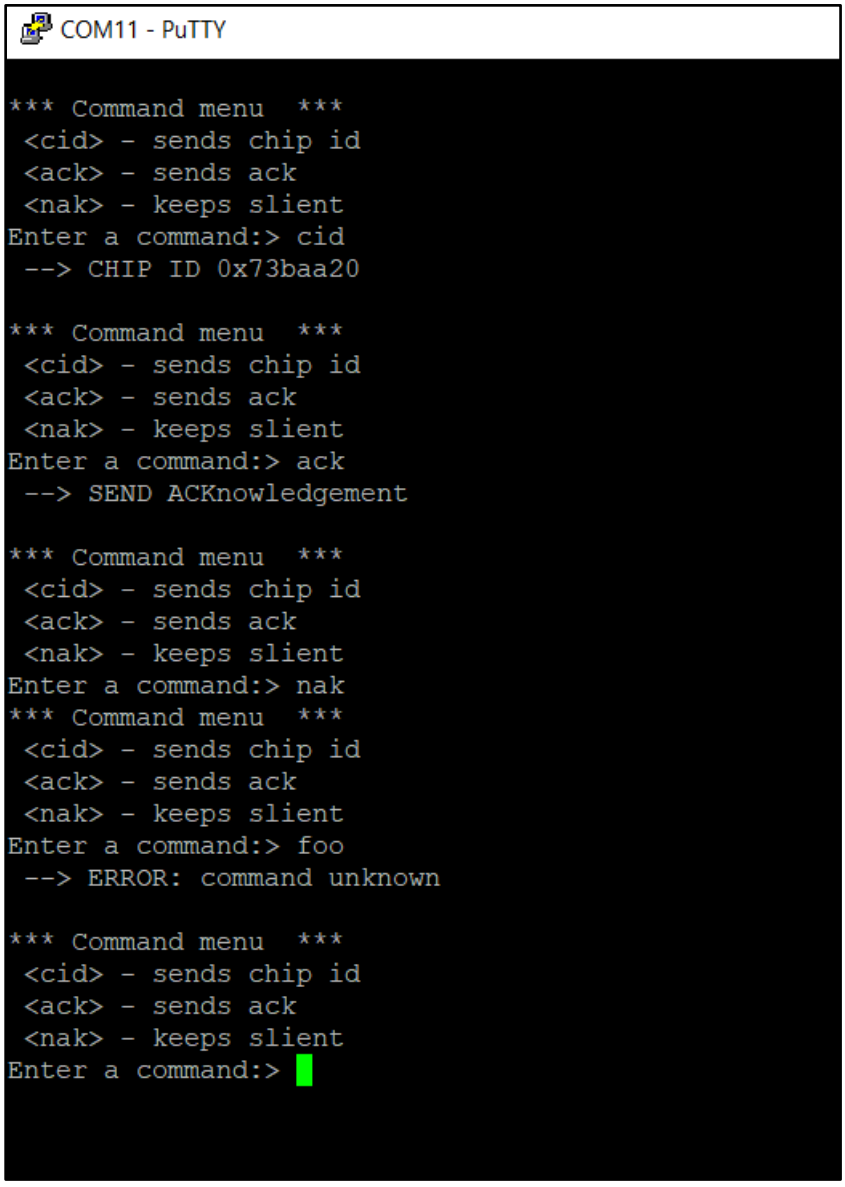

Figure 3 All commands executed with their expected response

2 Hardware

This chapter shows how to run the TLE9893_2QKW62S_UART_TRANSCEIVER example with the TLE988X/TLE989X evaluation board. For this the project must be opened and compiled.

Figure 2 shows the TLE988X/TLE989X evaluation board. The application code must be loaded via a debugger (e.g. ULINK or J-Link) to the board. The board must be powered with 12V (red and black connections).

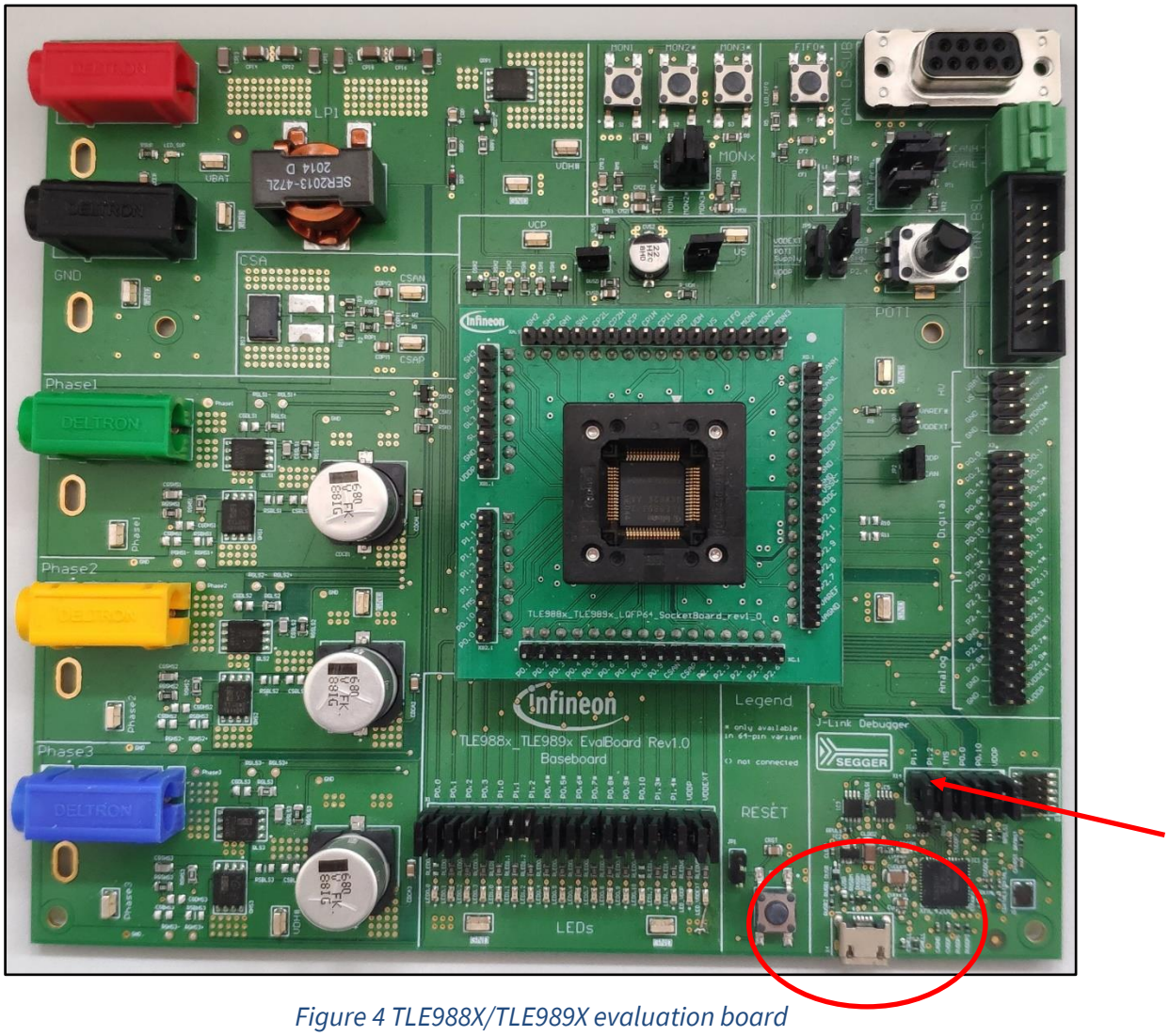

Figure 4 TLE988X/TLE989X evaluation board

The output pin P1.1 and the input pin P1.2 can be tapped on the red arrow.

Alternatively, a USB connection can be established to a local PC, which emulates a virtual COM port. The relevant COM device number can be identified via the Device Manager on Windows systems or the dmesg tool on Unix based operating systems.

In order to show the output on a command console, free tools like Putty or TeraTerm can be used. The UART1 in this example is configured with:

- a transmission baud rate of 115200,
- 8 data bits,
- 1 stop bit,
- no parity and no flow control.

3 Implementation

This chapter shows the process to follow to get a working TLE9893_2QKW62S_UART_TRANSCEIVER example.

3.1 Get the example via the Pack Installer for Keil

Open the Pack Installer within the Keil IDE. See Figure 5 below.

Choose the appropriate device (here TLE9893_2QKW62S) on the left-hand side. On the right-hand side, select the tab Examples, where you can access the TLE9893_2QKW62S_UART_TRANSCEIVER example.

Clicking on “Copy” will copy the example on your computer and open it.

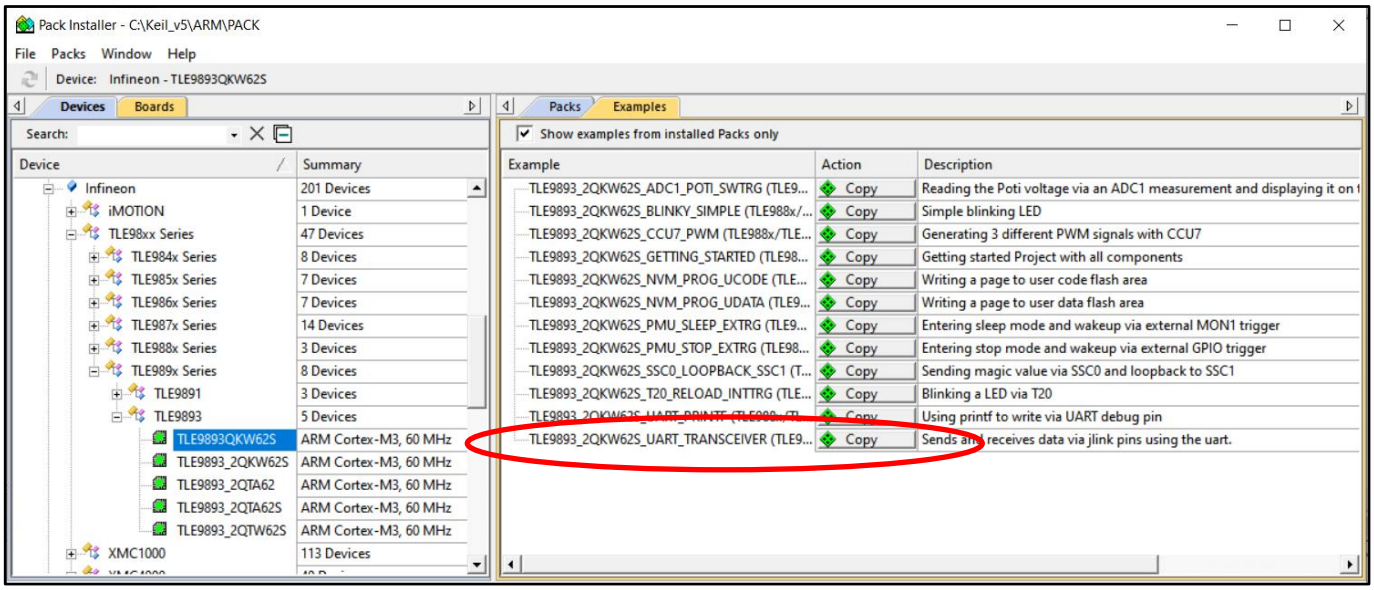

Figure 5 Keil Pack Installer

In order to redirect the stdout functions - the printf call in the example, adjust the runtime environment setting for the compiler within the Keil IDE. Select the option “User” under Compiler -> I/O -> STDOUT (see Figure 6).

In order to redirect the stdin functions - the stdin_getchar call in the example, adjust the runtime environment setting for the compiler within the Keil IDE. Select the option “User” under Compiler -> I/O -> STDIN (see Figure 6).

In the TLE9893_2QKW62S_UART_TRANSCEIVER example, the debug output and the debug input are shown. Therefore, the stdout and stdin is set. In case stderr or file access is necessary, this can be set in the Manage Run-Time Environment window as well.

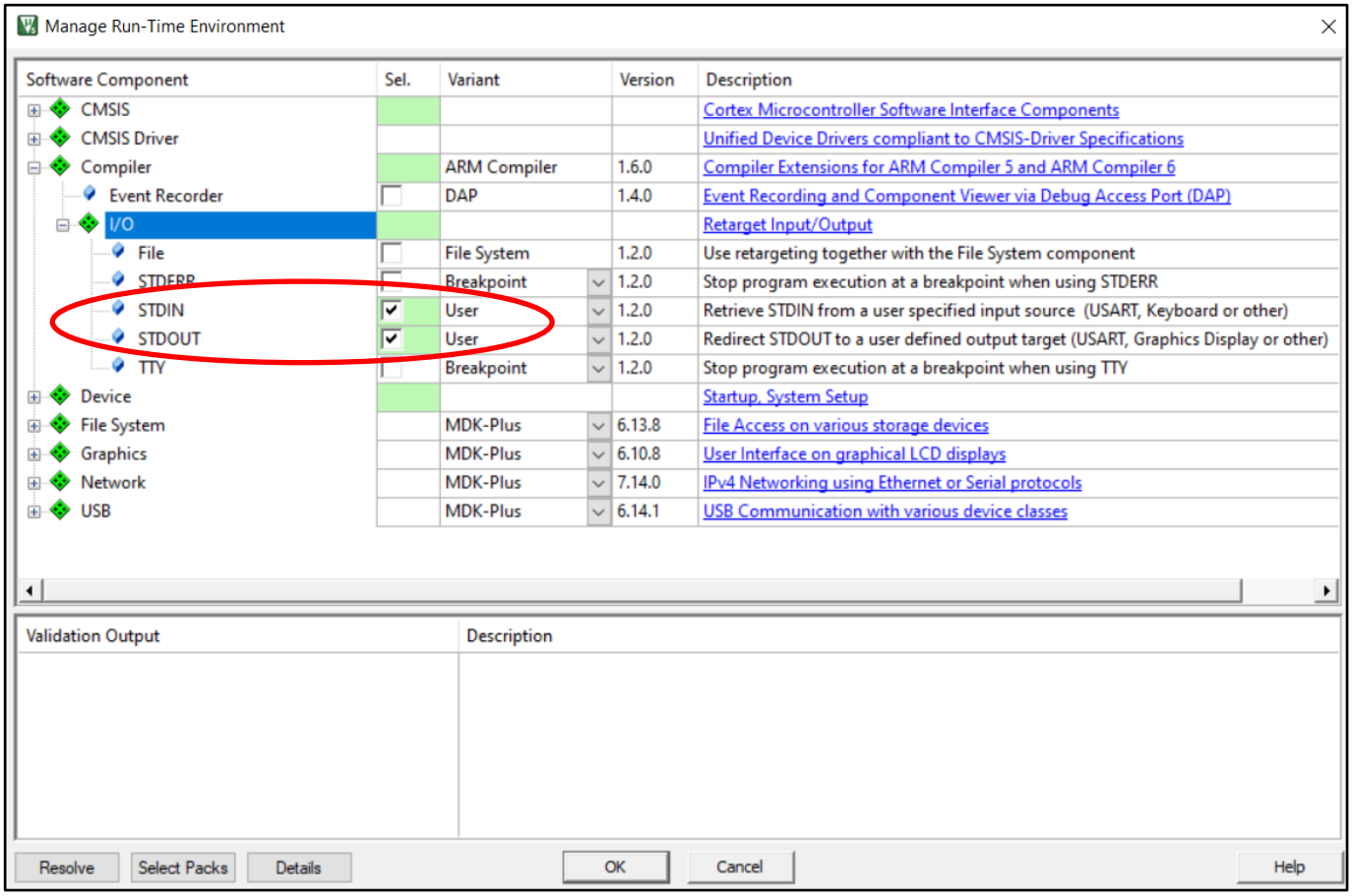

Figure 6 RTE settings for stdout and stdin

Within the runtime environment settings at the device folder tree, the correct set of the device design step must be approved. The design step is written on the device. By default, AA is selected. (see Figure 7).

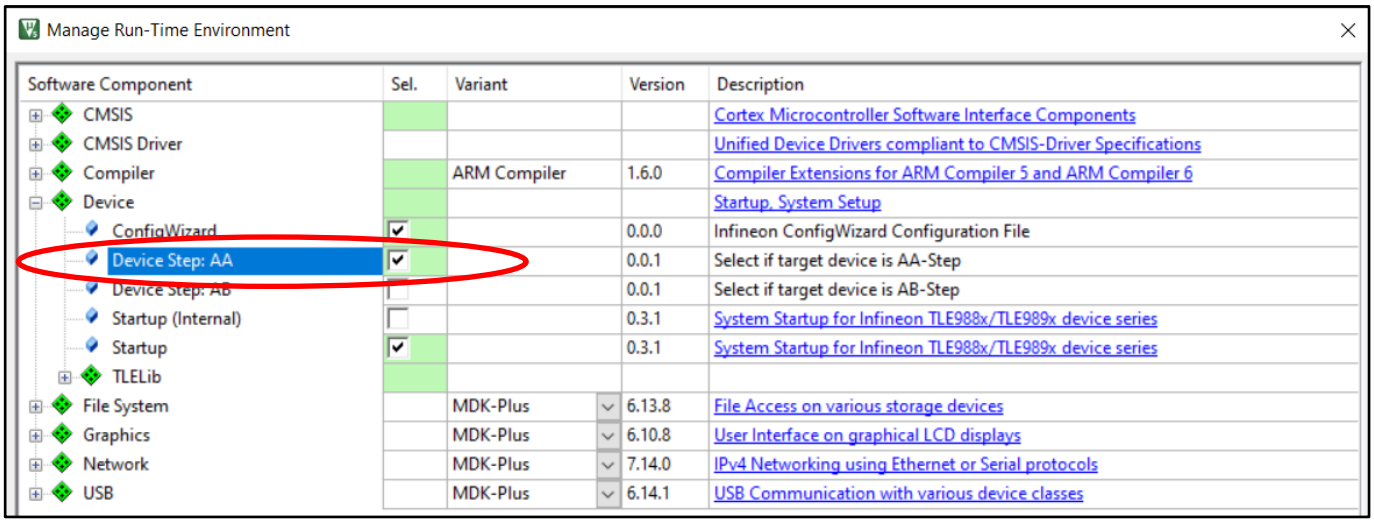

Figure 7 RTE settings for design step

3.2 Configuration

In order to see the configured output pin for the UART, start the tool Config Wizard can be started. It is available within the Keil IDE through a shortcut in the Tools menu.

The Config Wizard opens and shows an overall status of the current pin configuration. In Figure 8, the pin P1.1 is configured as the debug output and the pin P1.2 is configured as the debug input.

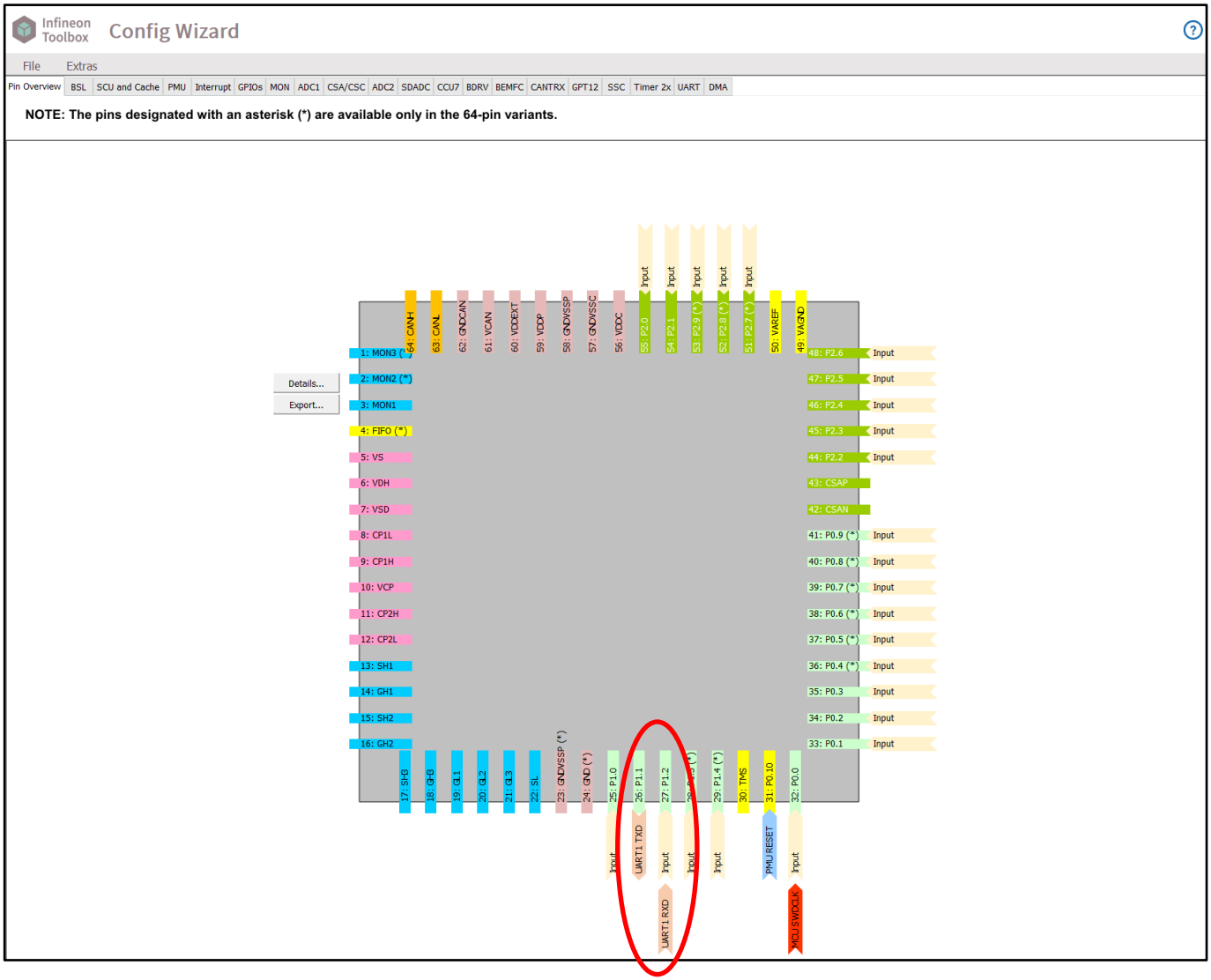

Figure 8 Config Wizard pin overview

In order to configure the UART module for the TLE9893_2QKW62S_UART_TRANSCEIVER example, select the UART tab.

Enable the UART1 module. Next, select the 8-bit UART mode with variable baudrate. The baudrate is set to 115200 in the blue box Baudrate Generator Settings. This is one of the common speed settings for the UART.

In the pink box Transmission Settings, select the pin P1.1.

In the green box Reception Settings, select the pin P1.2 and set the radio box Enable receiver of serial port.

In the yellow box Interrupt, enable both receive interrupt radio boxes and rewrite an intuitive name for the interrupt service routine, here uart_receive.

See Figure 9 Config Wizard, module UART for more details.

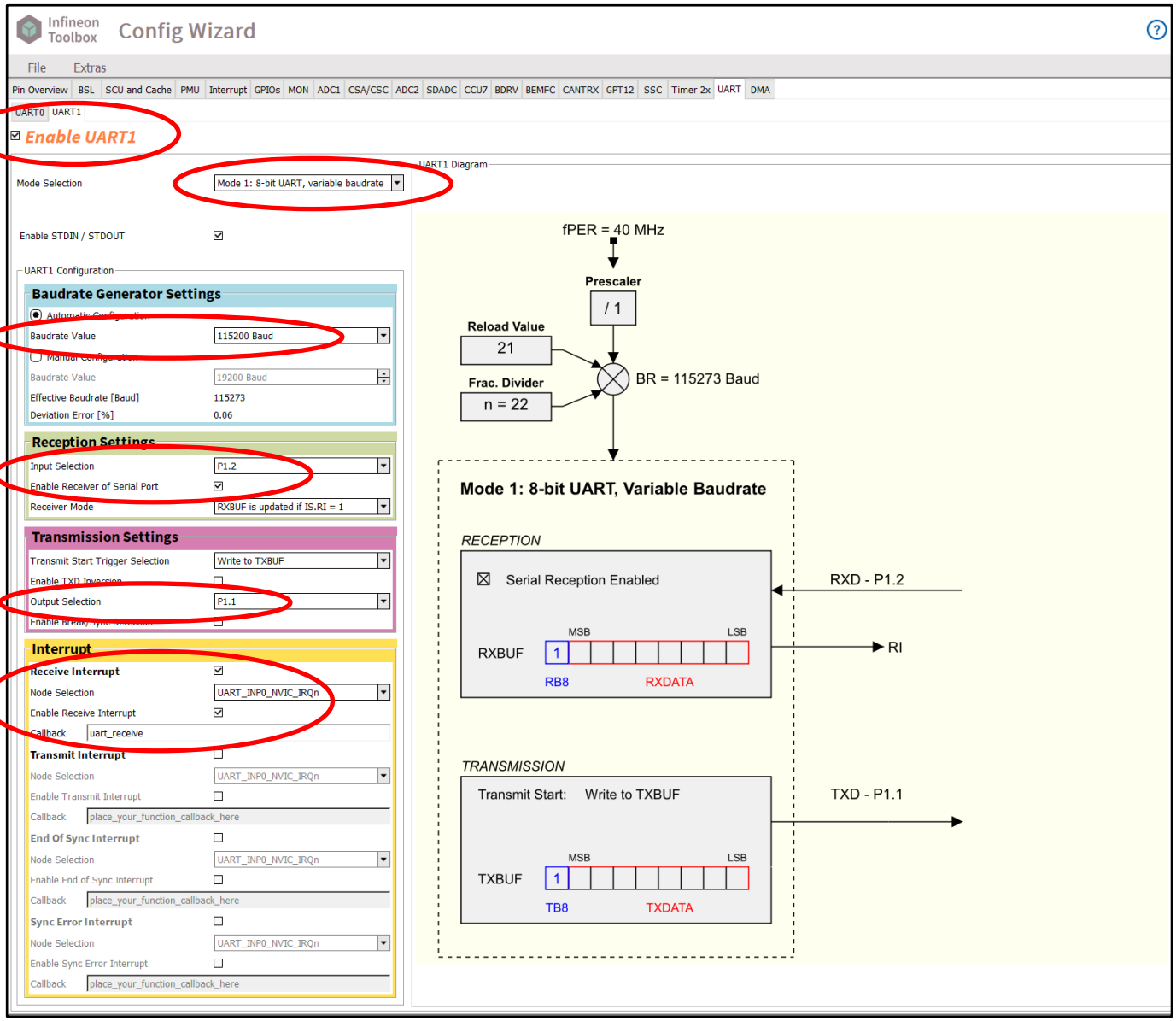

Figure 9 Config Wizard, module UART1

Finally, save your configuration to take these changes into account (File -> Save).

3.3 Sample code

Figure 10 shows the application code of the TLE9893_2QKW62S_UART_TRANSCEIVER example.

Within the endless loop, the `handleCommand()` method is called when the `b_cmdTrigger` value is true. The value will become true when a full command is received. This is set by the `uart_receive()` interrupt service routine (see Figure 11).

```
104   for (;;)
105   {
106     /* Main watchdog service */
107     (void) PMU_serviceFailSafeWatchdog();
108
109     /* Check for activated command handler trigger */
110     if (b_cmdTrigger == true)
111     {
112       /* Handle new command */
113       handleCommand();
114     }
115   }
116 }
```

Figure 10 Main loop in the application code

The `uart_receive()` interrupt service routine (see Figure 11) is called when a new byte is received and available in the uart receive buffer.

First, to avoid buffer overflows, the already received bytes without a newline character are checked against the maximum allowed `BUFFER_SIZE`. If the maximum is reached, the boolean `b_cmdTrigger` is set to true without receiving a newline or getting the pending character.

Otherwise the pending character is received and echoed to the stdout (to see the entered characters on the console). Next the received character is checked for a newline. If so the boolean `b_cmdTrigger` is set to true.

Finally, the receive counter is incremented before the interrupt service routine is finished.

```
/* UART receive ISR */
void uart_receive()
{
  /* Check for buffer overflow */
  if (u8_readCnt < BUFFER_SIZE)
  {
    /* Receive byte from P1.2 */
    u8_buffer[u8_readCnt] = (uint8) stdin_getchar();

    /* Echo byte to stdout to show character on console */
    printf("%c", u8_buffer[u8_readCnt]);

    /* Check if received character was a newline */
    if (u8_buffer[u8_readCnt] == 0x0d)
    {
      /* Add a final 0x00 to the string buffer */
      u8_buffer[u8_readCnt] = 0x00;

      /* Trigger command handler */
      b_cmdTrigger = true;
    }
    /* Increment receive counter for new character */
    u8_readCnt++;
  }
  else
  {
    /* Receive buffer is full -> handle command without newline */
    b_cmdTrigger = true;
  }
```

Figure 11 uart_receive() interrupt service routine

Once a full command is received, the `b_cmdTrigger` value is set to true and the `handleCommand()` method is called (see Figure 12).

Initally no character is received and the `u8_readCnt` is 0. In that case only the command menu is written to the stdout.

If a command is available (`u8_readCnt` is not 0), the command will be interpreted. In this example three commands are implemented: cid, ack, nak:

- The ‘cid’ command reads out the device chip identifier and sends it back to the stdout.
- The ‘ack’ command returns an acknowlegdement to the stdout.
- The ‘nak’ command just accepts the command and returns nothing.

In case the command cannot be interpreted, an error is written to the stdout.

Once the command is interpreted and handled, the command menu is written to the stdout, the `u8_readCnt` is set to 0 and the `b_cmdTrigger` is set to false. The system is now waiting for a new command again.

```
118  /* Command handler */
119  void handleCommand()
120  {
121    /* Declare and initiate chip ID value */
122    uint32 u32_chipId = 0;
123
124    /* Check for initial call - only the menu must be printed */
125    if (u8_readCnt != 0)
126    {
127      /* Check for 'cid' command */
128      if (strcmp((void *) u8_buffer, "cid") == 0)
129      {
130        /* Read unique chip ID */
131        user_cid_get(&u32_chipId);
132        /* Send out chip ID via stdout */
133        printf("\n --> CHIP ID 0x%x\n", u32_chipId);
134      }
135      /* Otherwise check for 'ack' command */
136      else if (strcmp((void *) u8_buffer, "ack") == 0)
137      {
138        /* Send out acknowledgement via stdout */
139        printf("\n --> SEND ACKnowledgement\n");
140      }
141      /* Otherwise check for 'nak' command */
142      else if (strcmp((void *) u8_buffer, "nak") == 0)
143      {
144      }
145      /* Otherwise unknown command */
146      else
147      {
148        /* Send error message via stdout */
149        printf("\n --> ERROR: command unknown\n");
150      }
151    }
152
153    /* Print a fancy simple command menu */
154    printf("\n*** Command menu  ***\n");
155    printf(" <cid> - sends chip id\n");
156    printf(" <ack> - sends ack\n");
157    printf(" <nak> - keeps silent\n");
158    printf("Enter a command:> ");
159    /* Reset u8_readCnt for new command */
160    u8_readCnt = 0;
161    /* Deactivate command handler until new one is received */
162    b_cmdTrigger = false;
163  }
```

Figure 12 Command handler

References

See the code examples at **www.infineon.com**

Revision history

Document version	Date of release	Description of changes
1.0	2021-10-22	Initial version
1.1	2022-10-13	Editorial changes

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2022-10-13

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.