XMC4500
USB Host
Mass Storage Example

Getting Started V1.0

Contents

1. Overview
2. Requirements
3. Setup
4. Implementing the application
5. How to test
6. Debug hints
7. Further reading
8. Limitations

Contents

1 Overview

2 Requirements

3 Setup

4 Implementing the application

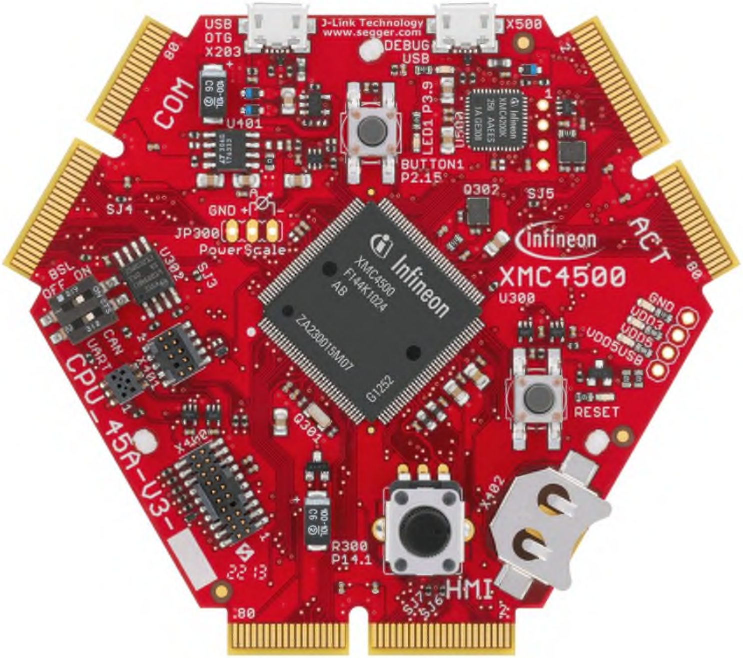
5 How to test

6 Debug hints

7 Further reading

8 Limitations

Overview



- This example demonstrates the implementation of USB host mass storage functionality. It is based on the free LUFA USB host stack implementation which is ported to XMC4500 family and provided within this example.

- Within this documentation you will be guided through all the building blocks of a typical USB host mass storage application on the XMC4500 family. You will be able to connect a USB flash drive device supporting the FAT file system to the host and perform file read and write access to the device.

- As a result you will be enabled to implement your own USB host mass storage functionality on the XMC4500 family.

Contents

1 Overview

2 Requirements

3 Setup

4 Implementing the application

5 How to test

6 Debug hints

7 Further reading

8 Limitations

Requirements - hardware

XMC4500 General Purpose board CPU_45A-V3

Micro-A plug
Std-A receptacle

USB cable micro-A plug to standard-A receptacle for host example

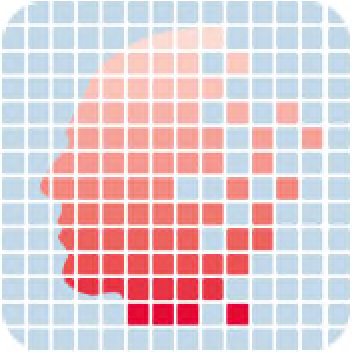
Micro-B plug
Std-A plug

USB cable micro-B plug to standard-A plug for debugging

USB flash drive

Requirements – hardware & software

- Windows laptop
- DAVE™ installed



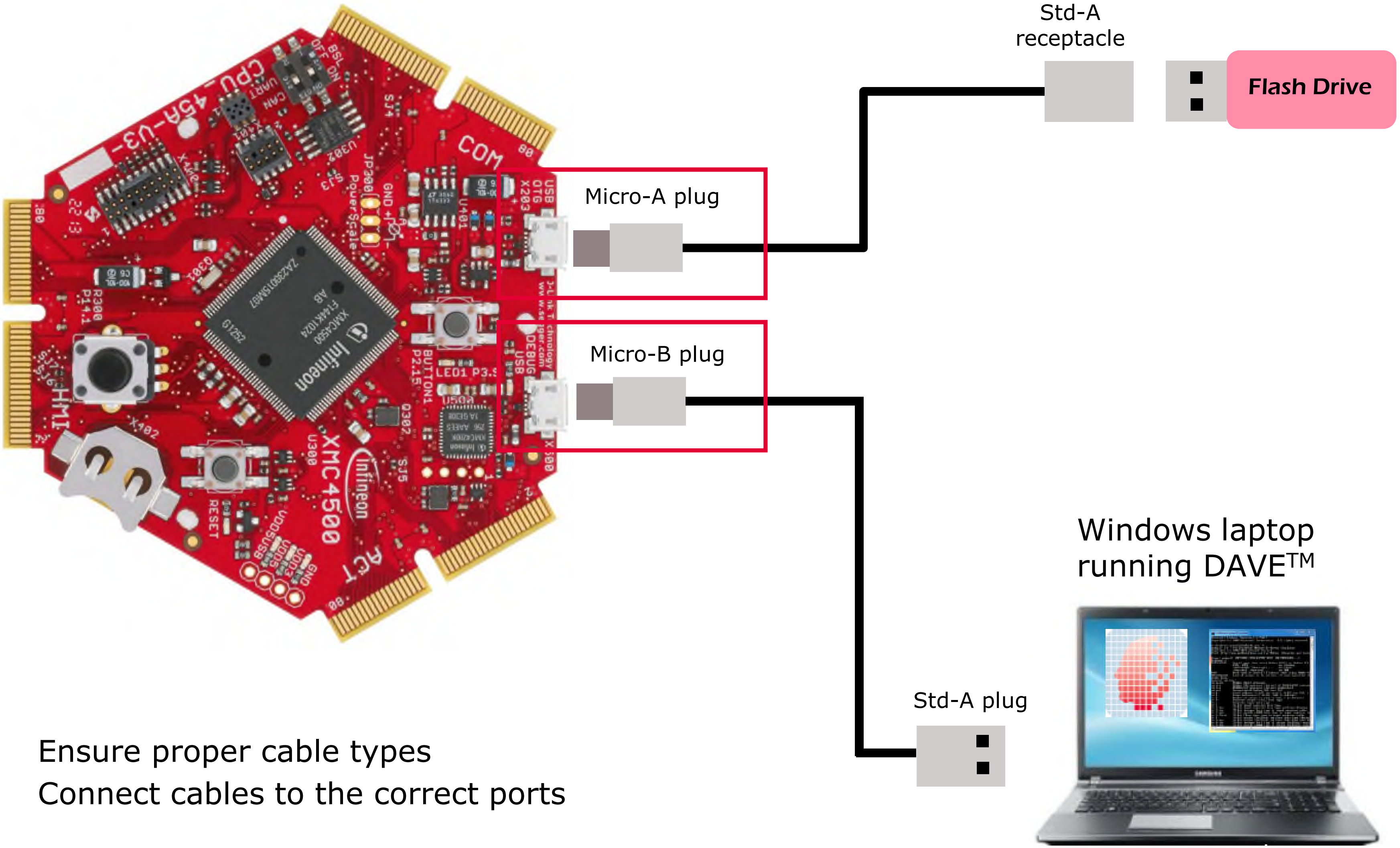
- DAVE™ (v4.1.4 or Higher)
- Download DAVE™ free of charge

  Link : [DAVE™ v4 Download]

Contents

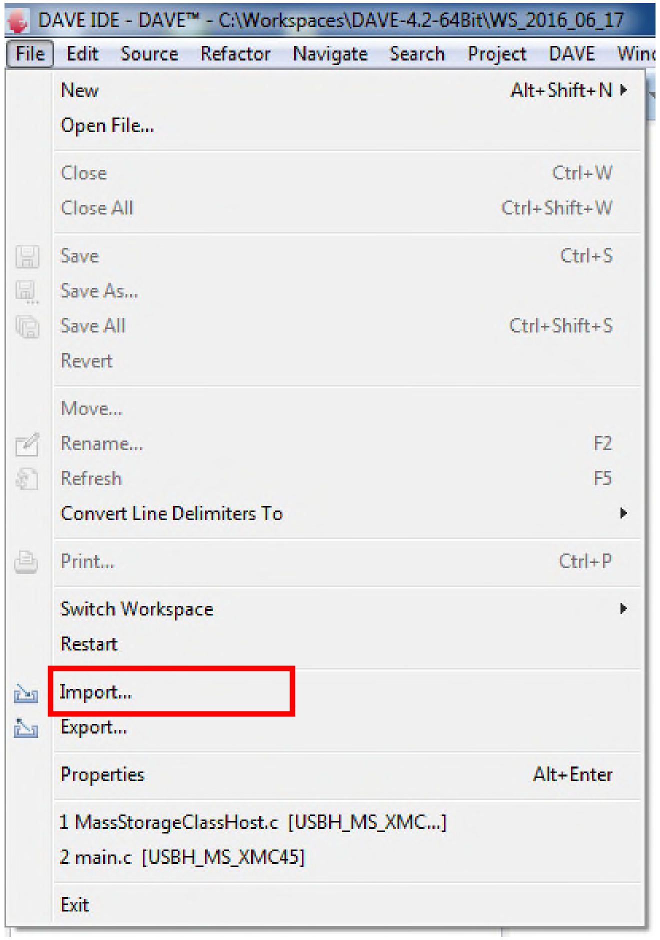
1. Overview
2. Requirements
3. Setup
4. Implementing the application
5. How to test
6. Debug hints
7. Further reading
8. Limitations

Setup - hardware



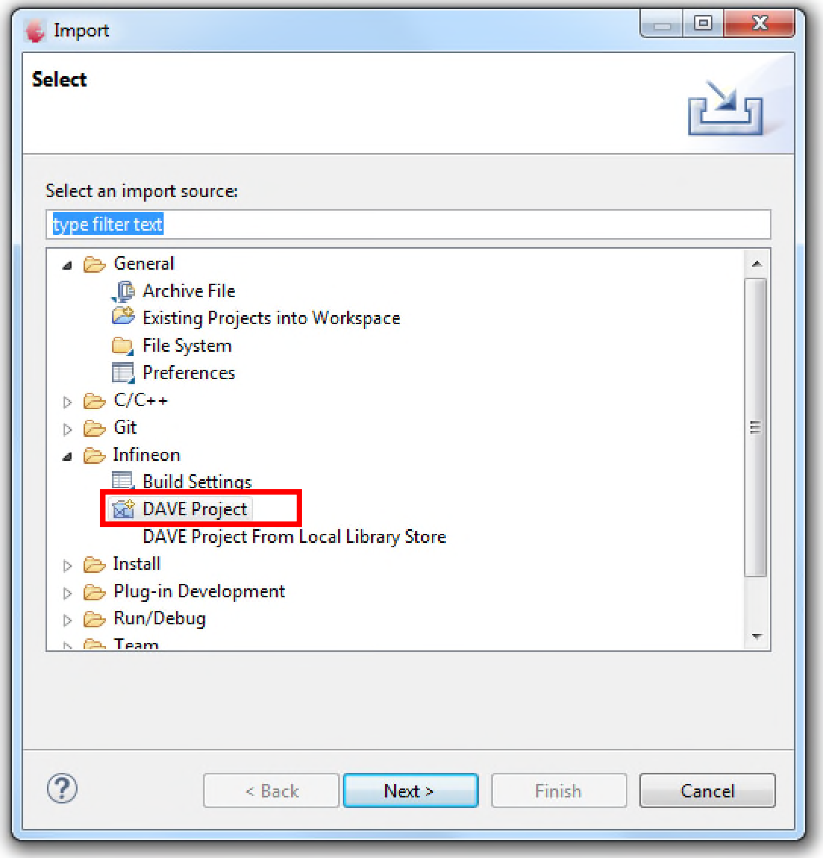
- Ensure proper cable types
- Connect cables to the correct ports

Setup – import example project in to DAVE^{TM}


DAVE IDE - DAVE™ - C:\Workspaces\DAVE-4.2-64Bit\WS_2016_06_17

1

Setup – import example project in to DAVE^{TM}



Import

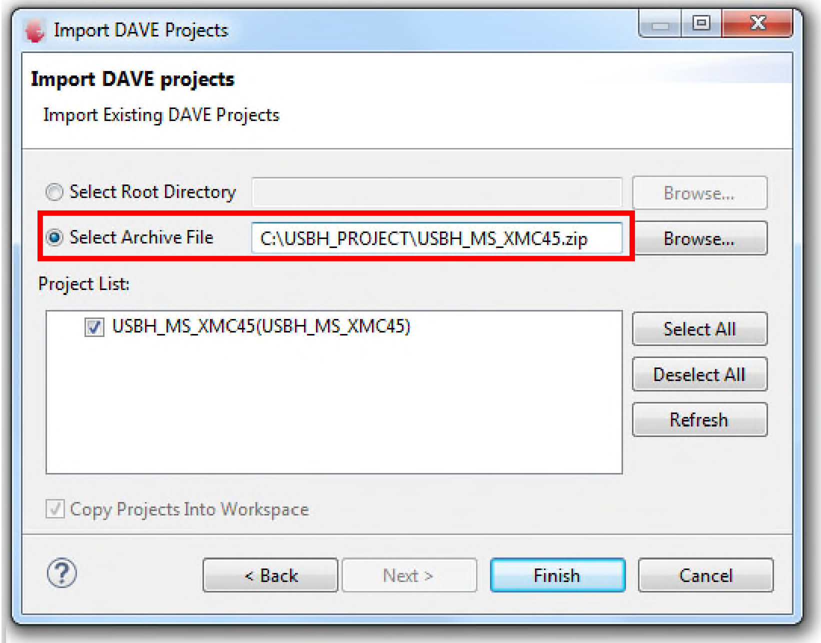
Select

Select an import source:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- C/C++
- Git
- Infineon
  - Build Settings
  - DAVE Project
  - DAVE Project From Local Library Store
- Install
- Plug-in Development
- Run/Debug
- Team

2



Import DAVE Projects

Import DAVE projects

Import Existing DAVE Projects

○ Select Root Directory

● Select Archive File C:\USBH_PROJECT\USBH_MS_XMC45.zip

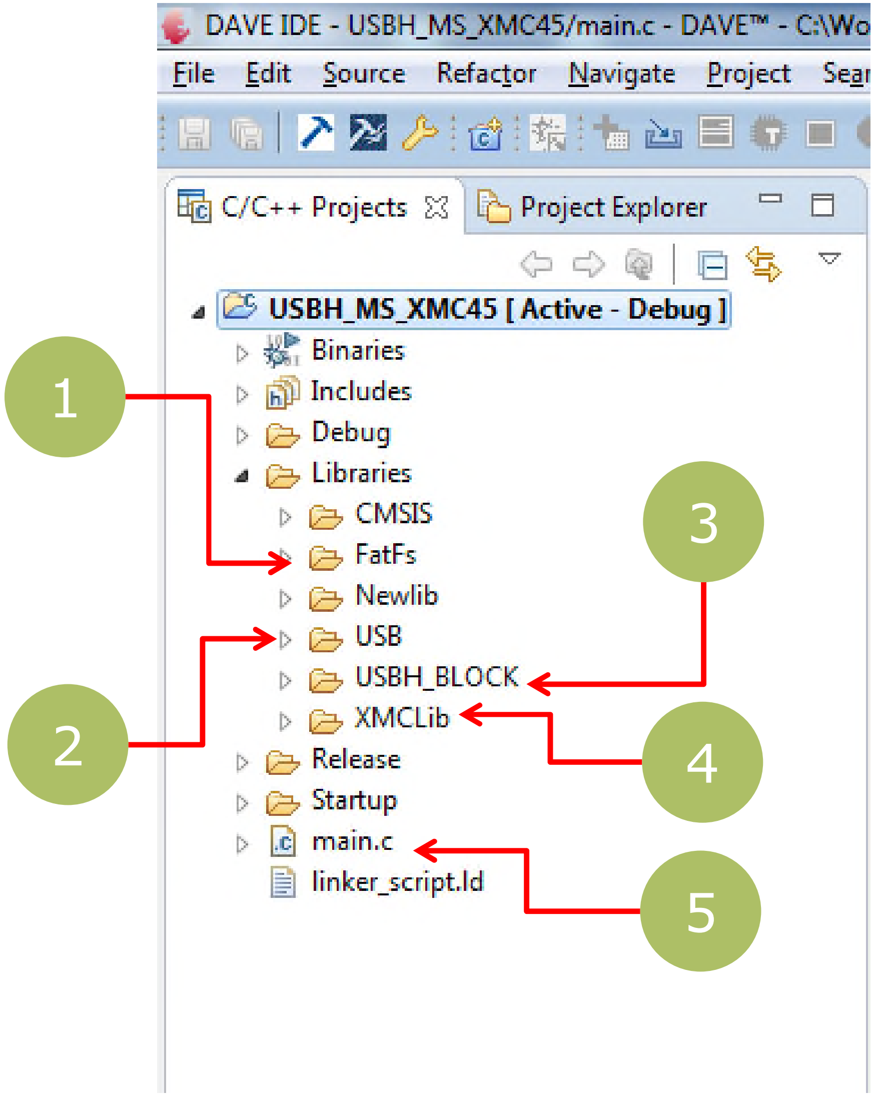
Project List:

☑ USBH_MS_XMC45(USBH_MS_XMC45)

☑ Copy Projects Into Workspace

3

Setup – import example project in to DAVE^{TM}



- Check the folder structure of the imported project.

1 Free FAT file system code

2 Free LUFA USB stack, mass storage class and glue layer for XMC low level driver

3 Code linking FAT file system and USB mass storage layer

4 XMCLib folder contains USB low level driver in file xmc_usb.c

5 The main file implementing the application

Contents

1 Overview

2 Requirements

3 Setup

4 Implementing the application

5 How to test

6 Debug hints

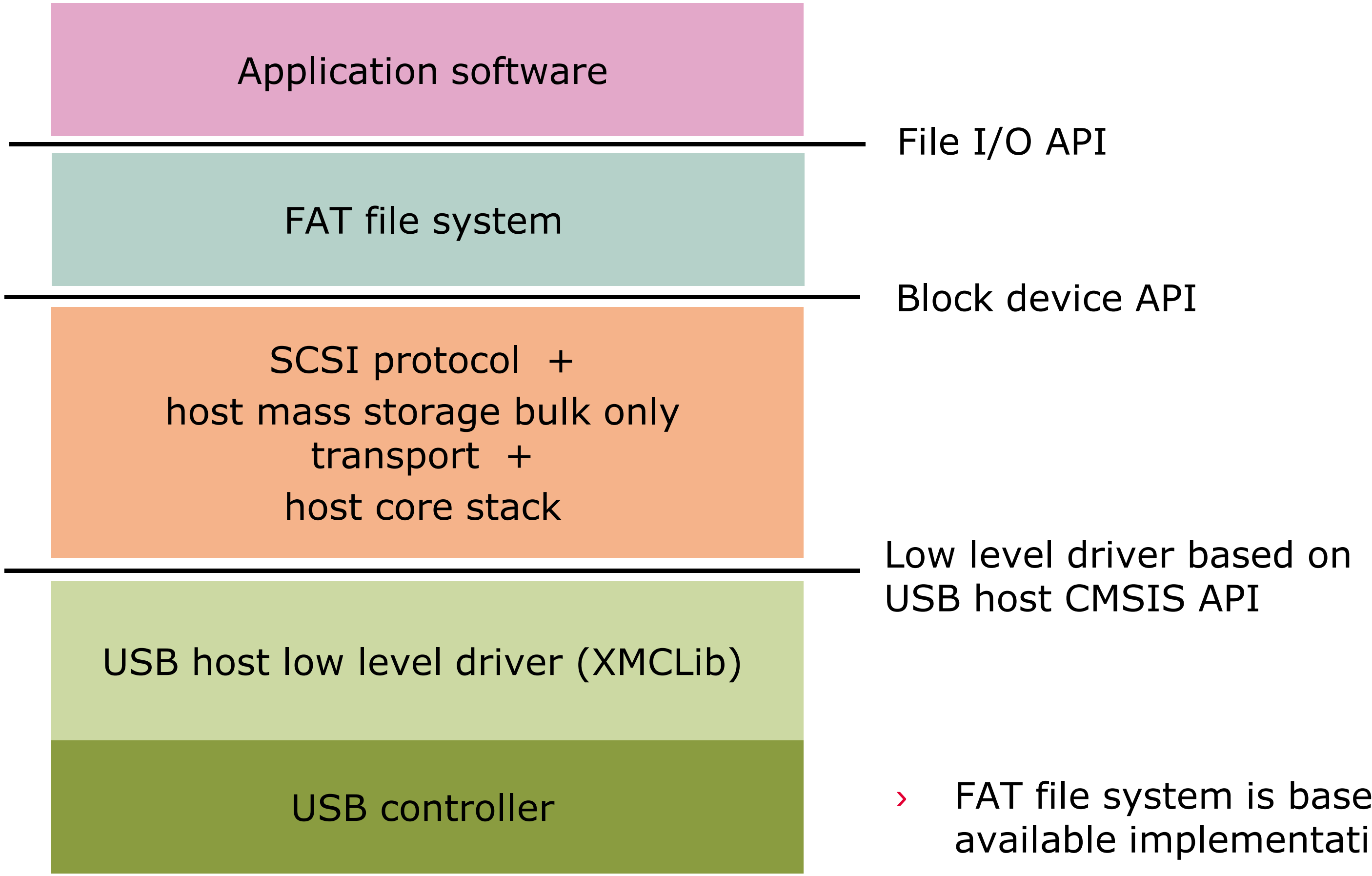
7 Further reading

8 Limitations

Example application behavior

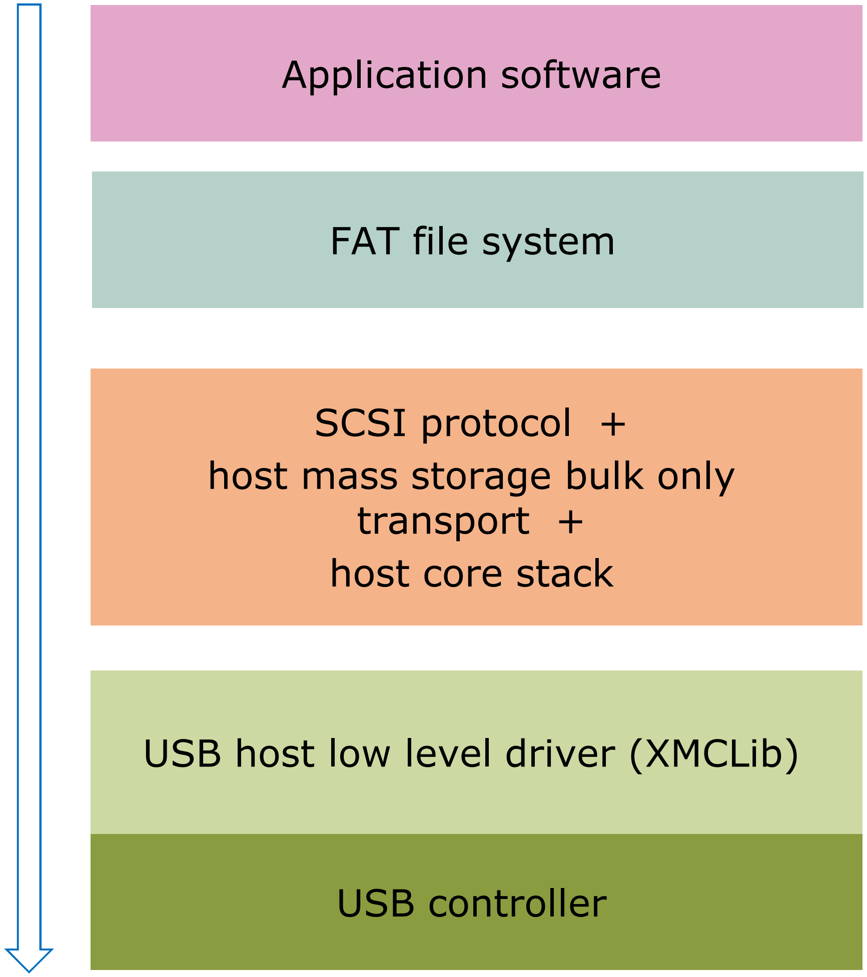
- XMC4500 General Purpose board is initialized as a USB host and provides VBUS on the USB port
- When a USB flash drive is connected, LED1 (on P3.9) is turned ON after device is successfully enumerated
- Host application creates a folder with the name ABSTEP and it creates 3 files, of sizes 1 KB, 2KB and 3KB using File IO calls.
- The files are read back and compared. LED blinks continuously if comparison is successful, else it stays ON continuously
- When USB flash drive is disconnected, LED1 is turned off

Simplified example application architecture



- FAT file system is based on a freely available implementation
- USB stack and mass storage class implementation is based on a freely available stack known as LUFA

Example application data flow



Application software

FAT file system

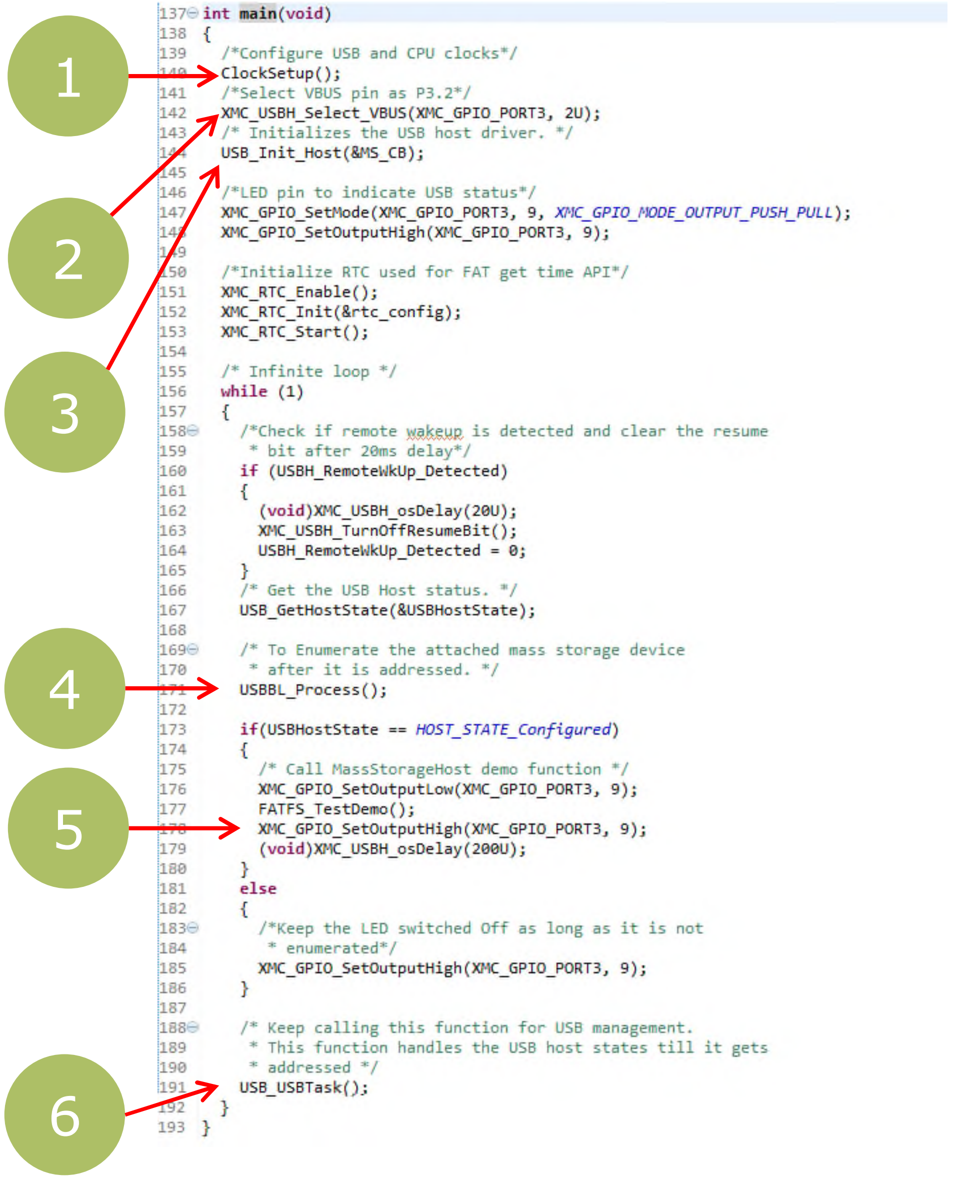
SCSI protocol +
host mass storage bulk only transport +
host core stack

USB host low level driver (XMCLib)

USB controller

- Application software launches a file IO request using FAT file IO API
- FAT file system converts the file IO calls in to block device calls.
- Blocks are read and written using the standard SCSI commands and the mass storage bulk only transport protocol state machine
- Data is transferred to/from USB mass storage device using low level driver APIs

Application – overview of main.c



```
int main(void)
{
  /*Configure USB and CPU clocks*/
  ClockSetup();
  /*Select VBUS pin as P3.2*/
  XMC_USBH_Select_VBUS(XMC_GPIO_PORT3, 2U);
  /* Initializes the USB host driver. */
  USB_Init_Host(&MS_CB);

  /*LED pin to indicate USB status*/
  XMC_GPIO_SetMode(XMC_GPIO_PORT3, 9, XMC_GPIO_MODE_OUTPUT_PUSH_PULL);
  XMC_GPIO_SetOutputHigh(XMC_GPIO_PORT3, 9);

  /*Initialize RTC used for FAT get time API*/
  XMC_RTC_Enable();
  XMC_RTC_Init(&rtc_config);
  XMC_RTC_Start();

  /* Infinite loop */
  while (1)
  {
    /*Check if remote wakeup is detected and clear the resume
     * bit after 20ms delay*/
    if (USBH_RemoteWkUp_Detected)
    {
      (void)XMC_USBH_osDelay(20U);
      XMC_USBH_TurnOffResumeBit();
      USBH_RemoteWkUp_Detected = 0;
    }
    /* Get the USB Host status. */
    USB_GetHostState(&USBHostState);

    /* To Enumerate the attached mass storage device
     * after it is addressed. */
    USBBL_Process();

    if(USBHostState == HOST_STATE_Configured)
    {
      /* Call MassStorageHost demo function */
      XMC_GPIO_SetOutputLow(XMC_GPIO_PORT3, 9);
      FATFS_TestDemo();
      XMC_GPIO_SetOutputHigh(XMC_GPIO_PORT3, 9);
      (void)XMC_USBH_osDelay(200U);
    }
    else
    {
      /*Keep the LED switched Off as long as it is not
       * enumerated*/
      XMC_GPIO_SetOutputHigh(XMC_GPIO_PORT3, 9);
    }

    /* Keep calling this function for USB management.
     * This function handles the USB host states till it gets
     * addressed */
    USB_USBTask();
  }
}
```

1 Clock setup for XMC4500

2 Select VBUS pin P3.2

3 Initialize USB host

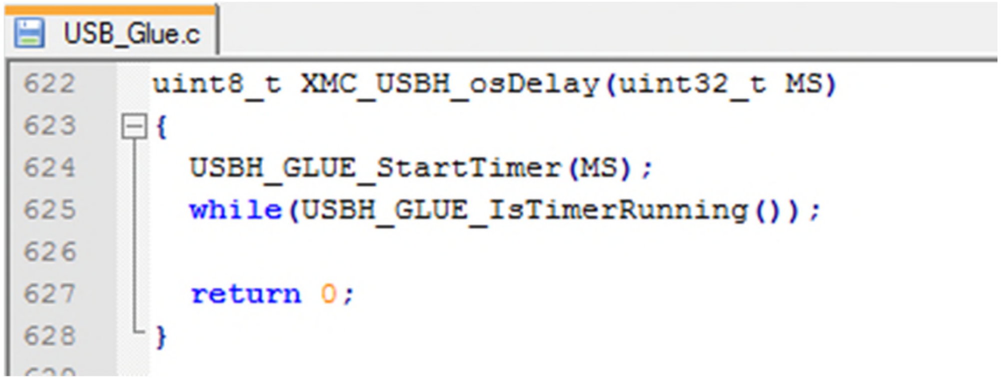
4 Fetch descriptors and match it to mass storage device

5 Demo application function

6 Manage connection events and initiate enumeration

Application – configure time delay API

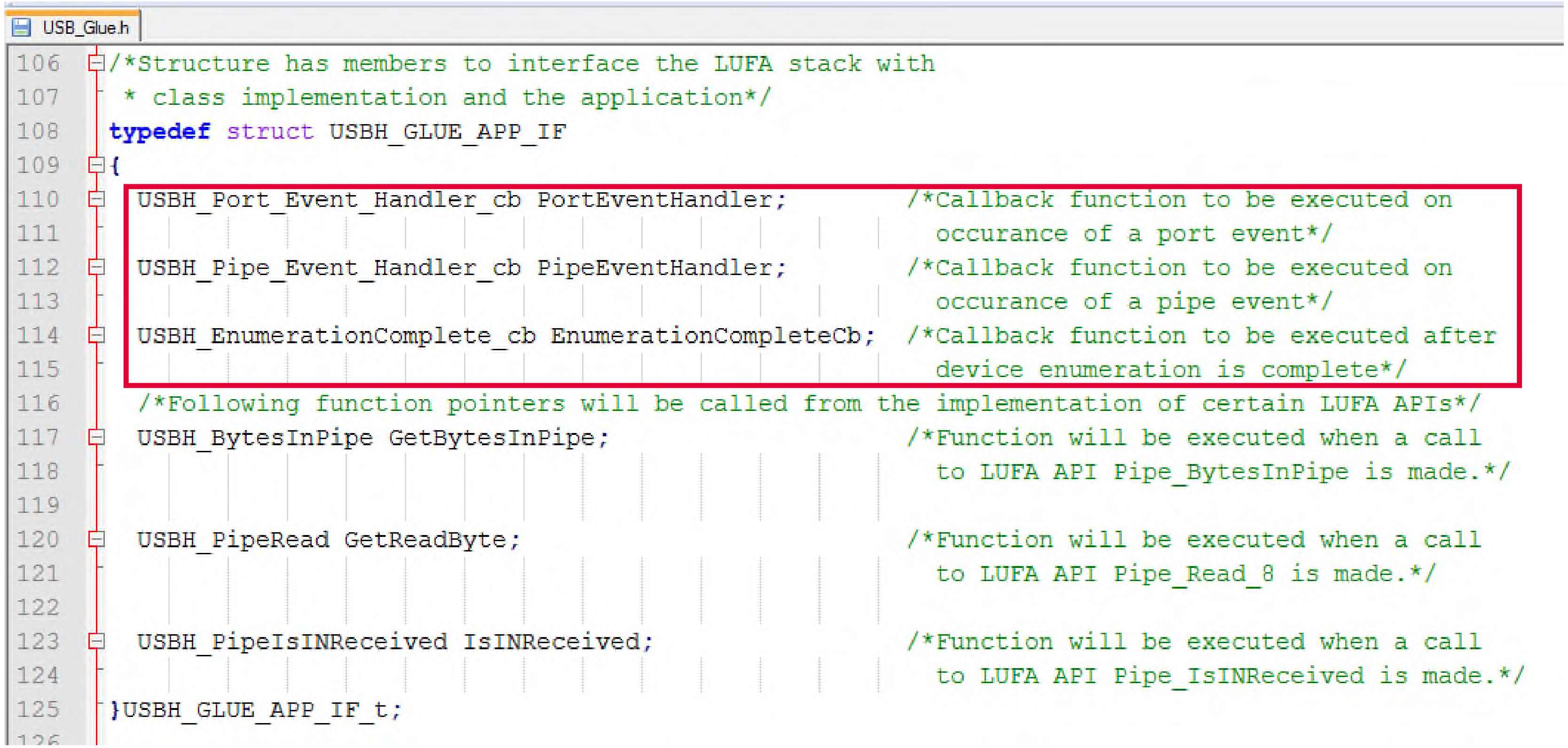
- The USB stack uses timing delay of the order of several tens of milliseconds to comply with the specification
- Port the time delay API <u>if required</u>. In the example code, the SysTick timer is used.
- Ensure timer is calibrated



```
uint8_t XMC_USBH_osDelay(uint32_t MS)
{
  USBH_GLUE_StartTimer(MS);
  while(USBH_GLUE_IsTimerRunning());

  return 0;
}
```

Application - callbacks

› Application registers for the following event callbacks

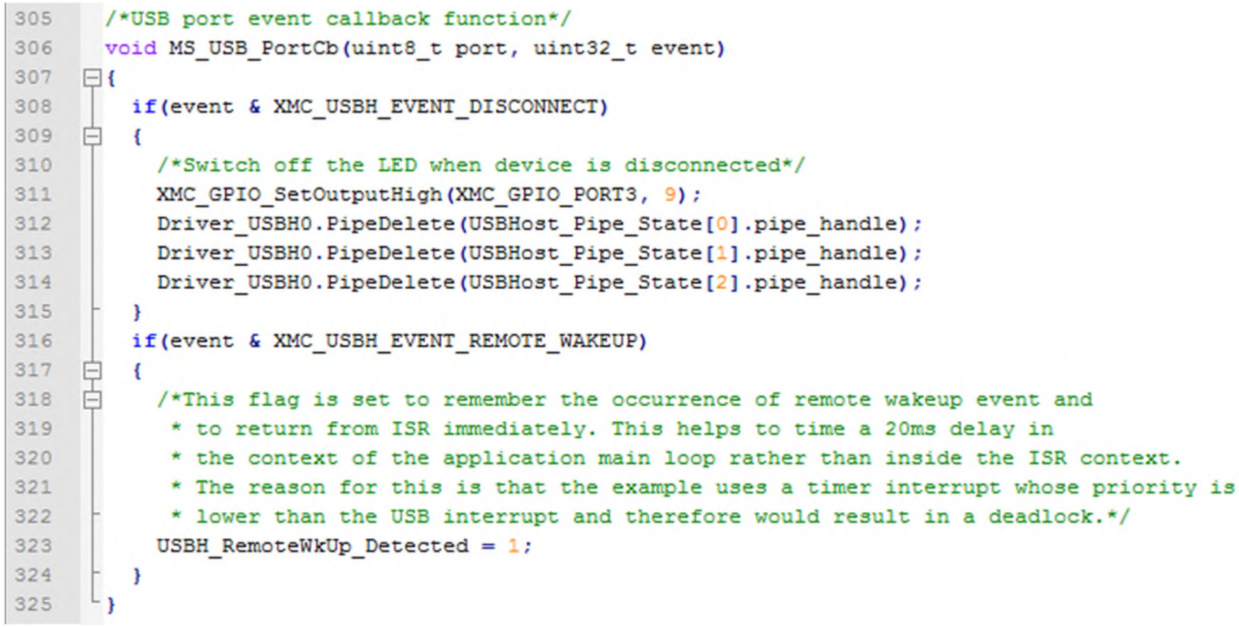


```c
/*Structure has members to interface the LUFA stack with
 * class implementation and the application*/
typedef struct USBH_GLUE_APP_IF
{
  USBH_Port_Event_Handler_cb PortEventHandler;          /*Callback function to be executed on
                                                          occurance of a port event*/
  USBH_Pipe_Event_Handler_cb PipeEventHandler;          /*Callback function to be executed on
                                                          occurance of a pipe event*/
  USBH_EnumerationComplete_cb EnumerationCompleteCb;  /*Callback function to be executed after
                                                          device enumeration is complete*/
  /*Following function pointers will be called from the implementation of certain LUFA APIs*/
  USBH_BytesInPipe GetBytesInPipe;                      /*Function will be executed when a call
                                                          to LUFA API Pipe_BytesInPipe is made.*/

  USBH_PipeRead GetReadByte;                            /*Function will be executed when a call
                                                          to LUFA API Pipe_Read_8 is made.*/

  USBH_PipeIsINReceived IsINReceived;                   /*Function will be executed when a call
                                                          to LUFA API Pipe_IsINReceived is made.*/
}USBH_GLUE_APP_IF_t;
```

Application – Implement and register required callbacks

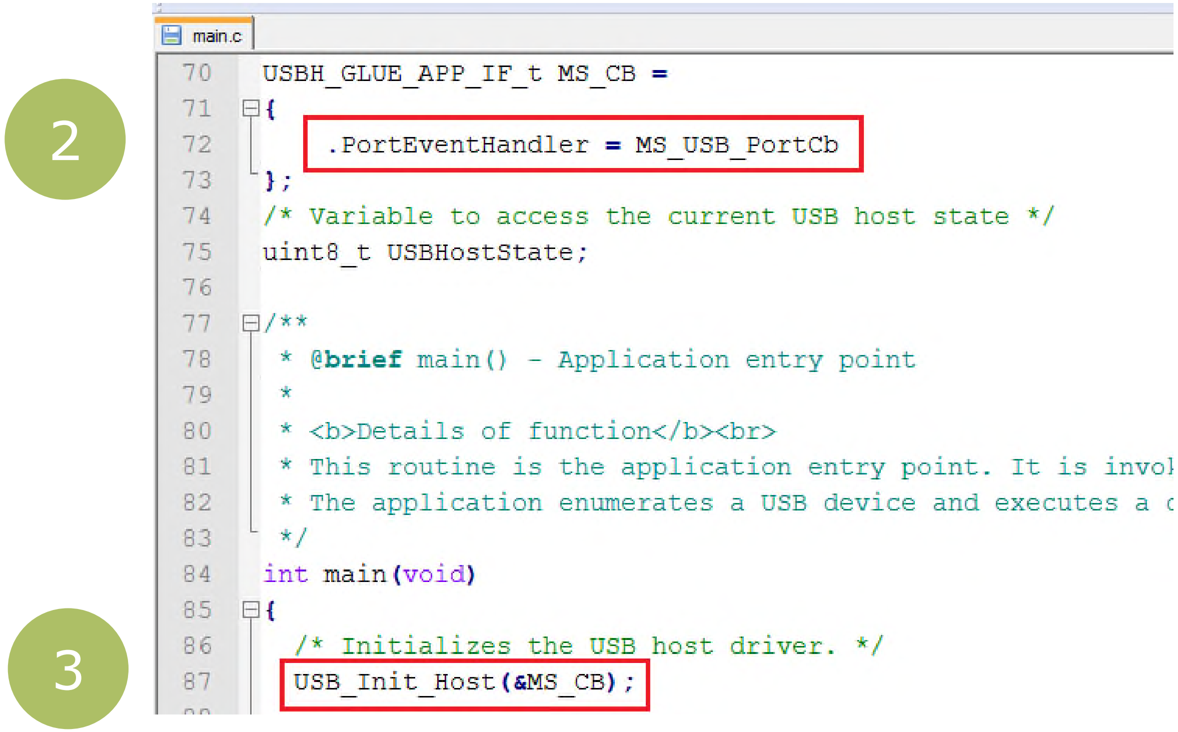
1 Implement call back function

1



```
/*USB port event callback function*/
void MS_USB_PortCb(uint8_t port, uint32_t event)
{
  if(event & XMC_USBH_EVENT_DISCONNECT)
  {
    /*Switch off the LED when device is disconnected*/
    XMC_GPIO_SetOutputHigh(XMC_GPIO_PORT3, 9);
    Driver_USBH0.PipeDelete(USBHost_Pipe_State[0].pipe_handle);
    Driver_USBH0.PipeDelete(USBHost_Pipe_State[1].pipe_handle);
    Driver_USBH0.PipeDelete(USBHost_Pipe_State[2].pipe_handle);
  }
  if(event & XMC_USBH_EVENT_REMOTE_WAKEUP)
  {
    /*This flag is set to remember the occurrence of remote wakeup event and
     * to return from ISR immediately. This helps to time a 20ms delay in
     * the context of the application main loop rather than inside the ISR context.
     * The reason for this is that the example uses a timer interrupt whose priority is
     * lower than the USB interrupt and therefore would result in a deadlock.*/
    USBH_RemoteWkUp_Detected = 1;
  }
}
```

Application – Implement and register required callbacks



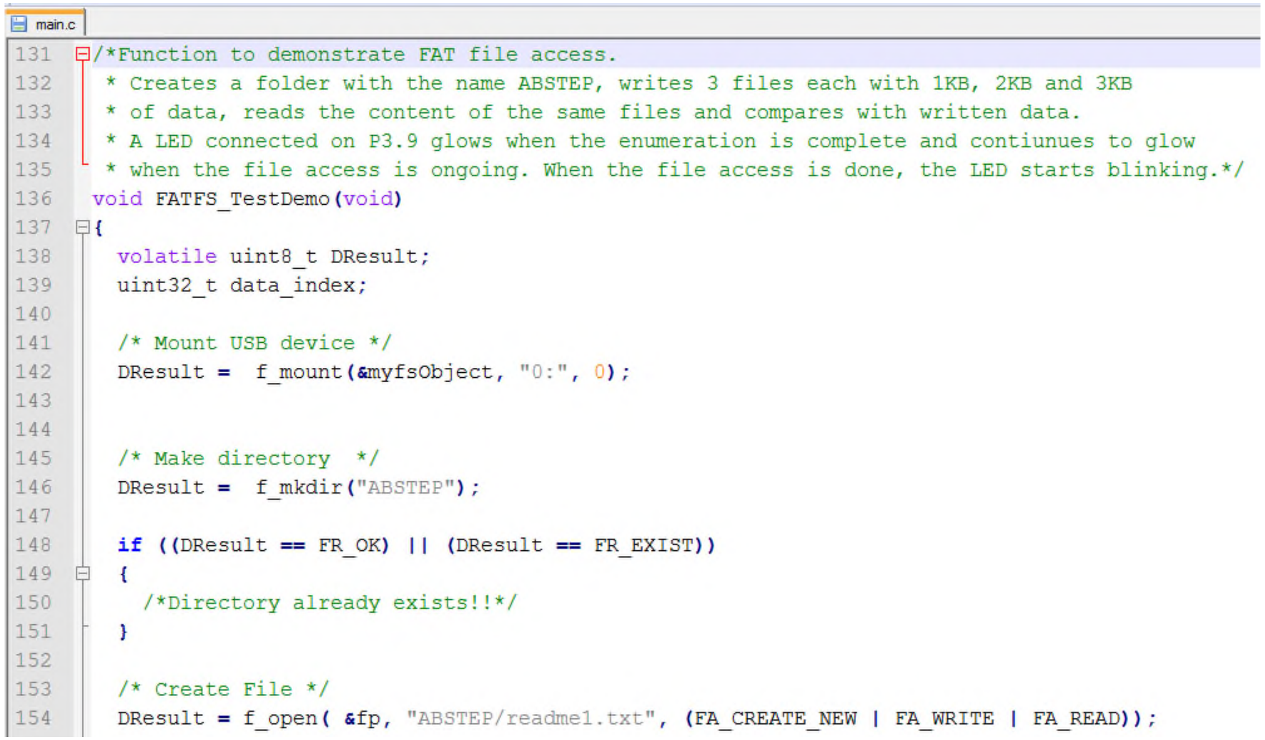
```
USBH_GLUE_APP_IF_t MS_CB =
{
    .PortEventHandler = MS_USB_PortCb
};
/* Variable to access the current USB host state */
uint8_t USBHostState;

/**
 * @brief main() - Application entry point
 *
 * <b>Details of function</b><br>
 * This routine is the application entry point. It is invo
 * The application enumerates a USB device and executes a
 */
int main(void)
{
  /* Initializes the USB host driver. */
 USB_Init_Host(&MS_CB);
```

2 Register call back function

3 Call initialization function

Application – file transfer demo

- File I/O APIs are called to transfer files between host and device
- Function FATFS_TestDemo() demonstrates file accesses



```
/*Function to demonstrate FAT file access.
 * Creates a folder with the name ABSTEP, writes 3 files each with 1KB, 2KB and 3KB
 * of data, reads the content of the same files and compares with written data.
 * A LED connected on P3.9 glows when the enumeration is complete and contiunues to glow
 * when the file access is ongoing. When the file access is done, the LED starts blinking.*/
void FATFS_TestDemo(void)
{
  volatile uint8_t DResult;
  uint32_t data_index;

  /* Mount USB device */
  DResult =  f_mount(&myfsObject, "0:", 0);


  /* Make directory  */
  DResult =  f_mkdir("ABSTEP");

  if ((DResult == FR_OK) || (DResult == FR_EXIST))
  {
    /*Directory already exists!!*/
  }

  /* Create File */
  DResult = f_open( &fp, "ABSTEP/readme1.txt", (FA_CREATE_NEW | FA_WRITE | FA_READ));
```

Contents

1 Overview

2 Requirements

3 Setup

4 Implementing the application

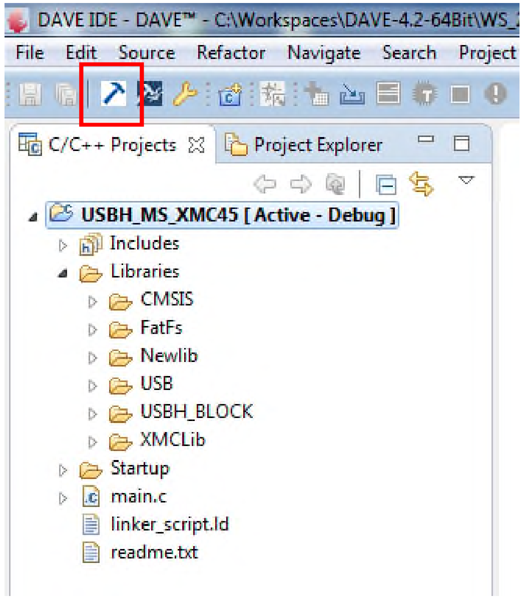
5 How to test

6 Debug hints

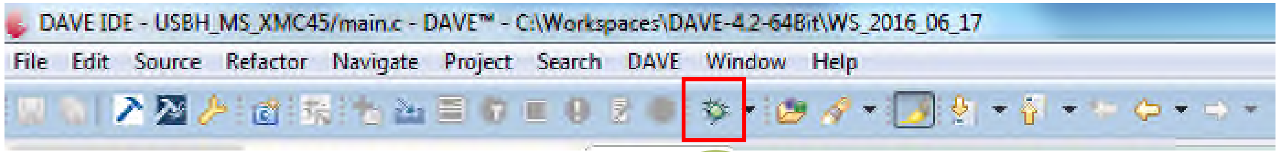
7 Further reading

8 Limitations

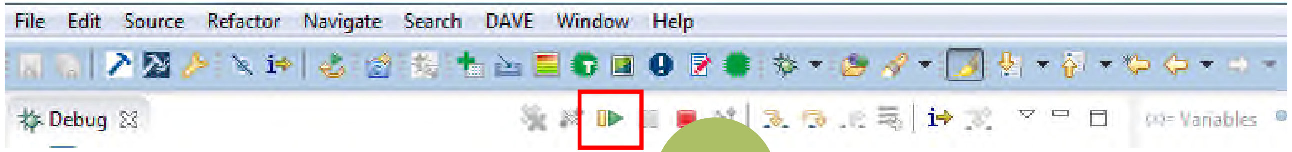
How to test


DAVE IDE - DAVE™ - C:\Workspaces\DAVE-4.2-64Bit\WS_
File Edit Source Refactor Navigate Search Project
C/C++ Projects
Project Explorer
USBH_MS_XMC45 [Active - Debug]
Includes
Libraries
CMSIS
FatFs
Newlib
USB
USBH_BLOCK
XMCLib
Startup
main.c
linker_script.ld
readme.txt

1


DAVE IDE - USBH_MS_XMC45/main.c - DAVE™ - C:\Workspaces\DAVE-4.2-64Bit\WS_2016_06_17
File Edit Source Refactor Navigate Project Search DAVE Window Help

2



3

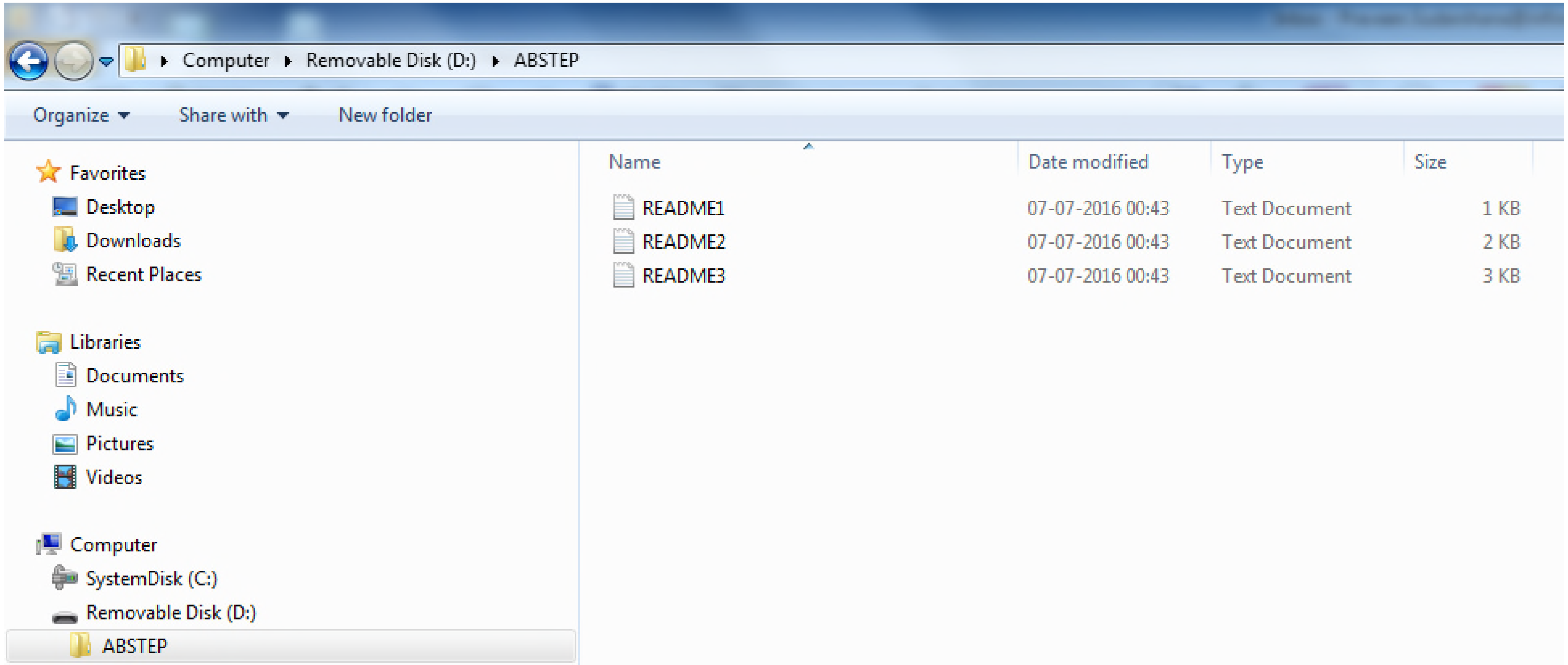
1 Build and download project into XMC4500 General Purpose board

2 Start debugger

3 Start the execution by pressing the run button

Example application behavior

Disconnect the USB flash drive and connect to a Windows laptop to confirm presence of files.



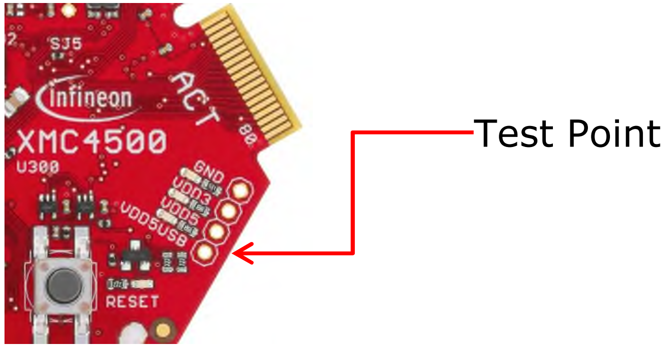
Name	Date modified	Type	Size
README1	07-07-2016 00:43	Text Document	1 KB
README2	07-07-2016 00:43	Text Document	2 KB
README3	07-07-2016 00:43	Text Document	3 KB

Contents

1. Overview
2. Requirements
3. Setup
4. Implementing the application
5. How to test
6. Debug hints
7. Further reading
8. Limitations

Debug Hints

- Check VBUS is available and within permissible range (4.4V to 5.25V) on the USB host port receptacle. Lower VBUS voltages due to excessive current draw may cause USB devices to disconnect intermittently or fail enumeration. Try connecting a different USB device.



- Long cables may cause malfunction due to signal quality issues. Use short cables if possible.

Contents

1. Overview
2. Requirements
3. Setup
4. Implementing the application
5. How to test
6. Debug hints
7. Further reading
8. Limitations

Further reading

- USB specifications and Micro-USB cables and connectors specification

  http://www.usb.org/developers/docs/usb20_docs/

- Mass storage class specifications
  - Mass storage class specification overview
  - Mass storage class bulk only

  http://www.usb.org/developers/docs/devclass_docs/

- SCSI specifications

  www.t10.org

- CMSIS USB host API specification

  https://www.keil.com/pack/doc/CMSIS/Driver/html/group__usb__interface__gr.html

Further reading

- FAT file system

  http://elm-chan.org/fsw/ff/00index_e.html

- LUFA USB stack

  http://www.fourwalledcubicle.com/index.php

- Books
  - USB Complete: The Developer's Guide, Jan Axelson
  - USB Mass Storage: Designing and Programming Devices and Embedded Hosts, Jan Axelson

Contents

1. Overview
2. Requirements
3. Setup
4. Implementing the application
5. How to test
6. Debug hints
7. Further reading
8. Limitations

Limitations

- FAT File system supports only short file names i.e. in 8.3 format.
- USB Hubs are not supported
- Over current protection is not supported
- DMA Transfers are not supported
- Low level driver does not support isochronous transfers

Part of your life. Part of tomorrow.