XMC4700 / 4800 USB Human Interface Device Example

Getting Started V1.0

1. Overview and Requirements
2. Setup
3. Implementing the application
4. How to test

Copyright © Infineon Technologies AG 2015. All rights reserved.

1 Overview and Requirements

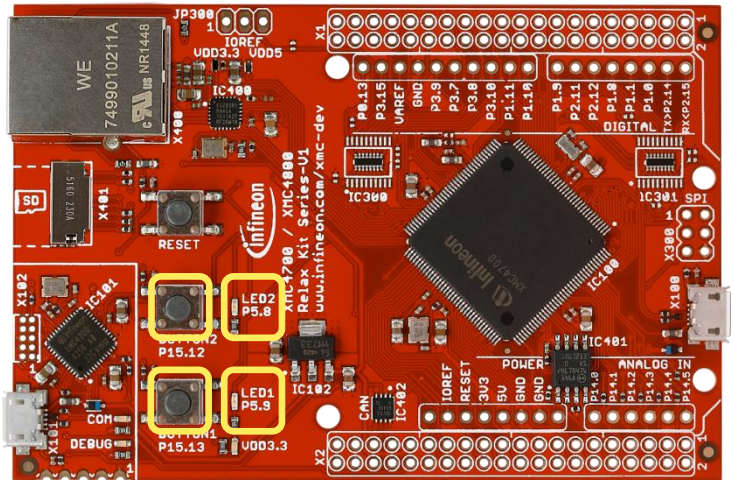
2 Setup

3 Implementing the application

4 How to test

Copyright © Infineon Technologies AG 2015. All rights reserved.

Overview



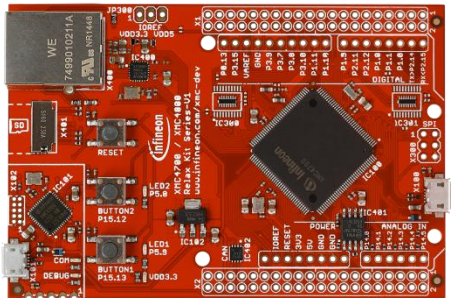
This example demonstrates the implementation of a USB Human Interface Device. It is based on the LUFA implementation which is ported to XMC4000 family and provided within this example.

Within this documentation you will be guided through all the building blocks of a typical USB HID application on the XMC4000 family. You will see the input button states of the board sent to the PC and the LEDs on the board controlled from PC.

As a result you will be enabled to implement your own USB HID application on the XMC4000 family.

Copyright © Infineon Technologies AG 2015. All rights reserved.

Requirements – hardware



XMC 4700 Relax Kit

or optional

XMC4800 Relax EtherCAT Kit



Windows Laptop installed

- DAVE v4 (Version4.1.4 or higher)
- Python™ v3.5.2 or higher
- pywinusb v0.4.1 or higher



2x Micro USB Cable (Debugger and USB HID connector)

Copyright © Infineon Technologies AG 2015. All rights reserved.

Requirements - free software downloads



DAVE™ (v4.1.4 or higher)
Link: Download DAVE (Version 4)



Python™ v3.5.2 or higher
Link: Python 3.5.2 installation

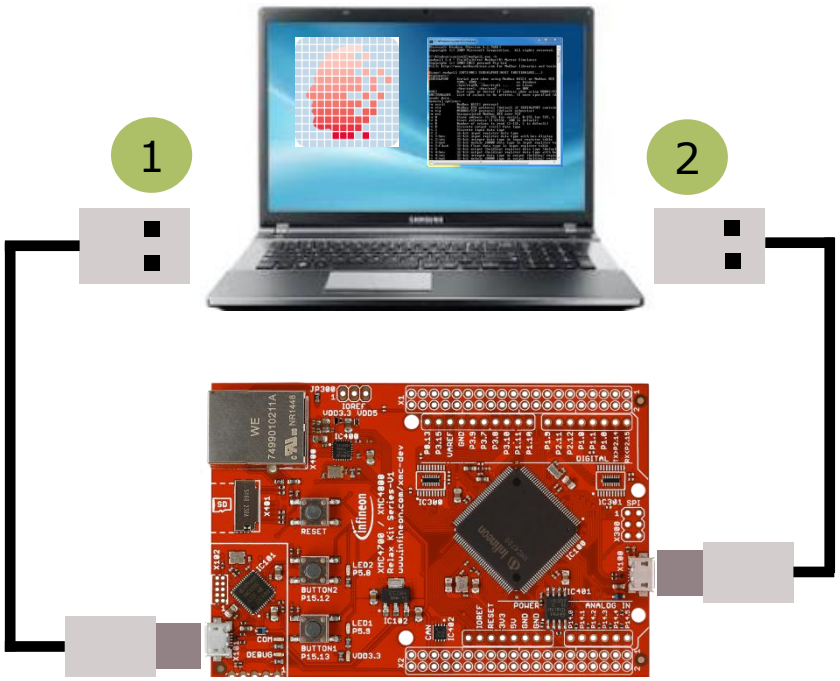


pywinusb 0.4.1 or higher:
Link: pywinusb-0.4.1.zip
Unzip and install: “setup.py install”

Copyright © Infineon Technologies AG 2015. All rights reserved.

1. Overview and Requirements
2. Setup
3. Implementing the application
4. How to test

Setup – Hardware



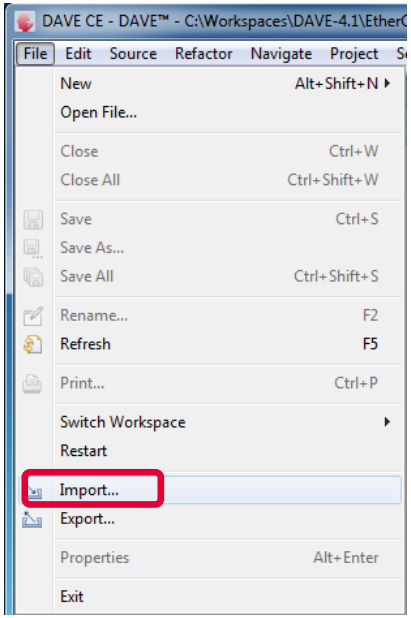
1. Micro USB cable connected to X101 connector (debugger)

2. Micro USB cable connected to X102 connector (XMC4700 USB HID)

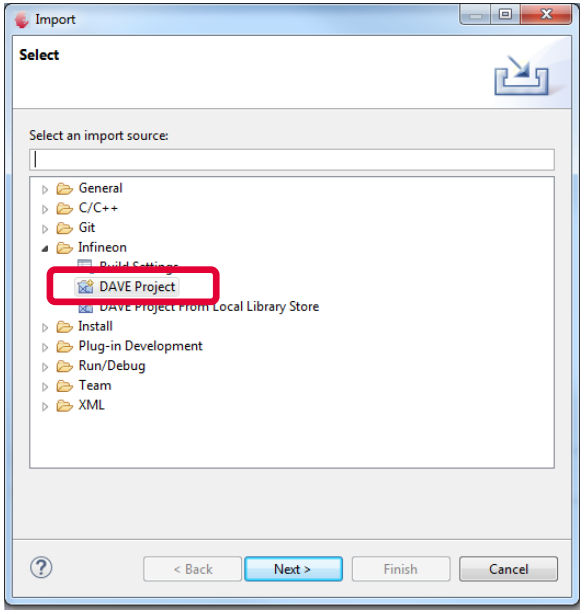
Copyright © Infineon Technologies AG 2015. All rights reserved.

Setup – Import example project into DAVE™

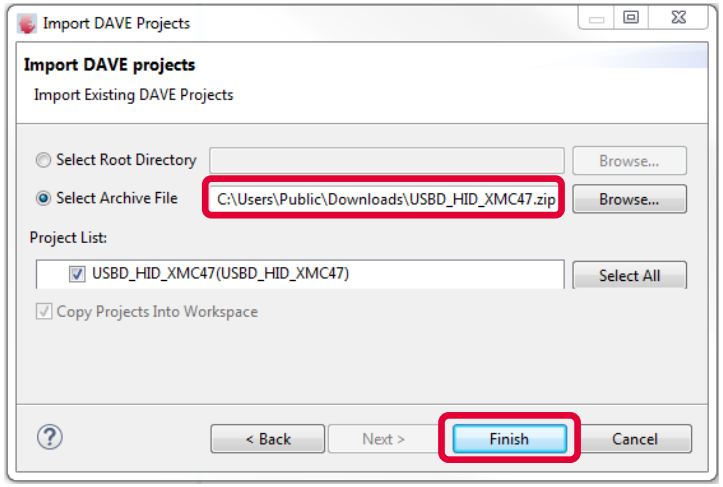
1



2

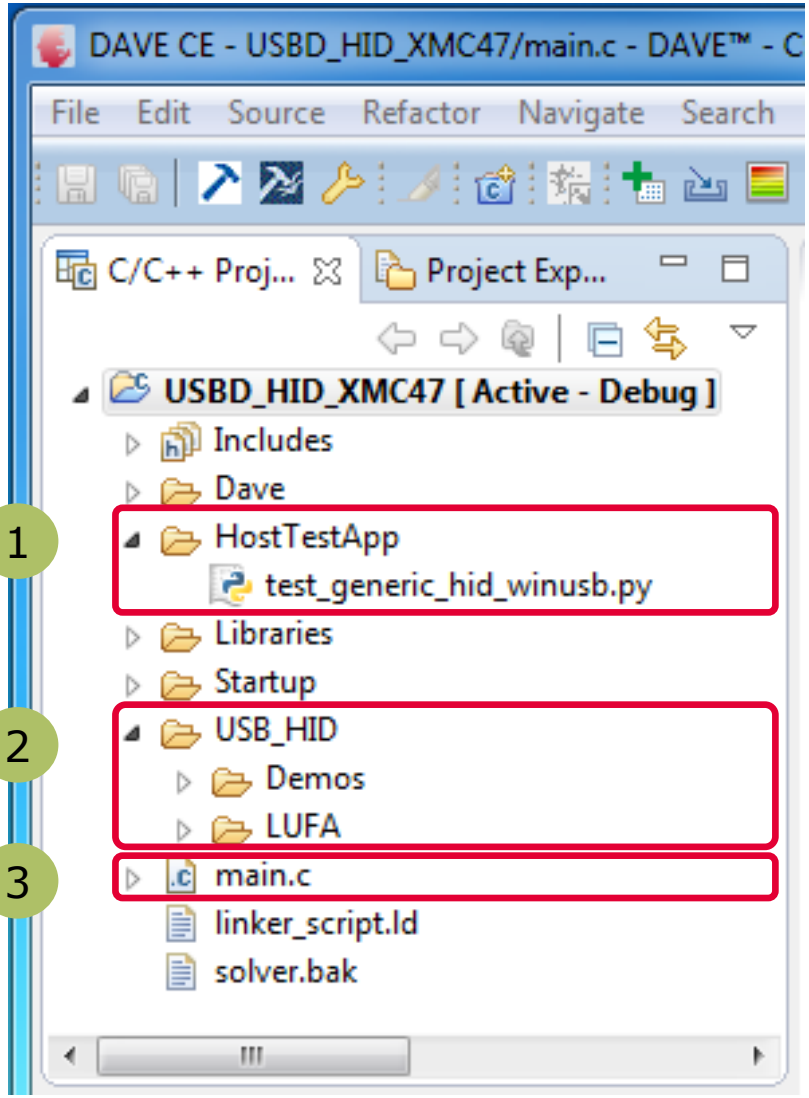


3



Copyright © Infineon Technologies AG 2015. All rights reserved.

Setup – Import example project into DAVE™



Check the folder structure of the imported project.
Beside standard DAVE project items you will find these USB HID related items:

1. Python script to access the USB HID device from PC (USB host)
2. USB HID device driver based on LUFA device class drivers
3. The main file implementing the example application

Copyright © Infineon Technologies AG 2015. All rights reserved.

1 Overview and Requirements

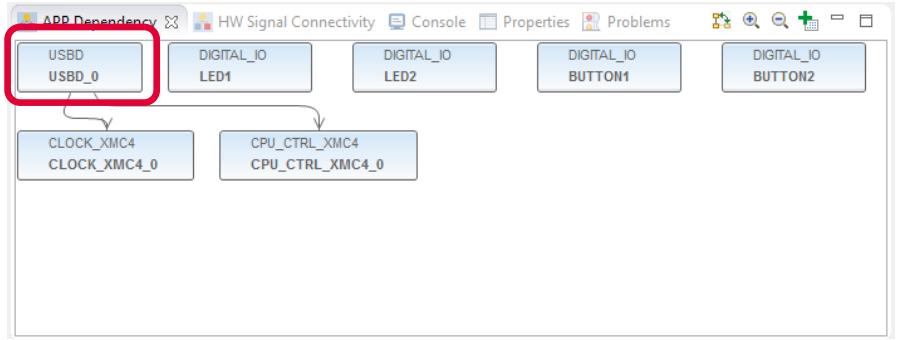
2 Setup

3 Implementing the application

4 How to test

APP structure

1



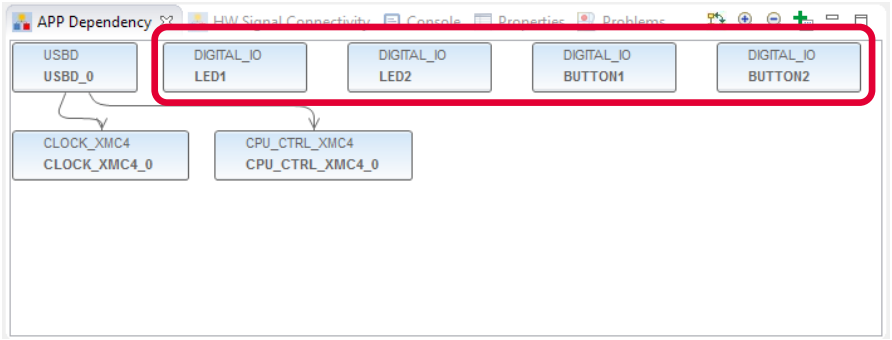
1 USBD APP is used to implement USB LUFA stack and access the USB low level drivers. CLOCK_XMC4 and CPU_CTRL_XMC4 APPs are automatically included by DAVE™ when the dependencies are resolved.

You do not need to assign pins to the USBD app, because inside hardware dedicated pins are assigned without secondary funtionality.

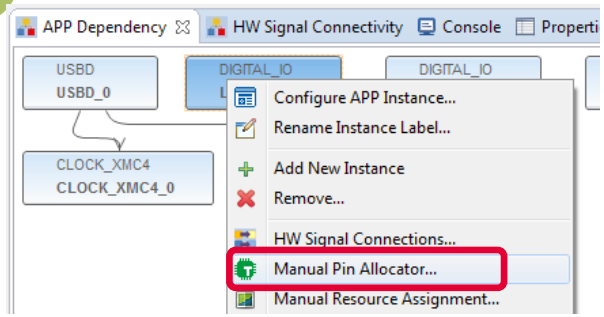
Copyright © Infineon Technologies AG 2015. All rights reserved.

APP structure

2



3



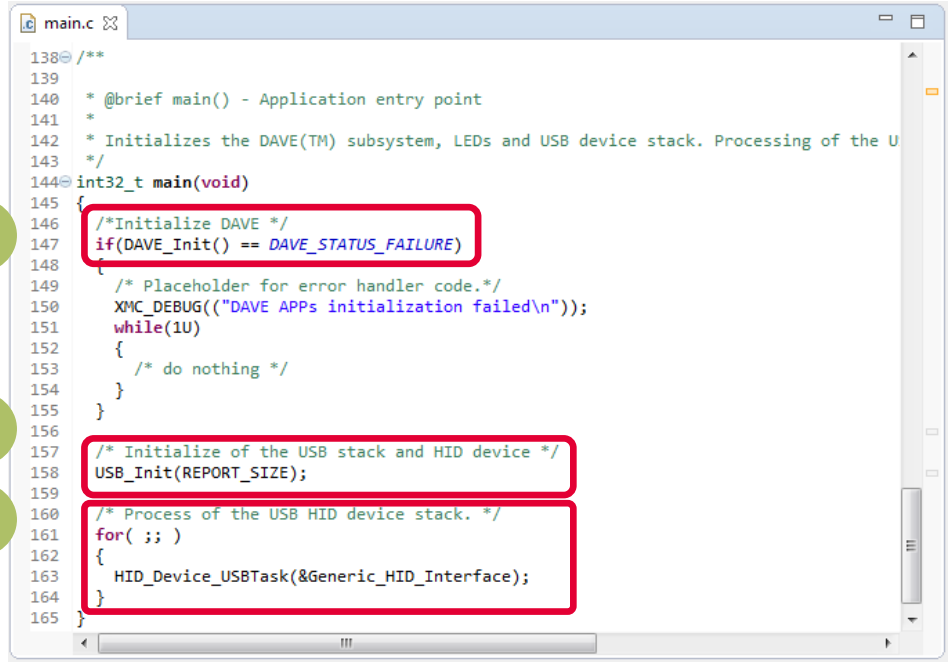
APP Instance Name	Resource	Port-Pin
BUTTON1		
	pin	(P15.13)
BUTTON2		
	pin	(P15.12)
LED1		
	pin	(P5.9)
LED2		
	pin	(P5.8)

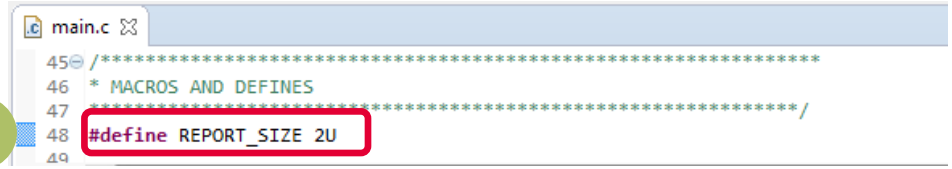
2 DIGITAL_IO APPs are used to access the input/output ports. Two ports are used to control LED1 and LED2 of XMC 4700 Relax Kit. Two ports are used as input to read the state of BUTTON1 and BUTTON2 of XMC 4700 Relax Kit.

3 Right click on each DIGITAL_IO APP. From the context menu select „Manual Pin Allocator" to open the pin allocation. For each DIGITAL_IO APP assign the correct pin of the XMC4700 Relax Kit.

Copyright © Infineon Technologies AG 2015. All rights reserved.

Application – Initialization inside main.c





1. Initialize DAVE™ system including all APPs
2. Initialize the USB stack. Define the report size (same in both directions). Maximum is 64 bytes.
3. Process USB stack
4. Define the report size specific to your application. Maximum size is 64 bytes sent/received every ms.

Copyright © Infineon Technologies AG 2015. All rights reserved.

Application – Callback for creating reports to host (device → host)

main.c

```
77  bool CALLBACK_HID_Device_CreateHIDReport(USB_ClassInfo_HID_Device_t* const HIDInterfaceInfo,
78                                           uint8_t* const ReportID,
79                                           const uint8_t ReportType,
80                                           void* ReportData,
81                                           uint16_t* const ReportSize)
82  {
83    uint8_t* Data = (uint8_t*)ReportData;
84    if (DIGITAL_IO_GetInput(&BUTTON1))
85    {
86      Data[0]=0;
87    }
88    else
89    {
90      Data[0]=1;
91    }
92    if (DIGITAL_IO_GetInput(&BUTTON2))
93    {
94      Data[1]=0;
95    }
96    else
97    {
98      Data[1]=1;
99    }
100   *ReportSize = REPORT_SIZE;
101
102   /* Return false: only send on change*/
103   return false;
104 }
105
```

1

2

1. Read button states and and store inside report.
2. Return „false“ to indicate, report should only be sent, if content has changed (button pushed or released).

Note: Callback is called every 1 ms.

Copyright © Infineon Technologies AG 2015. All rights reserved.

Application – Callback for processing reports from host (host→ device)

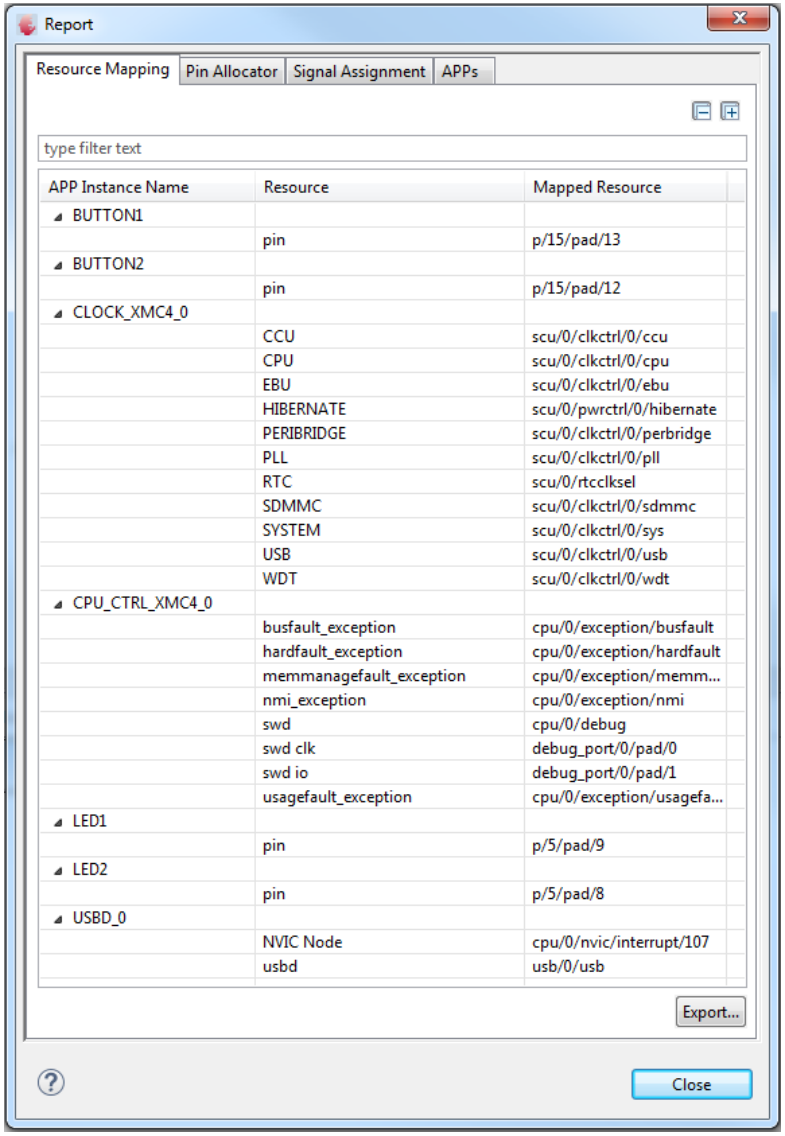
main.c

```
114 void CALLBACK_HID_Device_ProcessHIDReport(USB_ClassInfo_HID_Device_t* const HIDInterfaceInfo,
115                                           const uint8_t ReportID,
116                                           const uint8_t ReportType,
117                                           const void* ReportData,
118                                           const uint16_t ReportSize)
119 {
120   uint8_t* Data       = (uint8_t*)ReportData;
121
122   /* Check first byte of received data and set LED1 */
123   if (Data[0]==0)
124   {
125     DIGITAL_IO_SetOutputLow(&LED1);
126   }
127   else
128   {
129     DIGITAL_IO_SetOutputHigh(&LED1);
130   }
131
132   /* Check second byte of received data and set LED2 */
133   if (Data[1]==0)
134   {
135     DIGITAL_IO_SetOutputLow(&LED2);
136   }
137   else
138   {
139     DIGITAL_IO_SetOutputHigh(&LED2);
140   }
141 }
```

Check report and set LED state

Copyright © Infineon Technologies AG 2015. All rights reserved.

Application – Overview on used HW resources



APP Instance Name	Resource	Mapped Resource
BUTTON1		
	pin	p/15/pad/13
BUTTON2		
	pin	p/15/pad/12
CLOCK_XMC4_0		
	CCU	scu/0/clkctrl/0/ccu
	CPU	scu/0/clkctrl/0/cpu
	EBU	scu/0/clkctrl/0/ebu
	HIBERNATE	scu/0/pwrctrl/0/hibernate
	PERIBRIDGE	scu/0/clkctrl/0/perbridge
	PLL	scu/0/clkctrl/0/pll
	RTC	scu/0/rtcclksel
	SDMMC	scu/0/clkctrl/0/sdmmc
	SYSTEM	scu/0/clkctrl/0/sys
	USB	scu/0/clkctrl/0/usb
	WDT	scu/0/clkctrl/0/wdt
CPU_CTRL_XMC4_0		
	busfault_exception	cpu/0/exception/busfault
	hardfault_exception	cpu/0/exception/hardfault
	memmanagefault_exception	cpu/0/exception/memm...
	nmi_exception	cpu/0/exception/nmi
	swd	cpu/0/debug
	swd clk	debug_port/0/pad/0
	swd io	debug_port/0/pad/1
	usagefault_exception	cpu/0/exception/usagefa...
LED1		
	pin	p/5/pad/9
LED2		
	pin	p/5/pad/8
USBD_0		
	NVIC Node	cpu/0/nvic/interrupt/107
	usbd	usb/0/usb

Copyright © Infineon Technologies AG 2015. All rights reserved.

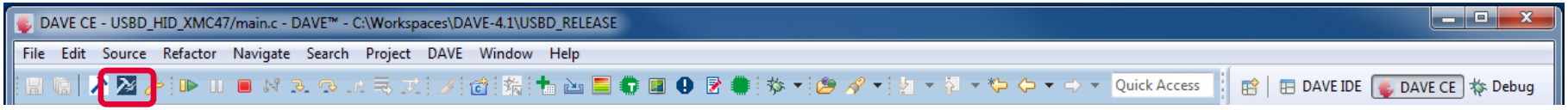
1. Overview and Requirements
2. Setup
3. Implementing the application
4. How to test

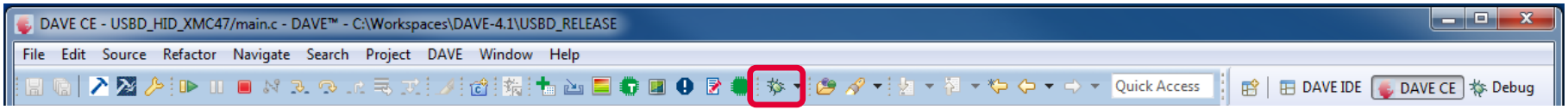
How to test



ACTIONS

1. Build and download the example application software to the XMC4700 / XMC4800 Relax Kit and start the debugger

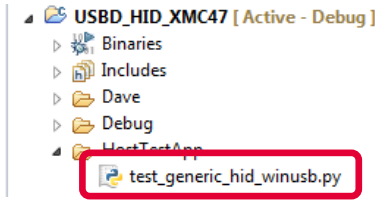




2. Start the software by the run button



3. Start test host python script on PC by double click

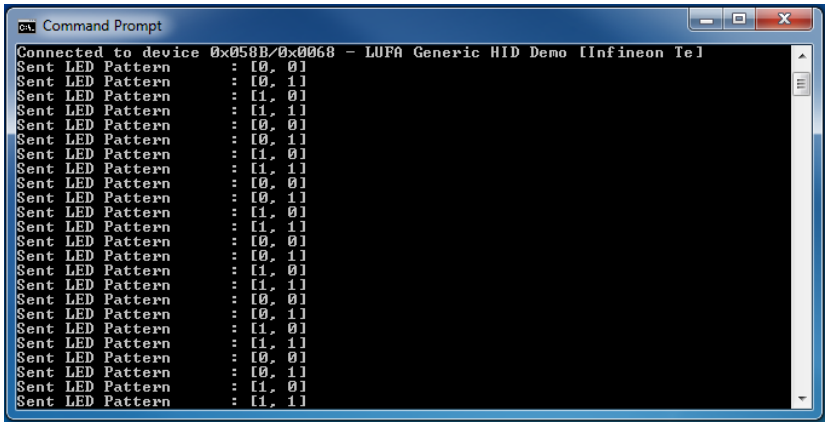


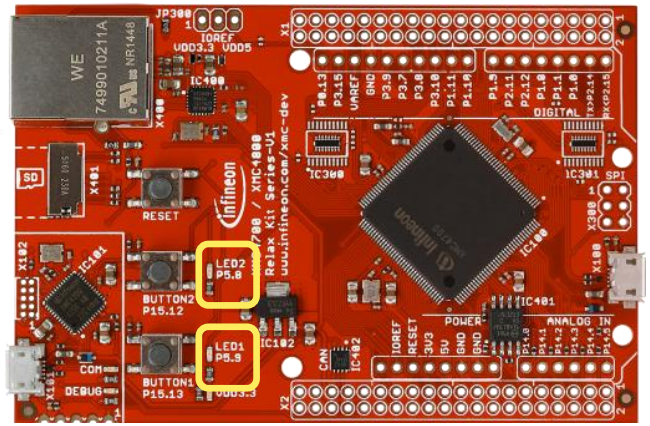
Copyright © Infineon Technologies AG 2015. All rights reserved.

How to test

OBSERVATIONS

The python script sends a HID report to the device every second. You will see the LEDs toggling simultaneously when a new LED pattern is sent.





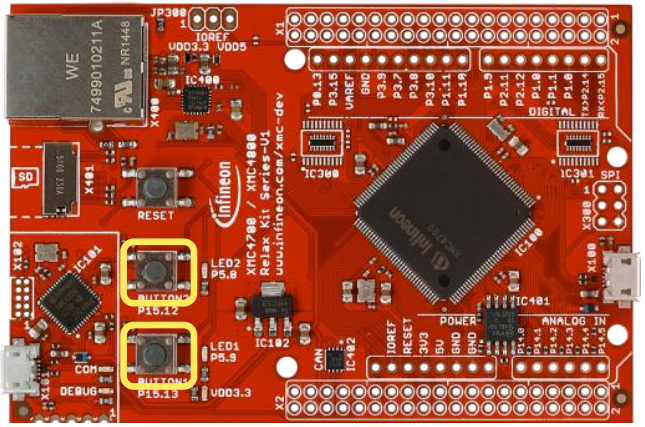
Copyright © Infineon Technologies AG 2015. All rights reserved.

How to test



ACTIONS

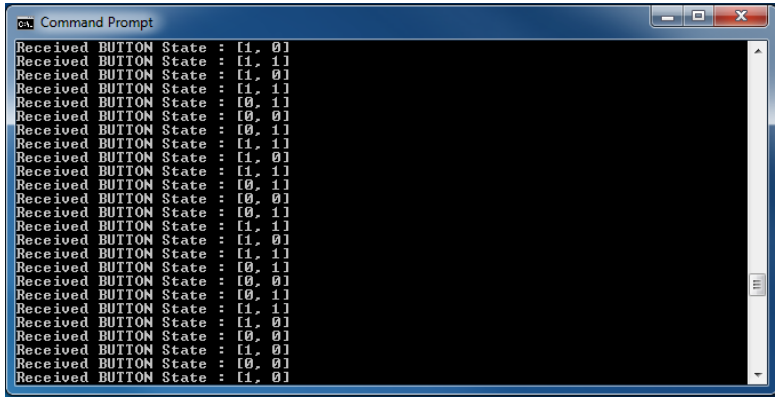
1. Push BUTTON1 and/or BUTTON2 on the XMC4700 Relax Kit.





OBSERVATIONS

The python script receives a HID report every time the button state changes.



Copyright © Infineon Technologies AG 2015. All rights reserved.



Part of your life. Part of tomorrow.

Infineon