Application Note

AN_348

FT51A FT800 Sensors Sample

Version 1.1

Issue Date: 2015-11-26

This document provides a guide for using the FT51A development environment to display sensor readings on an FT800 display.

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

This application note documents an example firmware project for the FT51A. The source code is available in the "examples\AN_348_FT51A_FT800_Sensors_Sample" folder of the FT51A Software Development Kit.

1.1 Overview

The readings from sensors on the FT51A EVM board are displayed on an FT800 display module. The display shows heart rate, temperature and force readings both numerically and graphically.

The hardware modules required to use the application note are a VM800B module (FT800 and display) and a FT51A EVM module. The sensors are built-in to the FT51A EVM module.

The FT800 on the VM800B module is connected to the SPI Master interface of the FT51A on pins DIO_2 (Slave Select 1), DIO_3 (SPI MOSI), DIO_4 (SPI MISO) and DIO_5 (SPI SCLK).

1.2 Features

The FT800 Sensors Sample has the following features:

- Open source firmware.
- Reads data from a temperature sensor (ADT7310) using the SPI Master interface.
- Converts the analogue voltage from a force sensor to a digital reading.
- Detects transitions on an analogue voltage input to make a simple heart rate monitor.
- Displays output on FT800 display via the SPI Master interface.

1.3 Limitations

The firmware does not implement a USB device to transmit data to a host. It does not implement Device Firmware Update (DFU) on the USB.

1.4 Scope

The guide is intended for developers who are creating applications, extending FTDI provided applications or implementing example applications for the FT51A.

In the reference of the FT51A, an "application" refers to firmware that runs on the FT51A; "libraries" are source code provided by FTDI to help user, access specific hardware features of the chip.

The FT51A Tools are currently only available for Microsoft Windows platform and are tested on Windows 7 and Windows 8.1.

2 Firmware Overview

The sensors firmware reads inputs from the sensors on the FT51A EVM board and converts these into values to display. It employs similar methods to those used in AN_347 FT51A Test and Measurement Sample Application Note.

2.1 FT800 Display

The VM800B module interface for displaying the graphics and text is configured over the SPI Master interface. Commands and data to send are stored in a buffer in RAM and sent in batches to the FT800 device on the VM800B module at regular intervals. The SPI Master interface is shared with the temperature sensor – the Slave Select lines of the SPI Master interface are used to address the FT800 or the temperature sensor.

The firmware includes an FT800 abstraction layer providing functions which are used to perform actions on the FT800. This layer is in the `ft_gpu` directory of the sample source code. The definition of `FT51A_PLATFORM` and `FT51A_PLATFORM_SPI` in the file `FT_Platform.h` are used to enable the FT51A abstraction in the libraries and use SPI for the communications. The same library can be used for other FTDI microcontrollers.

2.2 FT51A Libraries

The firmware uses the SPI Master library, general config library and the IOMUX library. The IOMUX library is used in the example code to set the output characteristics of the SPI Master interface. A code module for the ADT7310 temperature sensor (via the SPI Master library) is included.

The firmware is designed for the FT51A EVM module and may be extended to use the LCD for displaying some information. If so, then it will need the I2C Master library added.

3 FT800 Sensors Firmware

The firmware included in the example code demonstrates reading sensors and displaying on an FT800 controlled display.

The firmware is designed for the FT51A EVM module. It will use the force sensor, temperature sensor and heart rate sensor. The force and heart rate sensors use an analogue voltage input which is converted using the ADC features. The temperature sensor uses an ADT7310 temperature sensor which is connected to the FT51A *via* an SPI bus.

All sensor readings and screen drawing triggers are initiated by timers in the `Force_Heartrate_Temperature` function in `ft800_demo_board.c`. This function contains a while loop that does not exit.

3.1 Timer

A timer is used to provide delays and time measurements for implementing polling intervals.

The first timer `ms_timer` is used to create general purpose delays, for instance when resetting the temperature sensor.

Each of the three sensors has a timer `timer_force`, `timer_pulse` and `timer_temp`. These timers operate independently. There is also a timer to update the FT800 display called `timer_display`.

```
void ms_timer_interrupt(const uint8_t flags)
{
    (void) flags; // Flags not currently used
    if (ms_timer)
    {
        ms_timer--;
    }
    if (timer_display)
    {
        timer_display--;
    }
    if (timer_force)
    {
        timer_force--;
    }
    if (timer_pulse)
    {
        timer_pulse--;
    }
    if (timer_temp)
    {
        timer_temp--;
    }
    // Reload the timer
    TH0 = MSB(MICROSECONDS_TO_TIMER_TICKS(1000));
    TL0 = LSB(MICROSECONDS_TO_TIMER_TICKS(1000));
}

void ms_timer_initialise(void)
{
    // Register our own handler for interrupts from Timer 0
    interrupts_register(ms_timer_interrupt, interrupts_timer0);

    // Timer0 is controlled by TMOD bits 0 to 3, and TCON bits 4 to 5.
    TMOD &= 0xF0; // Clear Timer0 bits
    TMOD |= 0x01; // Put Timer0 in mode 1 (16 bit)
    // Set the count-up value so that it rolls over to 0 after 1 millisecond.
```

```
    TH0 = MSB(MICROSECONDS_TO_TIMER_TICKS(1000));
    TL0 = LSB(MICROSECONDS_TO_TIMER_TICKS(1000));
    TCON &= 0xCF; // Clear Timer0's Overflow and Run flags
    TCON |= 0x10; // Start Timer0 (set its Run flag)
}
```

3.2 Writing to FT800

Commands to the FT800 can either be sent discretely to registers or to internal RAM in batches when the display is being constantly updated by the firmware.

Both methods will enable the SPI Master to talk to the FT800 with the Slave Select 1 (SS1) line going high before sending commands. The next stage is to send an address of the register or area of RAM to be addressed in the FT800. The data for the register or RAM follows before SS1 is de-asserted to signal the end of the transfer.

A discrete register change is performed using the `FT800_Write08` or `FT800_Write16` functions in the file `ft800_demo_board.c`. These functions call the hardware abstraction layer functions to turn on slave select and send the register commands before turning off slave select.

Batches of commands will be stored in a RAM buffer on the FT51A called `Ft_CmdBuffer` (the size and therefore the end offset of this buffer are held in the variable `Ft_CmdBuffer_Index`). Helper functions `FT800_CmdBuffer_AddCmd` and `FT800_CmdBuffer_AddStr` functions are available to abstract the operation of the buffer.

When the batch is complete the commands are sent to the FT800 RAM using the `FT800_CmdBuffer_Send` function. The size of `Ft_CmdBuffer` is 4 kB.

Instructions for an entire screen update can be stored in the buffer before being transmitted to the FT800 at a fixed interval. The update is triggered when the `timer_display` value reaches zero at which point all current sensor readings are formatted and the commands to create the screen update are generated.

This document does not deal with the FT800 commands used to render the graphics on the display.

3.3 Sensor Reading Acquisition

Each sensor is read from the firmware's main loop, in the `Force_Heartrate_Temperature` function.

3.3.1 Force Sensor

When the `timer_force` reaches zero, a new ADC is triggered and the timer restarted. The ADC conversion will take place in the background and an interrupt generated when it is complete.

```
        if (timer_force == 0)
        {
            IO_REG_INTERRUPTS_WRITE(IO_CELL_SAMPLE_0_7_1, MASK_IO_CELL_0_SAMPLE);
            timer_force = FORCE_TIMER;
        }
```

The ADC interrupt handler (section below) will update the `sampleValueForce` and `sampleReadyForce` variables when the reading is complete.

```
    IO_REG_INTERRUPTS_READ(IO_CELL_INT_0_1, interrupt);
    // FSR Output
    if (interrupt & MASK_SD_CELL_0_INT)
    {
        IO_REG_INTERRUPTS_READ(IO_CELL_0_ADC_DATA_L_1, sample_l);
```

```
        IO_REG_INTERRUPTS_READ(IO_CELL_0_ADC_DATA_U_1, sample_h);

        sampleValueForce = ((sample_h << 8) | sample_l);
        sampleReadyForce = TRUE;

        // Clear ADC interrupt register bit
        IO_REG_INTERRUPTS_WRITE(IO_CELL_INT_0_1, MASK_SD_CELL_0_INT);
    }
```

The main loop will arrive back at the force sensor code and update the force reading when the `sampleReadyForce` flag is set by the interrupt handler.

```
        if (sampleReadyForce)
        {
            sampleReadyForce = FALSE;
            if (sampleValueForce > 900)
            {
                sampleValueForce = 900;
            }
            force = sampleValueForce;
        }
```

The `timer_force` will continue to run until the next sample is due.

3.3.1 Heart Rate Sensor

When the `timer_pulse` reaches zero, a new ADC is triggered and the timer restarted. The ADC conversion will take place in the background and an interrupt generated when it is complete.

```
        if (timer_pulse == 0)
        {
            IO_REG_INTERRUPTS_WRITE(IO_CELL_SAMPLE_8_15_1, MASK_IO_CELL_10_SAMPLE);
            timer_pulse = PULSE_TIMER;
        }
```

The ADC interrupt handler will update the `sampleValuePulse` and `sampleReadyPulse` variables when the reading is complete.

```
    IO_REG_INTERRUPTS_READ(IO_CELL_INT_1_1, interrupt);
    // Pulse Rate
    if (interrupt & MASK_SD_CELL_10_INT)
    {
        IO_REG_INTERRUPTS_READ(IO_CELL_10_ADC_DATA_L_1, sample_l);
        IO_REG_INTERRUPTS_READ(IO_CELL_10_ADC_DATA_U_1, sample_h);

        sampleValuePulse = ((sample_h << 8) | sample_l);
        sampleReadyPulse = TRUE;

        // Clear ADC interrupt register bit
        IO_REG_INTERRUPTS_WRITE(IO_CELL_INT_1_1, MASK_SD_CELL_10_INT);
    }
```

The main loop will arrive back at the heart rate sensor code and update the heart beat reading when the `sampleReadyPulse` flag is set by the interrupt handler.

```
        if (timer_pulse == 0)
        {
            IO_REG_INTERRUPTS_WRITE(IO_CELL_SAMPLE_8_15_1, MASK_IO_CELL_10_SAMPLE);
            timer_pulse = PULSE_TIMER;
```

```
        }
        if (sampleReadyPulse)
        {
            sampleReadyPulse = FALSE;
            pulseSamples[pulseIndexer] = sampleValuePulse;

            // Keep track of how many samples in a block are greater than threshold ...
            if (sampleValuePulse >= PULSE_THRESHOLD)
            {
                pulseCounter++;

                // Mark as a heart beat if number of samples in a block exceeds set value
                if (pulseCounter == PULSE_IS_BEAT)
                {
                    // Set top bit (bits 0-9 contain the actual sensor value)
                    pulseSamples[pulseIndexer] |= 0x8000;
                    // Start looking for the next heart beat ...
                    pulseCounter                = 0;
                    LED_HR_ON();
                }
            }
            else
            {
                LED_HR_OFF();
                // Deal with gaps (value < threshold) when counting of samples in a block
                if (pulseCounter > 0)
                {
                    pulseCounter--;
                }
            }
            pulseIndexer = (++pulseIndexer) % PULSE_SAMPLES;
        }
```

The loop fills a buffer with `PULSE_SAMPLES` number of samples with the top bit being set if a sequence of consecutive samples is above a threshold. The size of this buffer is calculated to hold 15 seconds worth of samples.

This buffer is parsed in the `RenderPulse` function where the measured heart rate is calculated and graphics are created to show a 'paper chart' history of detected pulses.

3.3.2 Temperature Sensor

An SPI bus is used to measure the temperature from an ADT7310 sensor connected to the SPI Master interface. When the `timer_temp` reaches zero the timer is reset and an SPI Master read is performed to the ADT3710.

```
        if (timer_temp == 0)
        {
            // Read the temperature from the SPI Master
            sampleValueTemp = temperature_read();
            timer_temp = TEMP_TIMER;
        }
```

The temperature is returned in units of 0.01 °C.

The temperature sensor library will assert Slave Select 0 (SS0) to a logic high to enable the ADT7310 device during SPI Master Transfers.

4 Possible Improvements

This implementation of the heart rate sensor has a simpler method of calculation than the method used in AN_347 FT51A Test and Measurement Sample Application Note.

5 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 – Tigard, Oregon, USA

Future Technology Devices International Limited (USA)
7130 SW Fir Loop
Tigard, OR 97223-8160
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

Distributor and Sales Representatives

Please visit the Sales Network page of the FTDI Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Appendix A – References

Document References

FTDI MCU web page: http://www.ftdichip.com/MCU.html

USB Test and Measurement Class specification:
http://www.usb.org/developers/docs/devclass_docs/USBTMC_1_006a.zip

IVI Foundation: http://www.ivifoundation.org/

SCPI specification: http://www.ivifoundation.org/docs/scpi-99.pdf

USB Device Firmware Update Class specification:
http://www.usb.org/developers/docs/devclass_docs/DFU_1.1.pdf

Acronyms and Abbreviations

Terms	Description
HID	Human Interface Device
MTP	Multiple Time Program – non-volatile memory used to store program code on the FT51A.
USB	Universal Serial Bus
USB-IF	USB Implementers Forum

Appendix B – Revision History

Document Title: AN_348 FT51A FT800 Sensors Sample

Document Reference No.: FT_001127

Clearance No.: FTDI# 431

Product Page: http://www.ftdichip.com/FTProducts.htm

Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial Release	2014-12-12
1.1	Update FT51 references to FT51A	2015-11-26