Application Note

AN_225

FT12 Series Firmware Programming Guide

Version 1.0
Issue Date: 2012-09-25

This document provides guidelines to firmware developers for developing microcontroller applications with FT12 series devices as a USB device peripheral.

Future Technology Devices International Limited (FTDI)
Unit 1, 2 Seaward Place, Glasgow G41 1HH, United Kingdom
Tel.: +44 (0) 141 429 2777 Fax: + 44 (0) 141 429 2758
Web Site: http://ftdichip.com

Table of Contents

1 Introduction

FT12 series of integrated circuits (FT120, FT121 and FT122) are USB device controllers that can be introduced into a microcontroller based system to provide the system with USB connectivity. FT12 devices provide the system designer with the flexibility to design USB devices of various configurations, several numbers of interfaces and several endpoints of different types. USB devices conforming to standard USB classes and vendor specific types can be developed using the FT12x chip, meaning that, USB peripheral devices of various types such as Mass Storage, Human Interface Device(keyboard/mouse/joystick), Printer, Communication Device Class(serial port), etc can be developed using the FT12 series.

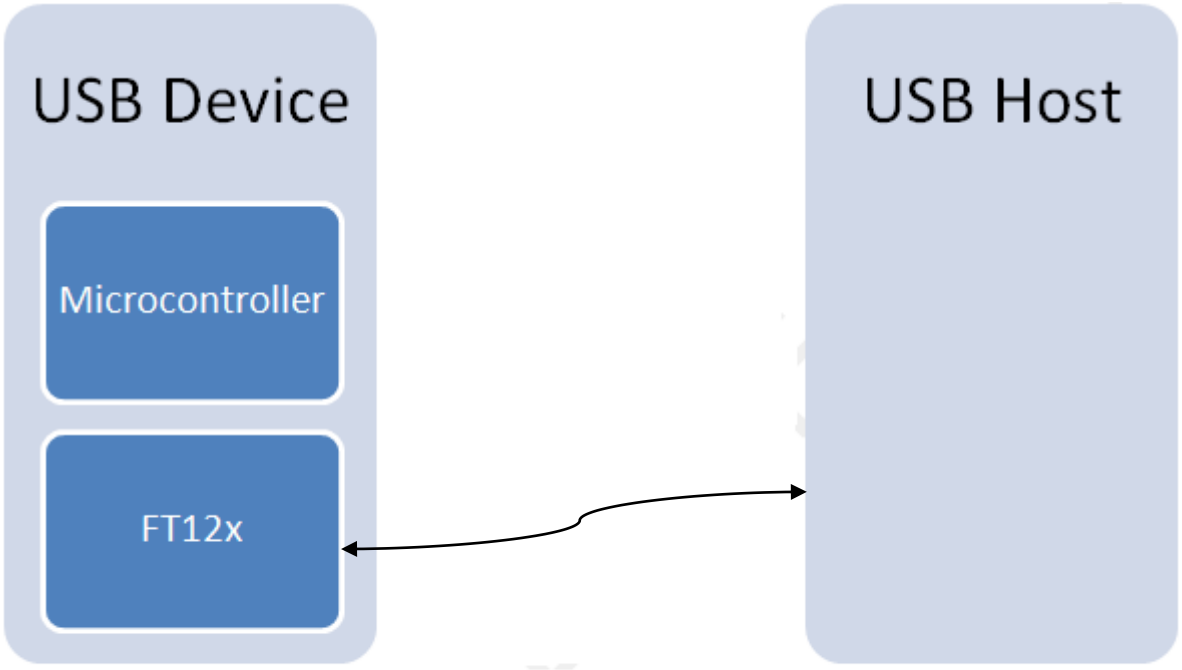

Figure 1-1: FT12x in a USB System

2 Overview of the FT12 Series Device Architecture

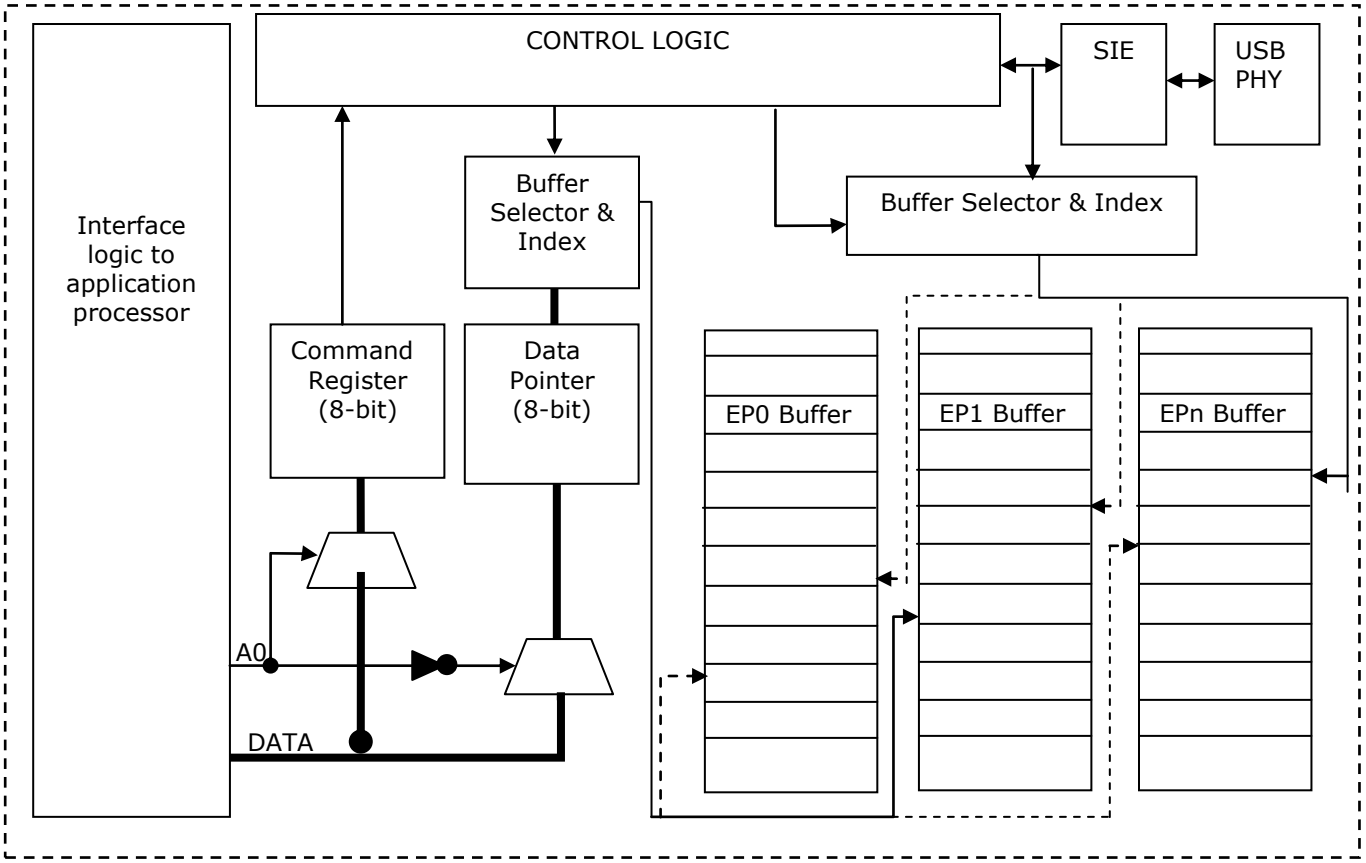

Figure 2-1: FT12 Series Architecture

The above diagram provides a diagrammatic overview of the FT12 series *architecture* that is visible to the programmer.

As it can be seen, the application processor interacts with the FT12x using two 8-bit registers, one which is a command register and other is a data pointer. The data pointer incorporates an auto increment logic, which means that the pointer automatically points to the next byte in the memory once a data byte has been read.

Once the chip has been initialized and the endpoints have been configured, data from the host will be transferred into the respective OUT endpoint buffers and an interrupt will be generated if configured for. Similarly, when the application processor selects an IN endpoint and writes data to it, that data is transferred by FT12x to the host when it receives an IN token in that endpoint, and an interrupt is generated thereafter if configured for. Essentially the FT12x chip will format data from the application processor into USB frames and transmit it to the USB host. It will perform vice-versa when it receives a USB packet from the host.

3 Interfacing FT12 Series Devices

The FT12 series delivers two interface options, one that interfaces with the application processor over parallel I/O lines, and one that interfaces over a Serial Peripheral Interface (SPI).
As it can be seen from figure 2, the FT12x has two memory locations that are visible to the user. One is the command register; the other is the data pointer. A control line, A0, is present in FT120 and FT122 which can be used to select these memory locations.
For FT121, the A0 line is internal to the chip. Every SPI transfer begins with a command phase followed by optional data read or write phases, depending upon the command. The chip internally pulls the A0 line high during the first byte of the transfer and then pulls it low for the subsequent bytes of the transfer. A new command cycle begins every clock after the Slave Select line has been pulled low. Commands and data are grouped together as one SPI transfer if the SPI Slave Select line is held low throughout.

3.1 Interfacing using parallel I/O lines

The application processor can read or write to the FT120/FT122 over parallel I/O using 8 data lines, and two control lines, i.e. RD_N and WR_N. Some microcontroller chips will provide an external bus compatible with this interface whereas others won't. When a compatible external peripheral I/O bus isn't available, the GPIO lines may be used to toggle using firmware to emulate such an external peripheral bus. Typical code to perform read/write would take the following form:

```
void WriteBuffer(bool A0, uint8 *buffer, uint32 size)
{
  uint32 i;
  SetPortOut();    //Sets the 8 GPIO lines connected to the 8 data lines of FT12x to output mode
  if(A0)
            SetA0();              //sets A0 line high
  else
            ResetA0();            //sets A0 line low
  for(i=0;i<size;i++)
  {
            SetDataLines(buffer[i]);        // put data on bus
            Delay(DELAY_WR_SETUP);    // write data setup time

            //strobe WR_N
            SetWriteLow();
            Delay(DELAY_WR); //PULSE WIDTH for WR_N low pulse should be 20nS (minimum)
            SetWriteHigh();
  }
}

void ReadBuffer(bool A0, uint8 *buffer, uint32 size)
{
  uint32 i;
  SetPortIn();          //Sets the 8 GPIO lines connected to the 8 data lines of FT12x to output mode
  if(A0)
            SetA0();              //sets A0 line high
  else
            ResetA0();            //sets A0 line low
  for(i=0;i<size;i++)
  {
            SetReadLow();               //assert RD_N
            Delay(DELAY_RD);            //read data hold time
            Buffer[i]=ReadData();       //read the 8 bits from the bus
            SetReadHigh();
  }
}
```

3.2 Interfacing using SPI

Interfacing with the FT121, the application processor can communicate using only one function that will first send a command and then optionally read or write data. While writing the function we have to make sure that the SPI Slave Select line isn't toggled between the command and the data phases.

```
void SPICommand(uint8 command, bool direction,uint8 *data, uint32 size)
{
  uint32 i;
  SetSSLow();                  //start the command cycle by pulling the Slave Select low
  SPI_Write(command,1);        //write 1 byte command
  If(size)
  {
     if(direction == DIRECTION_WRITE)
        SPI_Write(data,size);
     else
        SPI_Read(data,size);
  }
  SetSSHigh();                 //signal the completion of the command cycle pulling the Slave Select high
}
```

The firmware can attempt to read the VendorID/ProductID/FTDID from the chip to ensure that the connections between the FT12 device and the application processor have been established correctly, and the read/write functions are functioning. Please refer to the datasheet of the IC to obtain the command operational codes and the expected return values. Please note that these commands can work before any initialization commands have been issued to the chip.
It is recommended to use a mutex or a spinlock along with the functions that read/write to the FT12 device to ensure that data are not garbled due to the same set of function getting called from another context.

4 Chip Initialization and Configuration

The chip will need to be initialized before it starts communicating over the USB bus. This is done using the SetMode command. This command is accompanied by two data bytes, the first provides various chip configuration parameters while the second sets the clock divisor for the output clock(CLKOUT) of the chip. After this, optionally for FT120 and FT122, if the application processor supports DMA data transfers then the DMA configuration can be set using the SetDMA command. Once these have been done, if the chip is FT121 or FT122 then the default endpoint configuration can be modified using the SetEndPointConfiguration command. The endpoint index 0 and 1 are configured as control endpoint, with buffer size 16 and in enabled state by default, and tthis cannot be modified.

Typical code for the initialization sequence may take the form:

```
void FT12x_Init(void)
{
  FT12x_SetMode(FT12X_NOLAZYCLOCK, FT12X_SETTOONE | FT12X_CLOCK_12M);//disconnect USB
  Delay(DELAY_100MS);
  FT12x_SetMode(FT12X_ENDP_NONISO|FT12X_DP_PULLUP,
          FT12X_SETTOONE|FT12X_CLOCKRUNNING|FT12X_CLOCK_12M);//connect USB
  #ifndef FT120       //this command applicable only for FT121 & FT122
          FT12x_SetEndpointConfig(ENDPOINT_2,1,1,3);
          FT12x_SetEndpointConfig(ENDPOINT_3,1,1,3);
  #endif
}
```

Other than the above, one also has to consider that typically the USB data will be handled by the application processor based on interrupts. The firmware has to configure the interrupt pin associated with the FT12x interrupt while the firmware configures the other interrupts and peripherals connected to the application processor.

5 FT12 Device Interrupt handling

Once the chip has been initialized and the DP_PULLUP bit has been set using the SetMode command, the USB host will start the enumeration sequence as shown below.

Sp	Index	m:s.ms.us	Len	Err	Dev	Ep	Record	Summary
FS	13	1:08.308.297	30.9 ms				<Reset> / <Chirp J> / <Tiny J>	
FS	14	1:08.339.266					<Full-speed>	
FS	15	1:08.371.533	16 B		00	00	Get Device Descriptor	Index=0 Length=64
FS	16	1:08.371.533	8 B		00	00	SETUP txn	80 06 00 01 00 00 40 00
FS	17	1:08.371.533	3 B		00	00	SETUP packet	2D 00 10
FS	18	1:08.371.536	11 B		00	00	DATA0 packet	C3 80 06 00 01 00 00 40 0C
FS	19	1:08.371.545	1 B		00	00	ACK packet	D2
FS	20	1:08.372.534	16 B		00	00	IN txn [6 POLL]	12 01 10 01 EF 02 01 10 03
FS	21	1:08.372.534	5.00 ms		00	00	[6 IN-NAK]	
FS	22	1:08.378.534	3 B		00	00	IN packet	69 00 10
FS	23	1:08.378.538	19 B		00	00	DATA1 packet	4B 12 01 10 01 EF 02 01 1C
FS	24	1:08.378.552	1 B		00	00	ACK packet	D2
FS	25	1:08.380.535	0 B		00	00	OUT txn	
FS	26	1:08.380.535	3 B		00	00	OUT packet	E1 00 10
FS	27	1:08.380.538	3 B		00	00	DATA1 packet	4B 00 00
FS	28	1:08.380.541	1 B		00	00	ACK packet	D2
FS	29	1:08.381.602	5.25 us				<Reset> / <Target disconnecte...	
FS	30	1:08.381.608	20.0 ms				<Reset> / <Chirp J> / <Tiny J>	
FS	31	1:08.401.668					<Full-speed>	
FS	32	1:08.433.542	0 B		00	00	Set Address	Address=01
FS	42	1:08.464.546	18 B		01	00	Get Device Descriptor	Index=0 Length=18
FS	61	1:08.480.549	100 B		01	00	Get Configuration Descriptor	Index=0 Length=255
FS	105	1:08.523.555	18 B		01	00	Get String Descriptor	Index=3 Length=255
FS	124	1:08.535.556	4 B		01	00	Get String Descriptor	Index=0 Length=255
FS	138	1:08.541.557	68 B		01	00	Get String Descriptor	Index=2 Length=255

Figure 5-1: USB Enumeration

The first USB packet that the FT12x device will receive will be a SETUP packet which will be a part of GetDeviceDescriptor. On receiving this packet the FT12x will generate an interrupt. When that happens, the application processor should read the interrupt register in FT12x, on doing that it will know that data has been received on endpoint 1. The handler for endpoint 1 should then issue the ReadLastTransactionStatus command to find out if the packet was a SETUP packet and initialize the control endpoint handler state machine to service a control request.
The interrupt service routine for FT12 device interrupt may take the following form:

```
void OnFT12xInterrupt()
{
    unsigned int interrupt_status;
    DisableFT12xInterrupt();//Disable the interrupt in the application processor that is connected to FT12x interrupt
    interrupt_status = FT12x_ReadInterruptRegister();//read which interrupt is it
    switch(interrupt_status & 0xFFFF)     //service the interrupt
    {
        case INTERRUPT_EP0:
            ep0_txdone();
            break;
        case INTERRUPT_EP1:
            ep0_rxdone();
            break;
        case INTERRUPT_EP2:
            ep1_txdone();
            break;
        case INTERRUPT_EP3:
            ep1_rxdone();
            break;
```

```
            ...
            ...
            ...
    }
    EnableFT12xInterrupt();
}

void ep1_txdone(void)          //data written to the EP has been transferred to host; clear the interrupt
{
    unsigned char ep_last;
    ep_last = FT12x_ReadLastTransactionStatus(3);   // Clear interrupt flag
}

void ep1_rxdone(void)          //data received from host, read it into buffer
{
    unsigned char len;
    unsigned char ep_last;
    ep_last = FT12x_ReadLastTransactionStatus(2); // Clear interrupt flag
    len = FT12x_ReadMainEndpoint(buffer+byteCounterRx); //Read data from FT12x to local memory
    RaiseFlagToProcessData(); //Raise some flag to process the received data in another thread or idle loop
}
```

Other than data transfer, the FT12x also raises interrupts when the host suspends the bus, performs a bus reset, or when the chip completes a DMA transfer. These conditions can be checked for by testing the specific bit in the returned value of the ReadInterruptRegister command, and then may be handled accordingly. For example, the USB device may be programmed for power saving mode after it sees a suspend interrupt, or it may wish to flush its buffers and reinitialize its buffer index after it receives a bus reset interrupt.

6 USB Device Enumeration

The first pair of endpoints(endpoint 0 and 1, or pipe 0) is a pair of control endpoints that receive standard, class specific and vendor specific requests and respond to them. The USB device enumeration requires only these two endpoints to complete the enumeration. Pipe 0 OUT is endpoint number 1. A control transfer consists of a SETUP transaction followed by one or more optional DATA (in or out) transaction(s), followed by a STATUS transaction as shown below

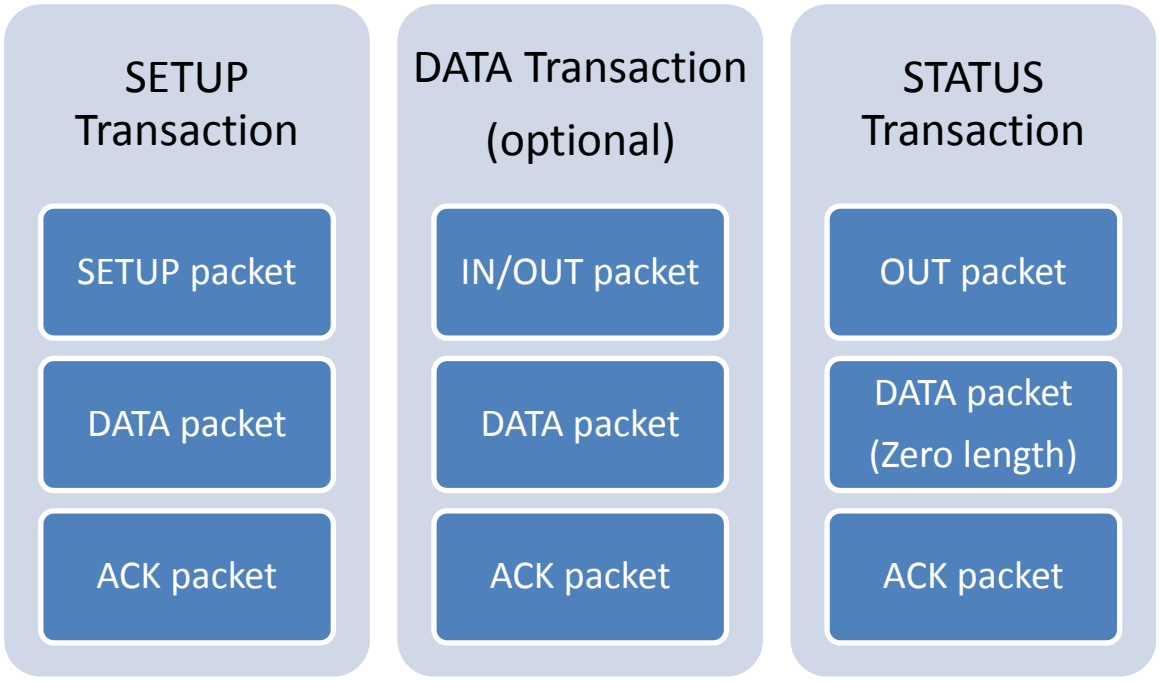

Figure 6-1: Control Transfer

An example of SETUP transaction can be seen in figure 3 where the standard USB request GetDeviceDescriptor is serviced. Typically, the following is the sequence that a USB host will follow to enumerate a device:

1) On detecting a USB device plugged into the bus, the host will issue a reset
2) Issue standard GetDeviceDescriptor request.
3) After the device transfers the first 8 bytes of the device descriptor, the host issues another bus reset.
4) The host will now put the device into addressed state by sending the SetAddress request. The device firmware should take the address sent by the host and set its own address to this by using the FT12x SetAddressEnable command.
5) Then the host will ask for the entire device descriptor by issuing the GetDeviceDescriptor request once again.
6) After that the host will request for device configuration by issuing the GetConfigurationDescriptor request. Note that some hosts make this request in two stages, first it requests for only the first 9 bytes of the Configuration Descriptor and then it sends the request once again to get the entire descriptor. Also, depending upon the intended functionalities of the device, the configuration descriptor can have many sub descriptors like endpoint descriptors, interface descriptors, metadata/companion descriptors, etc.
7) Lastly the host will issue the GetStringDescriptor request to obtain the manufacturer, product and serial number strings.

7 Example Firmware

FTDI is providing example firmware that runs on a LPC1114 microcontroller connected to FT12 series device. One such example connects to the USB host as a composite device having a CDC-ACM serial port interface and a HID keyboard interface. When this device is connected to a host (a windows PC), a loopback serial port appears on the PC. Any data written to this serial port will be loopback back to the PC. An additional keyboard will also appear connected to the PC. This keyboard uses two keys (press-buttons) and two LEDs of the UMFT12XEV Evaluation board. The two keys and the LEDs correspond to CAPSLOCK and NUMLOCK of a normal keyboard.

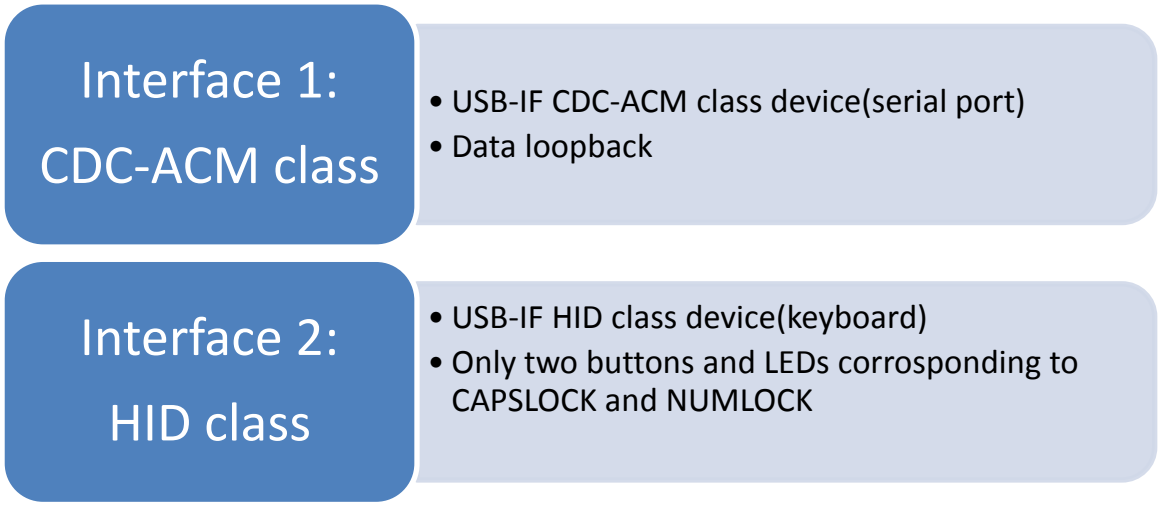

Figure 7-1: USB interfaces of the example USB device firmware

Pipe	Endpoint	Direction	Type	Interface
0	0	OUT	Control	
	1	IN	Control	
1	2			CDC-ACM Control
	3	IN	Interrupt	
2	4	OUT	Bulk	CDC-ACM DATA
	5	IN	Bulk	
3	6			HID
	7	IN	Interrupt	

Figure 7-2: Endpoint map

7.1 LPC1114 Microcontroller

The LPC1114 is a low cost 32bit ARM Cortex-M0 CPU based microcontroller from NXP Semiconductors. It has 8 kilobytes SRAM, 32 kilobytes flash and it can run at frequencies up to 50MHz. The microcontroller features a serial wire debug, system tick timer, nested vectored interrupt controller, 10-bit ADC, UART, SPI, I2C and WDT.

7.2 LPCXpresso Target Board

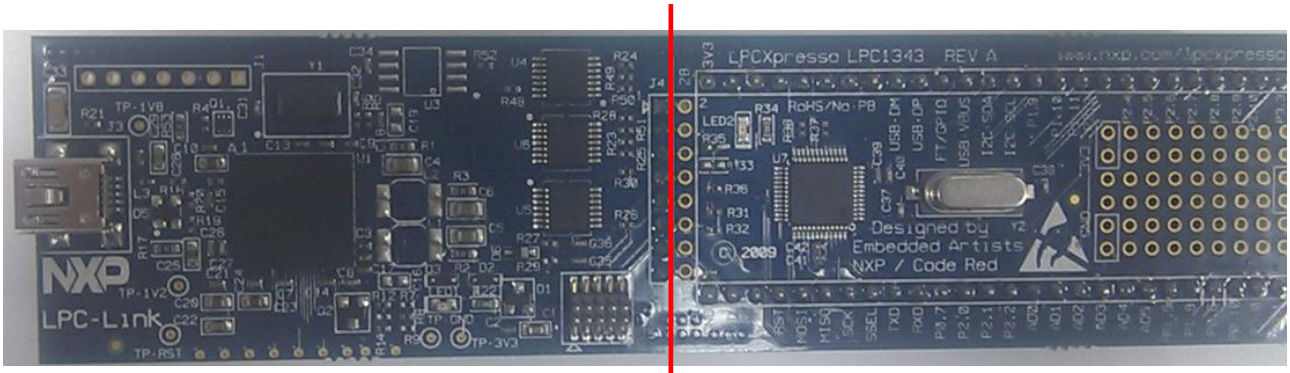

Figure 7-3: LPCXpresso Target Board populated with LPC1114

Figure 7 shows a LPCXpresso Target Board populated with a LPC1114 microcontroller. The part of the board to the left of the red line is NXP's debugging hardware for LPC microcontrollers. The board can actually be cut into two parts and the LPC1114 target in the left of the red line can work independently if no debugging is required.

7.3 UMFT12XEV Evaluation Kit

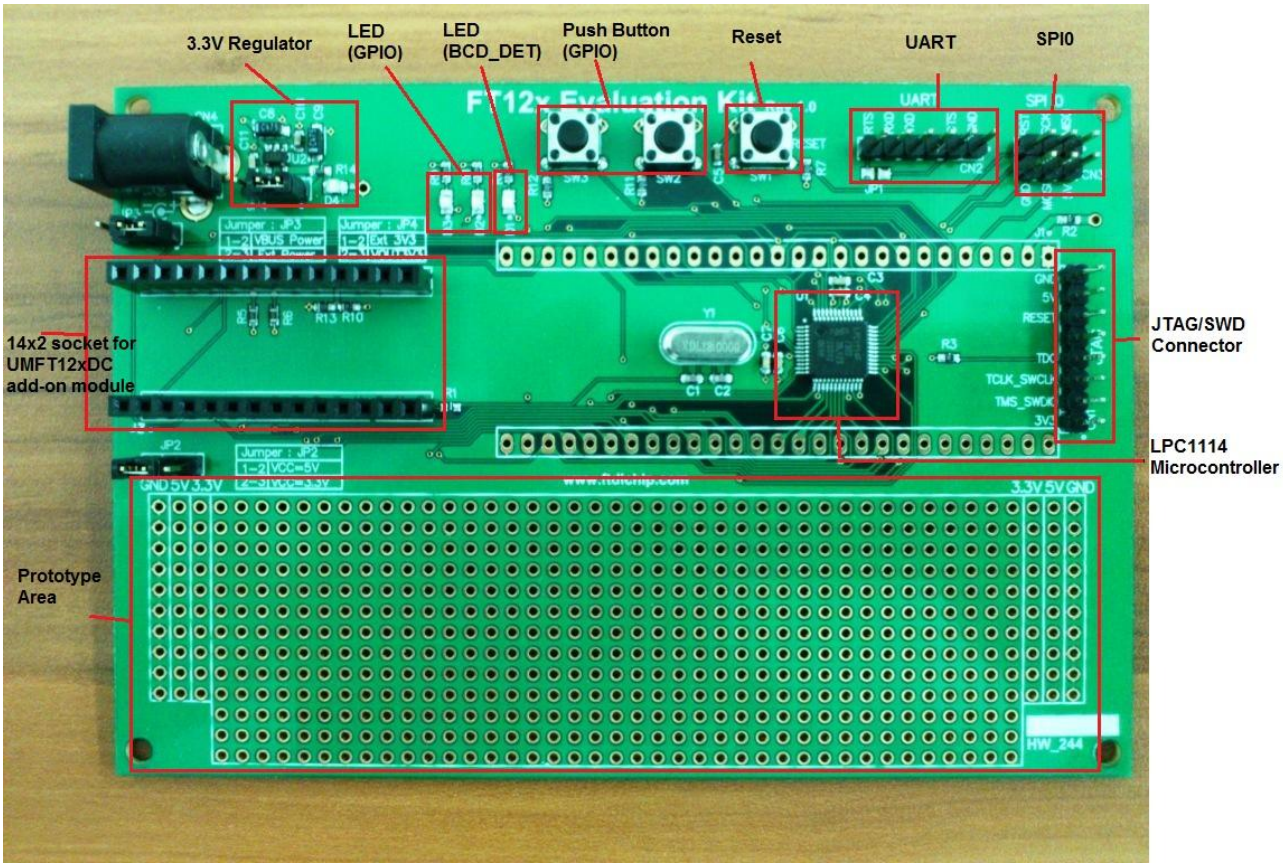

Figure 7-4: UMFT12XEV Evaluation Kit

Binary images of the firmware can be downloaded to the board through the UART port, or the left half of the LPCXpresso board(the debugger part) can be connected to CN1(8 pin JTAG header on the extreme right in figure 8) to debug the LPC1114 on the UMFT12XEV Evaluation Kit.

7.4 LPCXpresso IDE

The LPCXpresso IDE is an eclipse based integrated development environment for LPC microcontrollers that comes with everything that is required to develop and debug applications. This tool is available for free download from Code Red Technologies (requires registration).
After LPCXpresso has been installed and registered, while executing the program it will ask for the workspace path(shown in figure 9). Here browse to the folder of the provided example source root directory that contains the directory ".metadata" (see section 7.5 for source tree directory structure).

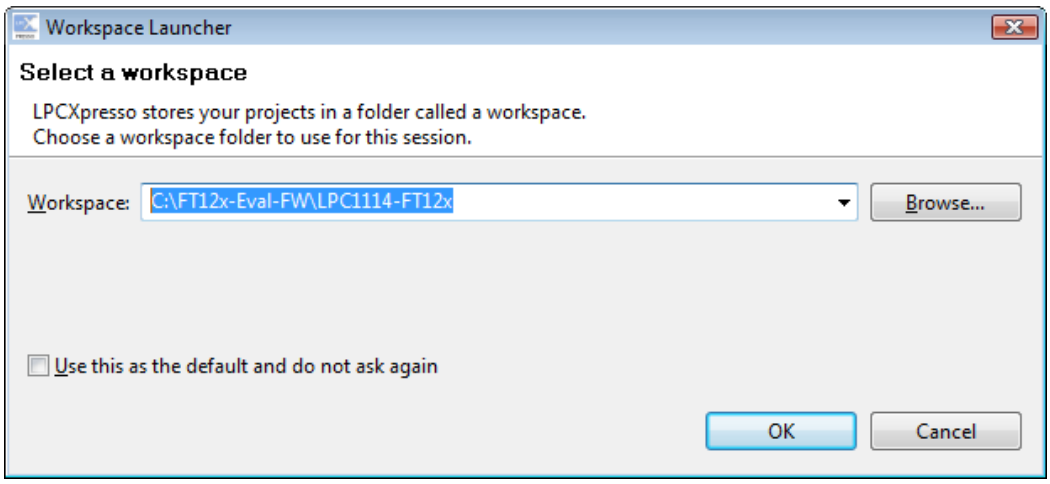

Figure 7-5: LPCXpresso Select workspace path

Once this is done, the example firmware project will be loaded into the workspace and it can either be built so that the binary can be downloaded into the LPC1114, or it can be debugged by right-clicking on LPC1114-FT12x in the project explorer window and on the context menu selecting Debug As -> C/C++ MCU Application, as shown in figure 10.
The LPCXpresso features standard building and debugging features, the software's help documentation provides details about it.

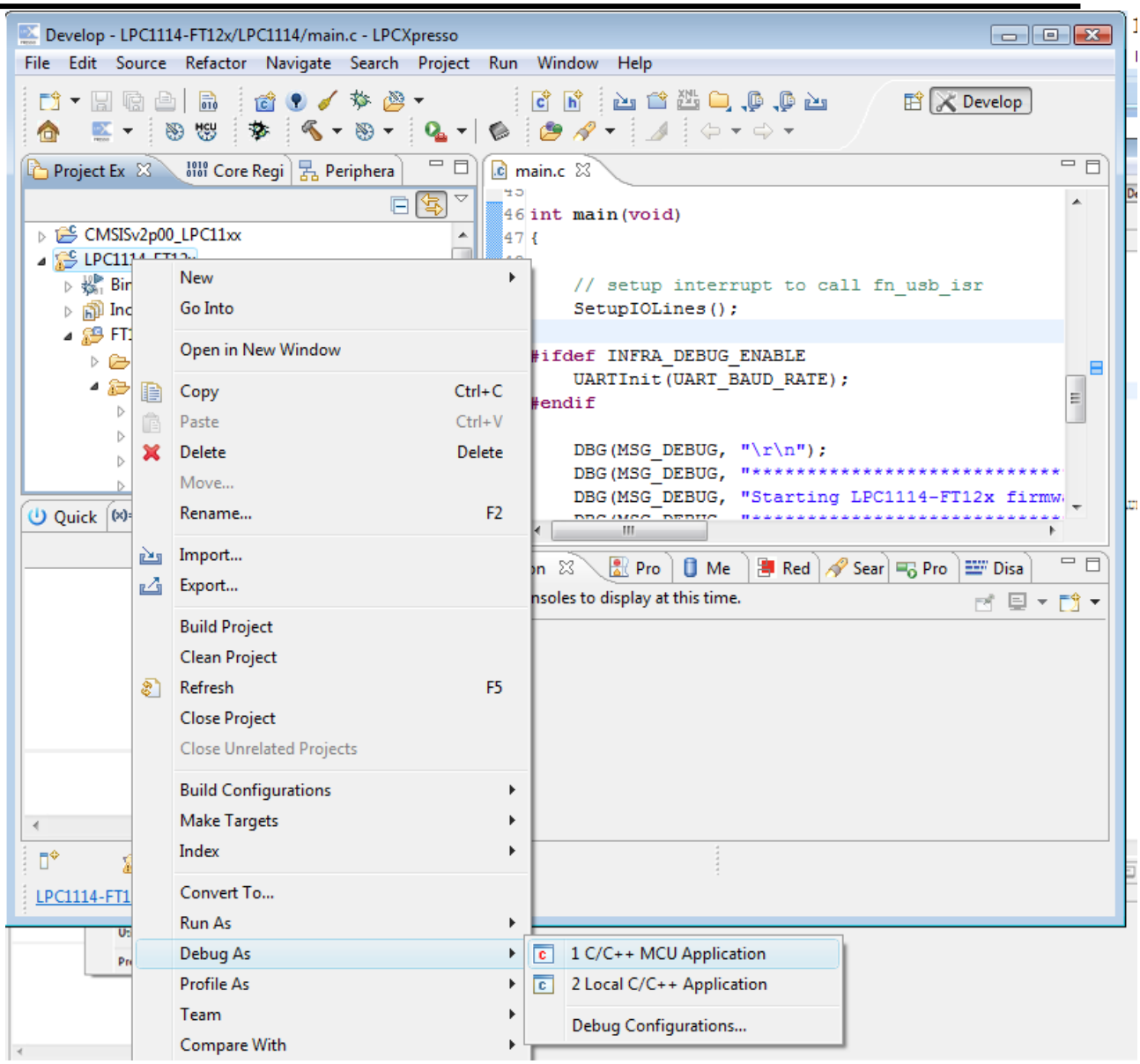

Figure 7-6: Debugging using LPCXpresso

7.5 Firmware directory structure

- Project Root
  - .metadata — *LPCXpresso generated directory*
  - CMSISv2p00_LPC11xx — *Standard CMSIS library for Cortex-M0*
  - LPC1114-FT12x
    - .cproject
    - .project
    - .settings
    - FT12x — *Contains all USB specific code*
      - Include
        - al.h
        - chap_9.h
        - ci.h
        - ftdi.h
        - mainloop.h
        - usb.h

- src
  - chap_9.c
  - ci.c
  - ftdi.c
  - fw.c
  - isr.c
- LPC1114 *Contains all LPC1114 specific drivers*
  - cr_startup_lpc11.c
  - gpio.c
  - gpio.h
  - main.c
  - ssp.c
  - ssp.h
  - uart.c
  - uart.h

The directory CMSISv2p00_LPC11xx contains standard CMSIS (Cortex Microcontroller Software Interface Standard) files provided by ARM.
The files and directories that begin with a "." (dot) are generated by LPCXpresso IDE.

7.6 The Reference Firmware

The reference firmware provided is divided into two directories, the LPC1114 directory has the uart drivers(uart.h & uart.c), the SPI drivers(ssp.h & ssp.c), the GPIO drivers(gpio.h & gpio.c), startup functions(cr_startup_lpc11.c) and firmware start point(main.c). Please note that the drivers are example code provided with LPCXpresso and the startup code is provided by Code Red Technologies, and they come with their own liabilities, please check the comments sections in the respective file headers.
The FT12x directory contains all FTDI specific code where:

- chap_9.c contains all USB2.0 specification chapter 9 specific code.
- ci.c contains all the FT12x specific commands
- ftdi.c contains hardware specific helper functions
- fw.c contains the main(idle) loop and several important functions
- isr.c contains the USB interrupt service routine & subroutines

After starting a new LPC1114 project on LPCXpresso, the cr_startup_lpc11.c was modified at the following sections to customize it for the FT12x reference firmware:

- The interrupt vector table: to add function pointers to SPI, UART & GPIO interrupts handlers in the respective drivers.
- The SysTick_Handler interrupt handler: to update the ClockTicks.

The main() function in main.c is called from ResetISR after BSS and libraries have been initialized. The main() function setups the directions of the I/O lines and their initial values, setups the interrupts, initializes microcontroller peripherals and then calls FT12x_main(). The call graph for function FT12x_main() is shown in figure 11.

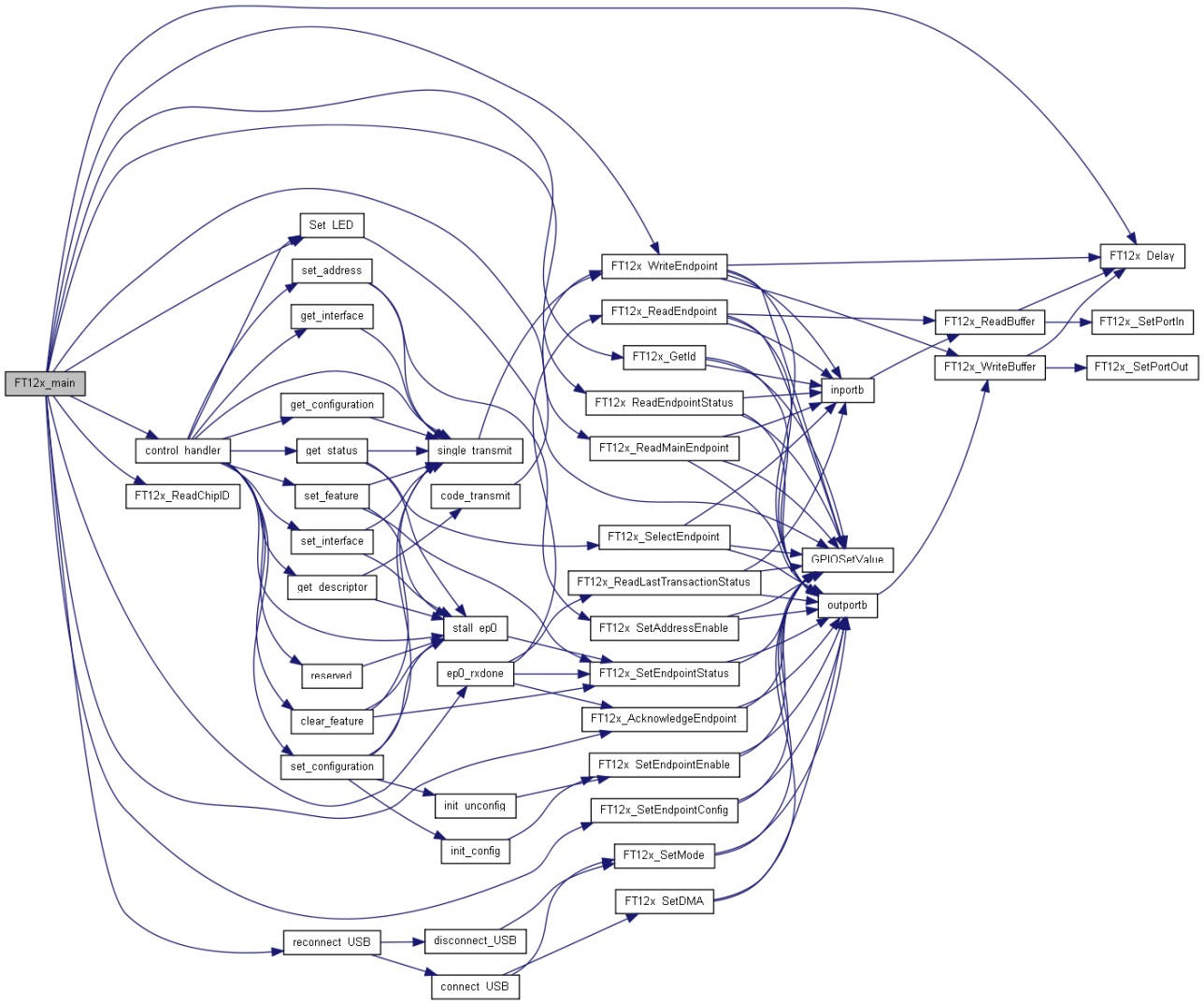

Figure 7-7: Call graph of FT12x_main

FT12x_main initializes the FT12x and then goes into a continuous loop that services control, HID and CDC requests when they are sent from the host. The continuous loop may be divided into the following parts:

- Process control requests
  - Standard USB requests
    - Get Status
    - Clear Feature
    - Set feature
    - Set Address
    - Get Descriptor
    - Get Configuration
    - Set Configuration
    - Get Interface
    - Set Interface
  - Class specific requests
    - HID requests
      - Set Idle
      - Set Out Report
    - CDC-ACM requests
      - Get Line Coding
      - Set Line Coding

- Set Control Line State

- Process HID requests
  - Write Key Data to HID Interrupt-IN endpoint when key is pressed on board
- Process CDC-ACM requests
  - If data had been received from host and are pending to be looped back then write a chunk of that data to the CDC-ACM Bulk-IN endpoint

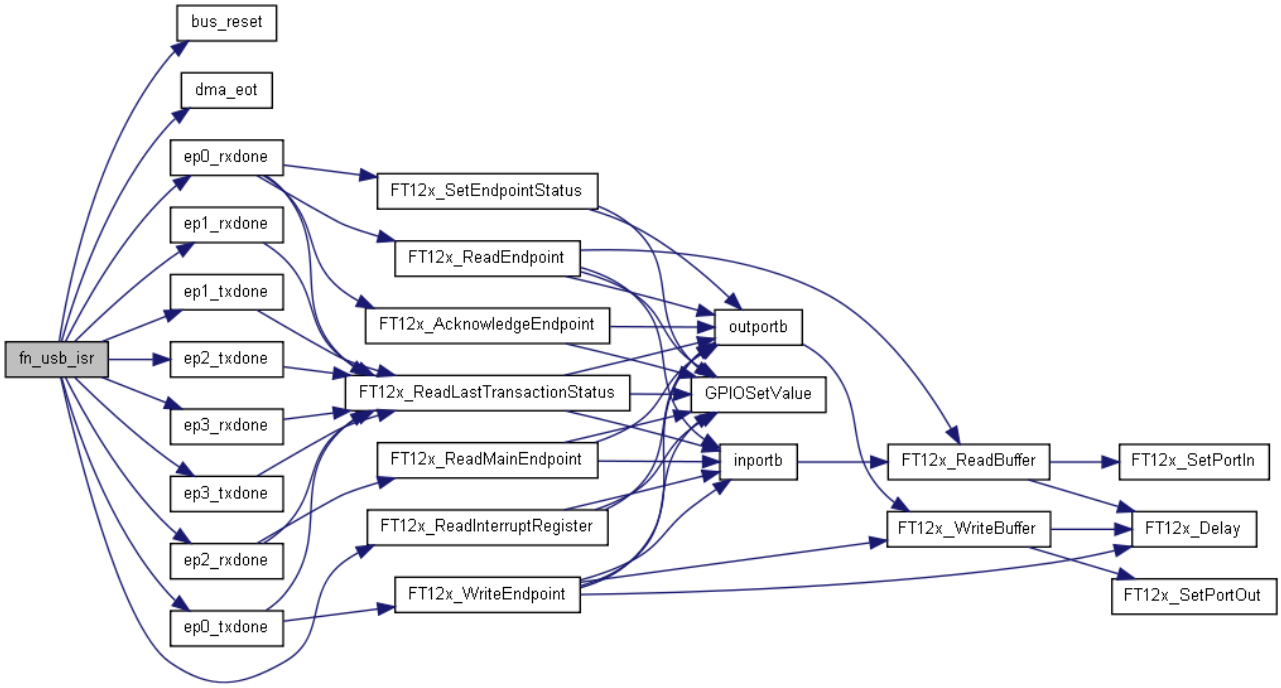

Figure 7-8: Call graph for USB ISR

GPIO pin1.11 is configured to call function fn_usb_isr() when FT12x raises a USB interrupt. The call graph for fn_usb_isr() is shown in figure 12. On entering the function the firmware issues command ReadInterruptRegister to FT12x to know what caused the interrupt. Then it will call the respective endpoint handler. If data was received from the host then the endpoint handler will copy the data from the FT12x to its local buffer, perform any minimum processing required (lower half processing) and then signal the main loop to perform the bulk of the processing by raising a flag.
Following are the other contexts that run the firmware apart from the USB ISR and the main loop:

- SysTick handler: This interrupt handler only increments a software clock for internal reference.
- GPIO pin 1.0 interrupt handler: This pin is connected to SW2 on the FT12x board. When this switch is pressed a CAPSLOCK toggle event is registered. This event is later detected in the main loop and the information is formatted into a report buffer and sent to the USB host via the HID interfaces interrupt endpoint.
- GPIO pin 1.1 interrupt handler: This pin is connected to SW3 on the FT12x board. When this switch is pressed a NUMLOCK toggle event is registered. This event is later detected in the main loop and the information is formatted into a report buffer and sent to the USB host via the HID interfaces interrupt endpoint.
- GPIO pin 3.2 interrupt handler: This pin is connected to push button SW1 on the FT12x Evaluation board. When the button is pushed the corresponding ISR calls the BSP's system reset function.

The LEDs D2 and D3 on the UMFT12XEV Evaluation board correspond to key status of CAPSLOCK and NUMLOCK respectively. They are toggled when the host sends class specific request Set Output Report to the control endpoint.

7.7 Recommendations for porting to other MCUs

Given below are some recommendations that may be useful while porting this firmware to another microcontroller:

- If using Parallel I/O, modify functions FT12x_WriteBuffer and FT12x_ReadBuffer in file ftdi.c to suite the connection and timings.
- If using SPI, replace SSP_Send() in ci.c with the appropriate SPI data transferring function.
- From the main function call FT12x_main after the microcontroller specific & peripheral initializations are done.
- Call function fn_usb_isr() in isr.c from the ISR associated with the interrupt pin of FT12x.
- Set variable KbdDataAvailable=0x39 and KbdDataAvailable=0x53 respectively in the ISRs associated with the CAPSLOCK and NUMLOCK keys.
- Modify function Set_LED() in ftdi.c to turn ON/OFF CAPSLOCK and NUMLOCK status LEDs.
- Modify the macros ENABLE and DISABLE in ftdi.h to enable and disable interrupts for that MCU.
- Configure a timer interrupt to increment variable ClockTicks approximately every 10mS.

.

8 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) sales1@ftdichip.com
E-mail (Support) support1@ftdichip.com
E-mail (General Enquiries) admin1@ftdichip.com

Branch Office – Hillsboro, Oregon, USA

Future Technology Devices International Limited
(USA)
7130 SW Fir Loop,
Tigard, OR 97223
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales) us.sales@ftdichip.com
E-Mail (Support) us.support@ftdichip.com
E-Mail (General Enquiries) us.admin@ftdichip.com

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited
(Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan , R.O.C.
Tel: +886 (0) 2 8791 3570
Fax: +886 (0) 2 8791 3576

E-mail (Sales) tw.sales1@ftdichip.com
E-mail (Support) tw.support1@ftdichip.com
E-mail (General Enquiries) tw.admin1@ftdichip.com

Branch Office – Shanghai, China

Future Technology Devices International Limited
(China)
Room 1103, No. 666 West Huaihai Road,
Shanghai, 200052
China
Tel: +86 21 62351596
Fax: +86 21 62351595

E-mail (Sales) cn.sales@ftdichip.com
E-mail (Support) cn.support@ftdichip.com
E-mail (General Enquiries) cn.admin@ftdichip.com

Web Site

http://ftdichip.com

Appendix A – References

Document References

FT120 Data Sheet
FT121 Data Sheet
FT122 Data Sheet

USB 2.0 Specification
USB CDC-ACM Class Specification
USB HID Class Specification
ARM Cortex M0 Technical Reference Manual
LPC1114 Datasheet
LPC1114 User Manual
Cortex Microcontroller Software Interface Standard(CMSIS) Specification
LPCXpresso User Guide – Getting Started with NXP LPCXpresso

Acronyms and Abbreviations

Terms	Description
ADC	Analogue to Digital Converter
CDC	Communication Device Class
DMA	Direct Memory Access
HID	Human Interface Device
I2C	Inter Integrated Circuit
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
WDT	Watch Dog Timer

Appendix B – List of Tables & Figures

List of Figures

Appendix C – Revision History

Document Title: AN_225 FT12 Series Firmware Programming Guide
Document Reference No.: FT_000748
Clearance No.: FTDI# 316
Product Page: http://www.ftdichip.com/FTProducts.htm
Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial Release	25-09-2012