Application Note

AN_223

How To Use The FT12 Series Devices

Version 1.0

Issue Date: 2012-07-23

FTDI's FT12 series of devices are full speed USB device IC's providing designers the opportunity to add USB interfaces to their designs. This document explains the benefits of selecting the FT12 series solution and how to implement it.

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

FTDI is well known for developing silicon and drivers that enable engineers to convert a peripheral device serial port to USB. This silicon is simple to install and the end user does not require any driver development to complete the design. The FT12 series will still provide a bridge between USB and the main embedded processor, but instead of connecting to the UART it will connect to the main processor bus (FT120, FT122) or an SPI master (FT121). The devices are also more configurable, allowing for different USB Device Classes to be designed. As such this requires the developer to create some supporting firmware for the embedded processor to support the FT12 devices. This application note will offer some guidance in the design steps required to select and implement an FT12 device in a design.

2 Selecting the Correct Device

An FT12 series device may be chosen for adding a USB device port to a design for a variety of reasons.

Primarily it is chosen because the design requires a USB device port. However there are other key considerations when selecting the FT12x over other solutions. The fact the device can be configured to appear as different device classes offering, bulk, isochronous and interrupt endpoints offers maximum flexibility. The internal DMA engines remove a lot of work from the main processor and the fact it is a separate device as opposed to an internal USB engine in the main processor provides some isolation and protection against misuse.

2.1 FT120

The FT120 is designed to appear as another peripheral on the processor bus. It contains a chip select line as well as an address and data bus which will act as any other peripheral on the bus. The device is also a drop in alternative to the now obsolete PDIUSBD12, allowing for legacy designs to have an extended life.

2.2 FT121

The FT121 provides the same programming flexibility as the FT120 and FT122, but is connected to the main processor as an SPI slave. This allows the device to be presented in a smaller package (16 pins as opposed to 28). The FT121 also provides battery charger detection, making it ideal for portable / battery powered designs that are rechargeable from the USB port.

2.3 FT122

The FT122 enhances the basic FT120 design by adding battery charger detection similar to the FT121.

3 Example Circuit

The diagram below shows how the FT120 may be connected to create a USB bridge to a main processor bus.

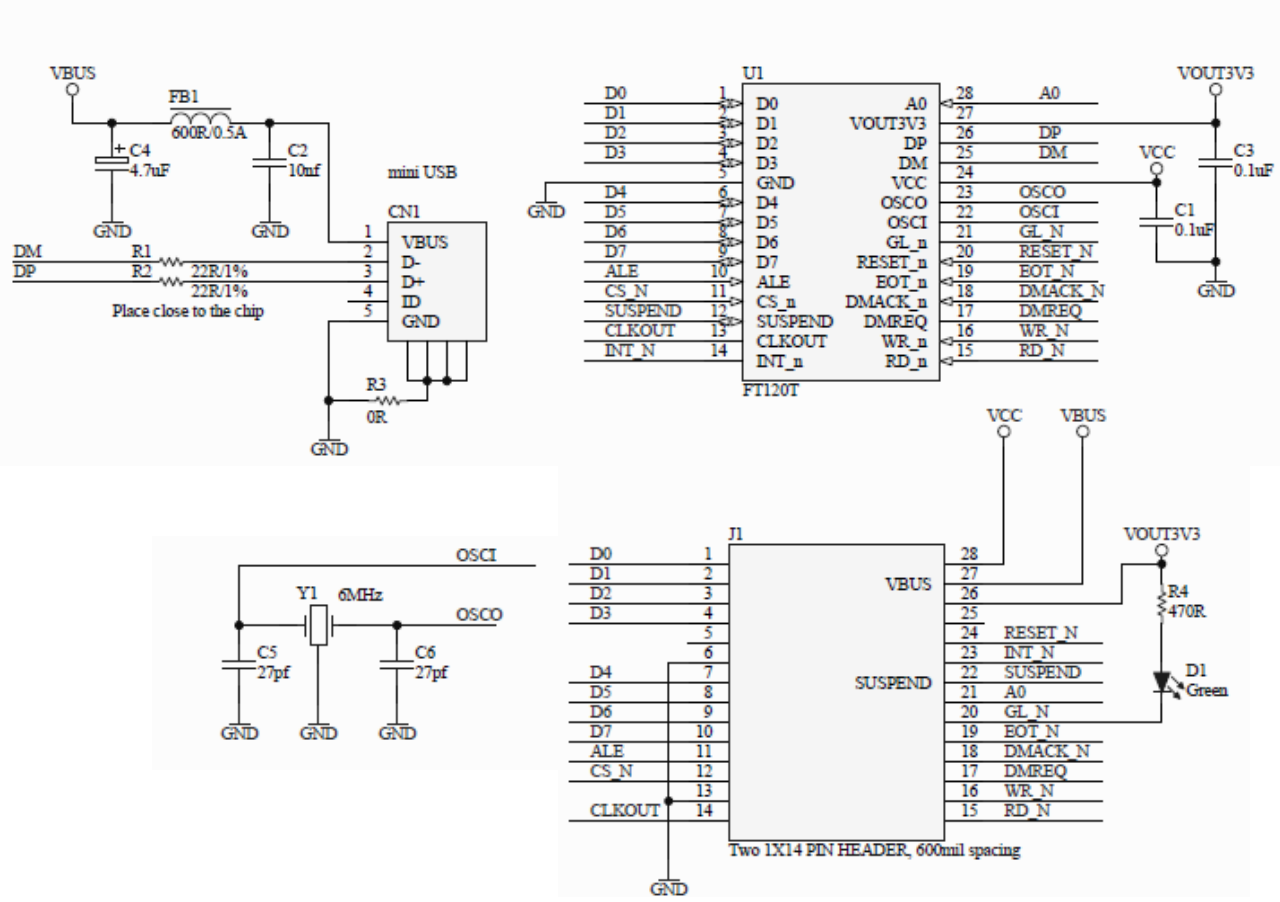

Figure 3.1 FT120 Schematic

4 Coding the FT12

Using the FT12 series of devices does rely on the developer having some understanding of USB protocol to allow for the configuration of the device. Once configured the reading and writing of data is relatively simple.

Example code that demonstrates configuring the device as a CDC class or a HID class USB deviceis available for download at:

www.ftdichip.com/support/software_examples/FT12.htm

4.1 Key Decisions

There are key decisions to be made before coding can begin.

4.1.1 Device Class

Start by determining what device class the design should present itself as.

If it is a "Vendor Specified" class then the developer will require coding for the FT12x AND drivers for the host PC.

If the design uses standard USB device classes then only the FT12x needs coded as there will most likely be standard drivers on the host PC.

A large part of this design decision will be based on function as well as perceived end user experience.

Standard USB Device Classes are defined by the USB Implementers Forum and may be found here.

4.1.2 Number of Endpoints

Endpoints are essentially the USB addresses that USB data is sent to or from.

An IN Endpoint will transfer USB data from the FT12x to the host PC, while an OUT Endpoint will transfer USB data from the host PC to the FT12x. Another USB rule is that an IN endpoint has an odd address while an OUT uses an even address.

The number of endpoints is largely defined by the device class chosen. A user may add additional endpoints, but this would then start to move the overall design to require dedicated drivers on the host PC.

All designs will include a control endpoint for configuration and control. This will be typically endpoint 0.

4.1.3 USB Transfer Mode

The transfer mode will be largely defined by the device class selected also.

A device that uses isochronous transfers has no error checking, but will send regular sized chunks at regular intervals. Applications such as audio may be suited to this.

A device that uses interrupt transfers will be receiving data at unspecified intervals, such as Human Interface Devices (HID). This may include mouse and keyboard interfaces.

The third main transfer type is bulk mode transfer. This is used for transferring large blocks of data and includes error checking.

4.1.4 Descriptors

Descriptors are fundamentally a list of constants that the USB host will use to match a USB device to a USB driver. The FT12 design must be given a unique set of descriptors to allow it to be identified on the USB bus. Key descriptors include Vendor Identifier (VID), Product Identifier (PID),

Manufacturer String, Product Description String and Serial Number. The developer can select their own values for these parameters, with the exception of the VID, which should be applied for via the USB Implementers Forum (www.usb.org)

4.2 Setup Code

To configure the FT12x the USB host will send standard instructions over the USB interface on the control endpoint. The main processor connected to the FT12x must decode these setup requests to supply the correct response via the FT12x control endpoint.

Standard setup requests are documented in the USB specification – (see chapters 8 and 9 USB Specification), and are not specific to FTDI.

Vendor commands within the setup request may be unique to the device class implemented.

```
do {
    //-------------------------------------------------------------------
    // SETUP :
    //
    // When a Setup Packet is received on the Control Endpoint process it
    //here.
    //-------------------------------------------------------------------
    if (SUT_Received) {
      #ifdef BUSY_CNT
      BUSY |= 1;
      #endif

      ctrl_in_queue_size = 0;
      ctrl_in_queue_zb   = 0;
      Do_Validate        = 1;

      // Read data out of Control Endpoint into setup_pkt variable
      FT120_RD_BUFFER_int(SEL_EP_CTRL_OUT, (BYTE *)&setup_pkt[0]);

      // Setup packet is a Vendor command, Stall if USB not Configured or
      // wLength MSB non Zero
      if (setup_pkt[0] & BIT(6)) {
        //if (USB_STATE != USB_CONFIGURED) {
        if (USB_STATE == USB_DEFAULT) {
          Stall_CTRL();
        } else if (setup_pkt[7] != 0x00) {
          Stall_CTRL();
        } else {
          Process_Vendor();
        }

      // Setup packet is a Class Type or Reserved, so set Stall as these
      // are not supported by the device beign created
      } else if (setup_pkt[0] & BIT(5)) {
        Stall_CTRL();

      // Standard Setup token, so process the packet.
      } else {
        Process_SUT();
      }

      // Acknowledge the setup packet in the FT120 and clear the control
      // buffers to allow further packets to be received
      FT120_CMD = SEL_EP_CTRL_IN;
```

```
      FT120_CMD = ACK_SETUP;
      FT120_CMD = CLR_BUFFER;
      FT120_CMD = SEL_EP_CTRL_OUT;
      FT120_CMD = ACK_SETUP;
      FT120_CMD = CLR_BUFFER;

      // Clear appropriate flags
      CTRL_IN_EP_FULL = 0;
      SUT_Received    = 0;
    }

    if (CTRL_OUT_Received) {
      #ifdef BUSY_CNT
      BUSY |= 1;
      #endif
      CTRL_OUT_Received = 0;
      FT120_CMD = SEL_EP_CTRL_OUT;
      FT120_CMD = CLR_BUFFER;
    }
…….

/*****************************************************************************
 * Function:        void Process_SUT(void)
 * PreCondition:    USB Interrupt received from Setup Token
 * Input:           None
 * Output:          None
 * Overview:        This function is called when a Setup Token has been
 *                  received and the data will be contained in memory *
 *                  setup_pkt. This should be parsed and appropriate action
 *                  taken.
 * Note:            None
 *****************************************************************************/
void Process_SUT(void) {
  BYTE wIndex;

  switch (setup_pkt[1]) {

    //=================== GET STATUS ==============================
    case USB_REQUEST_CODE_GET_STATUS:

      // Check general protocol and Stall if incorrect
      if (USB_STATE == USB_DEFAULT) {
        Stall_CTRL();
      } else if (setup_pkt[6] != 2) {
        Stall_CTRL();
      } else if ((setup_pkt[2] | setup_pkt[3] | setup_pkt[7]) != 0) {
        Stall_CTRL();
      } else if (wIndex_matches_an_endpoint(setup_pkt[5], setup_pkt[4]) ==
0) {
        Stall_CTRL();

      // In ADDRESS state Stall any request for Interface or a non zero
           Endpoint
      } else if ((USB_STATE == USB_ADDRESS) &&
                 ((setup_pkt[0] == 0x81) || (setup_pkt[4] != 0))) {
        Stall_CTRL();

      // The request is a valid access
```

```
  } else {

    switch (setup_pkt[0]) {

      //-------- Recipient: Device -----------
      case 0x80:
        CTRL_BUFFER_x[0] = 0;         // remote_wakeup, self_powered
        CTRL_BUFFER_x[0] |= remote_wakeup;
        CTRL_BUFFER_x[1] = 0;

        ctrl_in_queue_size = 2;
        ctrl_in_queue_ptr = (__xdata BYTE *)&CTRL_BUFFER_x[0];

        break;

      //-------- Recipient: Interface --------
      case 0x81:
        CTRL_BUFFER_x[0] = 0;
        CTRL_BUFFER_x[1] = 0;

        ctrl_in_queue_size = 2;
        ctrl_in_queue_ptr = (__xdata BYTE *)&CTRL_BUFFER_x[0];
        break;

      //-------- Recipient: Endpoint ---------
      case 0x82:
        CTRL_BUFFER_x[1] = 0;
        wIndex  = (setup_pkt[4] & 0x0F);
        if (wIndex == 0x00) {
          CTRL_BUFFER_x[0] = 0;
        } else if ((endpoint_halt & wIndex) == wIndex) {
          CTRL_BUFFER_x[0] = 0x1;          // Halt
        } else {
          CTRL_BUFFER_x[0] = 0;
        }

        ctrl_in_queue_size = 2;
        ctrl_in_queue_ptr = (__xdata BYTE *)&CTRL_BUFFER_x[0];
        break;

      //-------- Recipient: Invalid ---------
      default:
        Stall_CTRL();
        break;
    }

  }
  break;

//====================== SET FEATURE ============================
case USB_REQUEST_CODE_SET_FEATURE:

  // Check general protocol and Stall if incorrect
  if (setup_pkt[2] > 2) {
    Stall_CTRL();
  ……..
//====================== CLEAR FEATURE ==========================
case USB_REQUEST_CODE_CLEAR_FEATURE:
```

```
      // Check general protocol and Stall if incorrect
      if (setup_pkt[2] > 2) {
        Stall_CTRL();
      } else if
..............
    //==================== GET CONFIGURATION ========================
    case USB_REQUEST_CODE_GET_CONFIGURATION:

      // Check general protocol and Stall if incorrect
      if (USB_STATE == USB_DEFAULT) {
        Stall_CTRL();
      } else if
............
    //==================== GET INTERFACE ============================
    case USB_REQUEST_CODE_GET_INTERFACE:

      // Check general protocol and Stall if incorrect
      if (USB_STATE != USB_CONFIGURED) {
        Stall_CTRL();
      } else if
..............

    //====================== SET INTERFACE ====================
    case USB_REQUEST_CODE_SET_INTERFACE:

      // Check general protocol and Stall if incorrect
      if (USB_STATE != USB_CONFIGURED) {
        Stall_CTRL();
      } else if
.........
    //==================== GET DESCRIPTOR ============================
    case USB_REQUEST_CODE_GET_DESCRIPTOR:

      // Check general protocol and Stall if incorrect
      if (setup_pkt[0] != 0x80) {
        Stall_CTRL();

      } else
...........
    //================ SET CONFIGURATION ========================
    case USB_REQUEST_CODE_SET_CONFIGURATION:

      // Check general protocol and Stall if incorrect
      if (USB_STATE == USB_DEFAULT) {
        Stall_CTRL();
      } else if
    .......

    //==================== SET ADDRESS ========================
    case USB_REQUEST_CODE_SET_ADDRESS:

      // Check general protocol and Stall if incorrect
      if (setup_pkt[0] != 0x00) {
        Stall_CTRL();
      } else if
...........
    //====================== DEFAULT ============================
    default:
```

```
      Stall_CTRL();
      break;
  }
}
```

4.3 Application Data Transfer

4.3.1 Read

Data sent from the USB host to the FT12x must be read by the embedded processor. The FT12x can interrupt the embedded processor that data is available or the embedded processor can poll the FT12x for data. If data is available it may be read into a buffer on the main processor for processing

```
//-------------------------------------------------------------------------
// EP2 OUT :
//
// When an OUT transfer occurs on EP 2 the data should be read out of
// the FT120 and into the OUT_BUFFER in memory.
//-------------------------------------------------------------------------
    if ((EP2_BUF_0_FULL) && (out_buffer_data_cnt < (OUT_BUFFER_SIZE - 64)))
    {
      #ifdef BUSY_CNT
      BUSY |= 1;
      #endif
      FT120_RD_OUT_2_BUFFER();

      EP2_BUF_0_FULL = 0;
      FT120_CMD = CLR_BUFFER;
    }

    if ((EP2_BUF_1_FULL) && (out_buffer_data_cnt < (OUT_BUFFER_SIZE - 64)))
      {
      #ifdef BUSY_CNT
      BUSY |= 1;
      #endif
      FT120_RD_OUT_2_BUFFER();

      EP2_BUF_1_FULL = 0;
      FT120_CMD = CLR_BUFFER;
      }
//-------------------------------------------------------------------------
// PERIPHERAL Tx :
//
// If the OUT_BUFFER has data in it and the peripheral is empty then
// send a data byte here. Update pointers and counters with wrap around
// if the end of buffer is reached.
//-------------------------------------------------------------------------
    if (peri_tx_empty && (out_buffer_data_cnt > 0))
    {
      #ifdef BUSY_CNT
      BUSY |= 1;
      #endif
      peri_tx_empty = 0;

      out_buffer_tail_ptr++;
      out_buffer_data_cnt--;
```

```
    if (out_buffer_tail_ptr == (__xdata BYTE *)(OUT_BUFFER_BASE_ADDR +
          OUT_BUFFER_SIZE))
    {
      out_buffer_tail_ptr = (__xdata BYTE *)OUT_BUFFER_BASE_ADDR;
    }
  }
```

4.3.2 Write

Data to be sent from the FT12x to the USB host is defined as a write in the embedded firmware. Provided the FT12x can accept more data (buffers are not full) then a write to the IN endpoint can occur at any time. It will be up to the USB host to provide the read request to get data out the USB port.

```
//------------------------------------------------------------------------
// EP1 IN :
//
// EP 1 IN buffer should be loaded with data when it is empty and there are
// either at least 62 bytes in the buffer, or the latency timer has
// expired. Example based on FT232X.
//------------------------------------------------------------------------
  if (USB_STATE == USB_CONFIGURED) {
    if (!EP1_IN_FULL && ((in_buffer_data_cnt >= 62) || latency_expired))
    {
      #ifdef BUSY_CNT
      BUSY |= 1;
      #endif
      EP1_IN_FULL = 1;
      FT120_Load_EP_1_IN();
    }
  }
```

5 Debugging FT12 Code

Realistically, the only way to debug the FT12 code being developed is to use a USB bus analyser to monitor the traffic on the bus. Such tools can be either purely software based, such as USB Monitor from HHD software or USB Sniffer, both of which can be downloaded from the internet.

Hardware based tools basically plug into the USB path between the host and the device to sniff traffic and examples of this include the Beagle Analyser from Total Phase or the USB Tracker from Ellisys.

There will be other tools out there.

An example of a trace from the Ellisys USB Tracker is shown in Figure 2.

The screen shot has captured the enumeration of a USB keyboard.

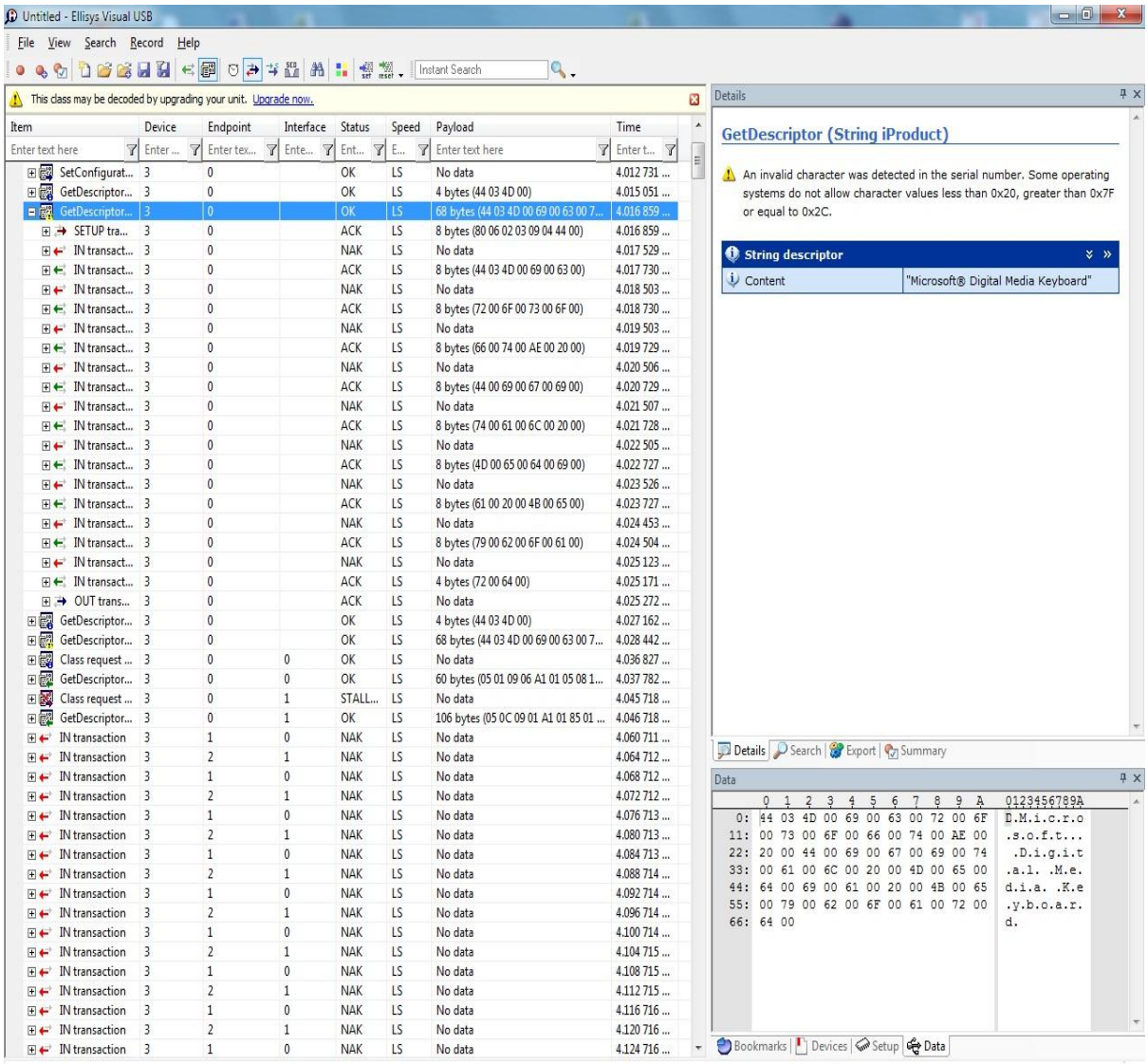

Figure 2 : USB Trace of a HID Class Device Enumerating

From the trace the developer can determine what traffic is on the bus (i.e. what requests the USB host made and the response the FT12x gave).

Anything with an arrow pointing left to right in the left hand pane is traffic from the host to the FT12x.

Anything with an arrow from the right to the left in the left hand pane is traffic generated by the FT12x firmware and sent to the host.

The Get Descriptor call from the host highlighted in the screenshot is asking for the device product string. The IN transfers that are marked NAK basically highlight the device is not ready.

The other IN packets translate from HEX to ASCII as "Microsoft® Digital Media Keyboard"

Many of the USB sniffer tools provide ASCII, hex and binary output windows to help with decoding the traffic.

6 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

FT120X Data Sheet

FT121 Data Sheet

FT122 Data Sheet

TN_110 What is USB

TN_113 Simplified Description of USB Enumeration

TN_116 USB Data Packet Structure

USB Implementers Forum Documents/Specifications.

Standard USB Device Classes

USB Specification

Other references

USB in a Nutshell

Sample code

www.ftdichip.com/support/software_examples/FT12.htm

Acronyms and Abbreviations

Terms	Description
DMA	Direct Memory Access
PID	Product Identifier
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
USB-IF	USB Implementers Forum (www.usb.org)
VID	Vendor Identifier

Appendix B – List of Tables & Figures

List of Tables

No table of figures entries found.

List of Figures

Appendix C – Revision History

Document Title: AN_223 How To Use The FT12 Series Devices
Document Reference No.: FT_000743
Clearance No.: FTDI# 315
Product Page: http://www.ftdichip.com/FTProducts.htm
Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial Release	2012-09-25