This document describes how to use FTDI’s UMFT51AA module as a replacement for a classic 40-pin DIP 8051-based microcontroller.
Table of Contents
1 Introduction ........................................................................................................... 2
2 Pin Configuration .................................................................................................. 3
3 Special-Function Registers (SFRs) ........................................................................ 4
   3.1 Peripheral Access .......................................................................................... 5
4 Power consumption ............................................................................................... 7
5 Program Security ................................................................................................... 8
6 Compiler differences ............................................................................................. 9
7 Interrupt Vectors .................................................................................................. 10
8 Electrical characteristics ....................................................................................... 11
9 Timer 2 .................................................................................................................. 12
10 Clock frequency and timing ................................................................................. 13
11 Sample code ......................................................................................................... 14
12 Initial firmware ..................................................................................................... 16
13 Contact Information ............................................................................................ 17
Appendix A – References ......................................................................................... 18
   Document References .......................................................................................... 18
   Acronyms and Abbreviations ................................................................................. 18
Appendix B – List of Tables & Figures .................................................................... 19
   List of Figures ....................................................................................................... 19
Appendix C – Revision History ................................................................................. 20
1 Introduction

Since Intel discontinued its 80C51, many other manufacturers have released their own 8051-compatible microcontrollers in the same 40-pin dual-inline package, with a matching pin configuration. Examples include the Philips P80C51, the Atmel 89S8253 and the Maxim DS80C320. FTDI’s UMFT51AA module (a small circuit board featuring an enhanced 8051-compatible microcontroller, the FT51A) is pin-compatible – and generally code-compatible – with such chips.

This document compares features of the FT51A with such chips, and describes any changes which might be required to make existing firmware run correctly on the FT51A.
2 Pin Configuration

The FT51A’s Input-Output Multiplexer (IOMUX) allows you to map any signal to any pin, but as shown in Figure 1, the default mapping routes the UMFT51AA’s Input-Output ports to the same pins as the classic 8051 and its derivatives.

<table>
<thead>
<tr>
<th>Classic 8051 derivative.</th>
<th>UMFT51AA</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.7</td>
<td>VCC</td>
</tr>
<tr>
<td>P1.6</td>
<td>P0.0</td>
</tr>
<tr>
<td>P1.5</td>
<td>P0.1</td>
</tr>
<tr>
<td>P1.4</td>
<td>P0.2</td>
</tr>
<tr>
<td>P1.3</td>
<td>P0.3</td>
</tr>
<tr>
<td>P1.2</td>
<td>P0.4</td>
</tr>
<tr>
<td>P1.1</td>
<td>P0.5</td>
</tr>
<tr>
<td>P1.0</td>
<td>P0.6</td>
</tr>
<tr>
<td>RST</td>
<td>P0.7</td>
</tr>
<tr>
<td>(RxD) P3.0</td>
<td>EA/VPP</td>
</tr>
<tr>
<td>(TxD) P3.1</td>
<td>ALE/PROG</td>
</tr>
<tr>
<td>(INT 0) P3.2</td>
<td>PSEN</td>
</tr>
<tr>
<td>(INT 1) P3.3</td>
<td>P2.7</td>
</tr>
<tr>
<td>(T0) P3.4</td>
<td>P2.6</td>
</tr>
<tr>
<td>(T1) P3.5</td>
<td>P2.5</td>
</tr>
<tr>
<td>P3.6</td>
<td>P2.4</td>
</tr>
<tr>
<td>P3.7</td>
<td>P2.3</td>
</tr>
<tr>
<td>XTAL2</td>
<td>P2.2</td>
</tr>
<tr>
<td>XTAL1</td>
<td>P2.1</td>
</tr>
<tr>
<td>GND</td>
<td>P2.0</td>
</tr>
</tbody>
</table>

Connections shown in red are unavailable or unnecessary.

Other 8051 compatible devices typically add alternative functionality to some of the pins: for example, Port 0 often doubles as a combined address/data bus for external memory, and pins on Port 3 receive and transmit serial data. The FT51A’s versatility allows the UMFT51AA to be configured similarly, with the following exceptions.

- The FT51A has 8 kB of ‘external’ data RAM and 16 kB of program memory built-in, so doesn’t need an address/data bus; pins for associated signals (such as ALE or PROG) are not connected to anything on the FT51A. The UMFT51AA is typically programmed through a dedicated connector, or with the USB Device Firmware Upgrade (DFU) standard.
- The FT51A uses interrupts 0 and 1 for its extended range of on-chip peripherals (enhanced UART, SPI Master and Slave etc.) so pins 2 and 3 of Port 3 are not available as external interrupt sources.
- The FT51A does not support counting external events on timers 0 and 1 so pins 3 and 4 of Port 3 cannot be used for this purpose.
- The FT51A has its own internal clock so there is no need to connect a crystal.
- Since the FT51A can route any signal to any pin, to get RxD and TxD on pins 2 and 3 of Port 3, this must be explicitly specified in software.
### Special-Function Registers (SFRs)

Like the 8051 and its derivatives, the FT51A has an area of RAM with registers to control the core and the on-chip peripherals. The FT51A has standard 8051 SFRs in the traditional locations (shown in green), plus extra SFRs to directly control the I²C Master, I²C Slave and USB Slave. All other on-chip peripherals are controlled indirectly through ten configurable channels, each comprising three SFRs (see section 3.1).

<table>
<thead>
<tr>
<th>+0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
<th>+6</th>
<th>+7</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>P0</td>
<td>SP</td>
<td>DPL0</td>
<td>DPH0</td>
<td>DPL1</td>
<td>DPH1</td>
<td>DPS</td>
</tr>
<tr>
<td>88</td>
<td>TCON</td>
<td>TMOD</td>
<td>TL0</td>
<td>TL1</td>
<td>TH0</td>
<td>TH1</td>
<td>CKCON</td>
</tr>
<tr>
<td>90</td>
<td>P1</td>
<td>EIF</td>
<td></td>
<td></td>
<td>IO_DATA_9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>SCON0</td>
<td>SBUF0</td>
<td>IO_ADDR_0_H</td>
<td>IO_ADDR_0_L</td>
<td>IO_DATA_0</td>
<td>IO_ADDR_1_H</td>
<td>IO_ADDR_1_L</td>
</tr>
<tr>
<td>A0</td>
<td>P2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A8</td>
<td>IE</td>
<td>IO_ADDR_2_H</td>
<td>IO_ADDR_2_L</td>
<td>IO_DATA_2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B0</td>
<td>P3</td>
<td>IO_ADDR_3_H</td>
<td>IO_ADDR_3_L</td>
<td>IO_DATA_3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B8</td>
<td>IP</td>
<td>IO_ADDR_4_H</td>
<td>IO_ADDR_4_L</td>
<td>IO_DATA_4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>T2CON</td>
<td>T2IF</td>
<td>RCAP2L</td>
<td>RCAP2H</td>
<td>TL2</td>
<td>TH2</td>
<td></td>
</tr>
<tr>
<td>D0</td>
<td>PSW</td>
<td>IO_ADDR_5_H</td>
<td>IO_ADDR_5_L</td>
<td>IO_DATA_5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D8</td>
<td>IO_ADDR_6_H</td>
<td>IO_ADDR_6_L</td>
<td>IO_DATA_6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E0</td>
<td>ACC</td>
<td>IO_ADDR_7_H</td>
<td>IO_ADDR_7_L</td>
<td>IO_DATA_7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E8</td>
<td>EIE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F0</td>
<td>B</td>
<td>I2CSOA</td>
<td>I2CSCR</td>
<td>I2CSBUF</td>
<td>I2CMSA</td>
<td>I2CMCR</td>
<td>I2CMBUF</td>
</tr>
<tr>
<td>F8</td>
<td>EIP</td>
<td>IO_ADDR_8_H</td>
<td>IO_ADDR_8_L</td>
<td>IO_DATA_8</td>
<td>FT122_CMD</td>
<td>FT122_DATA</td>
<td>IO_ADDR_9_H</td>
</tr>
</tbody>
</table>

#### Standard 8051 register
- Register to control an enhancement which may be found on other 8051 derivatives
- Register to control directly an FT51A-specific peripheral
- Register to control indirectly an FT51A-specific peripheral

As well as the standard peripherals, the FT51A has some of the features of enhanced 8051 derivatives such as a third timer (registers C8 to CD) and a second data pointer (registers 84 to 86). The following table compares some of the relevant registers with those found on the Atmel AT89S8253.
### 3.1 Peripheral Access

The table of SFRs refers to direct and indirect access to on-chip peripherals. Timer's 0–2, the basic UART, the I²C and the FT122 compatible USB module are all controlled by dedicated SFRs, accessed directly in code as shown in these examples.

```c
TMOD &= 0xF0; // Clear Timer0 bits
TMOD |= 0x01; // Put Timer0 in mode 1 (16 bit)
TCON &= 0xCF; // Clear Timer0's Overflow and Run flags
TCON |= 0x10; // Start Timer0 (set its Run flag)

SCON &= ~TX_COMPLETE; // Clear UART transmit-complete flag
SBUF = c; // Put byte into transmit buffer
while (!SCON & TX_COMPLETE); // Wait for all bits to be sent

FT122_CMD = SELECT_ENDPOINT + 0; // Select USB endpoint 0
if (FT122_DATA & SELECT_BUFFER_FULL) // Endpoint status appears in FT122_DATA
...

I2CMBUF = data; // Prepare to send byte from I2C Master
I2CMCR = I2C_FLAGS_RUN; // Start sending byte
do {
    status = I2CMCR;
} while (status & I2C_STATUS_BUSY); // Wait until byte sent
```

All other peripherals (such as Timers A–D, SPI and ADC) are accessed indirectly using one of the ten Input-Output channels, each consisting of a 16-bit address (split into high and low bytes, e.g. IO_ADDR_0_H and IO_ADDR_0_L) and 8 bits of read/write data (e.g. IO_DATA_0). This allows firmware to map up to ten more peripheral control registers into the SFR space at any time. The following example maps three SPI Master Registers into channels 5, 6 and 7.

<table>
<thead>
<tr>
<th>Register address</th>
<th>FT51A name and function</th>
<th>AT89S8253 name and function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x86</td>
<td>DPS</td>
<td>SPDR</td>
</tr>
<tr>
<td></td>
<td>Select active data pointer</td>
<td>SPI data</td>
</tr>
<tr>
<td>0x8E</td>
<td>CKCON</td>
<td>AUXR</td>
</tr>
<tr>
<td></td>
<td>Modify tick duration for timer 0, 1 or 2.</td>
<td>ALE and Power-down configuration</td>
</tr>
<tr>
<td>0x8F</td>
<td>Not used</td>
<td>CLKREG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External crystal clock generation</td>
</tr>
<tr>
<td>0x91</td>
<td>EIF</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>Extended interrupt flags for I²C Master and Slave</td>
<td></td>
</tr>
<tr>
<td>0x96</td>
<td>Not used</td>
<td>EECON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EEPROM control, and selection of active data pointer</td>
</tr>
<tr>
<td>0xC9</td>
<td>T2IF</td>
<td>T2MOD</td>
</tr>
<tr>
<td></td>
<td>Timer 2 interrupt flags</td>
<td>Timer 2 mode</td>
</tr>
<tr>
<td>0xE8</td>
<td>EIE</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>Extended interrupt enable for I²C Master and Slave</td>
<td></td>
</tr>
<tr>
<td>0xE9</td>
<td>STATUS</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>Allows firmware to check if safe to enter low-power mode</td>
<td></td>
</tr>
<tr>
<td>0xF8</td>
<td>EIP</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>Extended interrupt priority for I²C Master and Slave</td>
<td></td>
</tr>
</tbody>
</table>
void SPIM_transceive(uint8_t  *src,  
   uint8_t  *dest,  
   uint16_t  length)  
{
  // Map SPI Master Transmit Buffer register to IO Channel 5
  IO_ADDR_5_H = MSB(SPI_MASTER_TX_DATA);
  IO_ADDR_5_L = LSB(SPI_MASTER_TX_DATA);

  // Map SPI Master Receive Buffer register to IO Channel 6
  IO_ADDR_6_H = MSB(SPI_MASTER_RX_DATA);
  IO_ADDR_6_L = LSB(SPI_MASTER_RX_DATA);

  // Map SPI Master Interrupt Flags register to IO Channel 7
  IO_ADDR_7_H = MSB(SPI_MASTER_INT);
  IO_ADDR_7_L = LSB(SPI_MASTER_INT);

  while (length)  
  {
    IO_DATA_7 = 0; // Clear interrupt flags
    IO_DATA_5 = *src++; // Send one byte
    while (!(IO_DATA_7 & MASK_SPI_TX_DONE)); // Wait for byte to be sent
    IO_DATA_6 = *dest++; // Store the received byte
    length--;
  }
}

By saving and restoring the IO address SFRs (or assigning them for every read and write), firmware may map more than ten peripheral-control registers. FTDI provide macros (similar to the one shown below) for this latter approach, dedicating one channel for general access and a second channel for access at interrupt level.

#define IO_REG_GENERAL_WRITE(address, data)  
   do  
   {  
     IO_ADDR_0_H = (uint16_t)(address) >> 8;  
     IO_ADDR_0_L = (uint8_t)(address);  
     IO_DATA_0 = (data);  
   }  
   while (0)  
   ...

IO_REG_GENERAL_WRITE(SPI_MASTER_TX_DATA, *src++);

See lib\inc\ft51_io_registers.h in the FT51A Software Development Kit for macros to read and write a byte, write a word, and set or clear bits.
4 Power consumption

The FT51A supports the original 8051’s two low-power modes as follows.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Software</td>
<td>Interrupt or hardware reset</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Power Management. Divides system clock by 256. If switchback enabled (bit 2 of PCON), returns to full speed when certain peripherals interrupt, or serial bit received.</td>
</tr>
<tr>
<td>Power Down</td>
<td>Software</td>
<td>Hardware reset</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Stop. Stops system clock</td>
</tr>
</tbody>
</table>

Other 8051 derivatives (such as the Maxim DS80C320) can be woken from Power-Down mode by an external interrupt. The FT51A does not support direct external interrupts but external devices connected to on-chip peripherals can generate interrupts. Also, the FT51A’s on-chip USB function can support Suspend, Resume and Remote Wakeup.
5 Program Security

The original 8051 has a feature to lock or encrypt program memory, to protect intellectual property. The FT51A does not support encryption but the internal MTP program memory may be protected from reads or writes using the security settings register TOP_SECURITY_LEVEL.

```c
// Lock FT51A device to prevent read or write access to the MTP
IO_REG_GENERAL_WRITE(TOP_SECURITY_LEVEL, MASK_SECURITY_LEVEL);
```

6 Compiler differences

Currently, FTDI supports only the SDCC tool chain. Other compilers, e.g. MikroC, Keil, use different C language extensions for SFR and sbit declarations, and for annotating ISRs, as shown in the following table.

<table>
<thead>
<tr>
<th>Keil</th>
<th>SDCC</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void timer0_ISR(void) interrupt 1</code></td>
<td><code>void timer0_ISR(void) __interrupt (1)</code></td>
</tr>
<tr>
<td>sfr P0 = 0x80;</td>
<td>__sfr __at (0x80) P0;</td>
</tr>
<tr>
<td>sbit P0_0 = 0x80;</td>
<td>__sbit __at (0x80) P0_0;</td>
</tr>
<tr>
<td>sbit P0_2 = P0^2;</td>
<td>Not supported. Use numeric form.</td>
</tr>
<tr>
<td>unsigned char xdata my_var;</td>
<td>__xdata unsigned char my_var;</td>
</tr>
</tbody>
</table>

See the SDCC documentation (http://sdcc.sourceforge.net/doc/sdccman.pdf) for a full list of C extensions.
7 Interrupt Vectors

The FT51A contains an 8051-compatible core with the usual peripherals (timers etc.) plus some enhanced peripherals. These extra peripherals occupy the interrupts that are normally reserved for ‘external’ signals.

<table>
<thead>
<tr>
<th>Interrupt number</th>
<th>FT51A Interrupt</th>
<th>AT89S8253 interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DMA</td>
<td>External 0</td>
</tr>
<tr>
<td>1</td>
<td>Timer 0</td>
<td>Timer/Counter 0</td>
</tr>
<tr>
<td>2</td>
<td>SPI, Extended UART, etc.</td>
<td>External 1</td>
</tr>
<tr>
<td>3</td>
<td>Timer 1</td>
<td>Timer/Counter 1</td>
</tr>
<tr>
<td>4</td>
<td>Serial port (basic)</td>
<td>Serial port, SPI</td>
</tr>
<tr>
<td>5</td>
<td>Timer 2</td>
<td>Timer 2</td>
</tr>
<tr>
<td>13</td>
<td>I²C Master</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>I²C Slave</td>
<td></td>
</tr>
</tbody>
</table>

The FT51A’s Watchdog timer does not generate an interrupt. Instead, it directly resets the processor but sets a flag which start-up code can detect.

```c
#include "ft51_io_registers.h"
#include "registers/ft51_timer_registers.h"

... uint8_t data;
...

IO_REG_GENERAL_READ(TIMER_CONTROL_2, data);

if (data & MASK_WDG_INT)
{
    // This is a restart after a watchdog reset.
}
else
{
    // This is a normal restart.
}
```
8 Electrical characteristics

Many traditional 8051 derivatives have GPIO ports which operate with a range of 0 to 5V. The FT51A analog-capable IO ports (0 and 2) can tolerate inputs up to 5V, but the digital-only ports (1 and 3) can only tolerate inputs up to 3.3V. No FT51A pad (analog or digital) can drive out 5V: the maximum is 3.3V. The FT51A will be damaged if these restrictions are not followed.
9 Timer 2

In addition to the 8051’s Timer 0 and Timer 1, the 8052 offered a third timer, Timer 2. The FT51A also offers this third timer, like many of the 8051 derivatives. Its operation is similar to the other 8051 timers, with the following differences.

- There is no mode-selection register; the mode is 16-bit auto-reload.
- The interrupt flag is not automatically cleared when the ISR is executed; it must be cleared explicitly by software.
- There is a pre-scale option to make the timer count at half-speed.

Here is an example program to toggle an IO pin using Timer 2 with its maximum interval (counting from 0 to 65535 at half speed).

```c
void main(void)
{
    P2_0 = 1; // Start with LED off
    T2CON = 0; // Initialise Timer2

    // Maximum (65536) count-up before overflow.
    RCAP2H = 0;
    RCAP2L = 0;
    T2 PRESCALE = 1; // Timer ticks at half-speed.
    T2 RUN = 1; // Start Timer2

    for ( ; ; ) // Infinite loop
    {
        if (T2IF & T2_OVERFLOW)
        {
            P2_0 = !P2_0; // Toggle bit 0 of Port 2
            T2IF &~ T2_OVERFLOW; // clear interrupt flag
        }
    }
}
```

The definitions for the above program are as follows.

```c
__sbit__at(0xA0) P2_0; // Bit 0 of Port 2
__sfr__at(0xC8) T2CON; // Timer 2 control register
__sfr__at(0xC9) T2IF; // Timer 2 interrupt flags
__sfr__at(0xCA) RCAP2L; // Timer 2 initial value (low byte)
__sfr__at(0xCB) RCAP2H; // Timer 2 initial value (high byte)
__sbit__at(0xC8) T2_RUN; // 1: running; 0: not running
__sbit__at(0xCF) T2 PRESCALE; // 1: tick = clock/24; 0: tick = clock/12

#define T2_OVERFLOW 1 // Bit 0 of T2IF indicates overflow
```
10 Clock frequency and timing

The FT51A can operate at 12, 24 or 48 MHz. Since 48 MHz is the default and is typically faster than other 8051 derivatives, existing code may need to be altered. Also, the FT51A core executes most instructions in a single clock period, rather than a traditional 8051's machine cycle which was 12 clock periods.

Any delay implemented as a C empty loop will need to be recalibrated for the FT51A and for the SDCC compiler. For example,

```c
for (i = 0; i < 45678; i++)
    // Do nothing
```

might become:

```c
for (i = 0; i < 1234; i++)
    // Do nothing
```

The same applies to a delay implemented in assembly. For example, the following code toggles a GPIO line every millisecond on an FT51A at 48 MHz, but every 37 milliseconds on an AT89S8253 at 10 MHz.

```assembly
__sbit __at (0xA0) P2_0; // Bit 0 of Port 2
void main(void)
{
    for (; ; ) // Infinite loop
    {
        __asm
            mov r0, #31; // Outer loop iterations
        00001$: mov r1, #249; // Inner loop iterations
        00002$: nop
        nop
djnz r1, 00002$
djnz r0, 00001$
cpl _P2_0 ; // Toggle bit 0 of Port 2
    __endasm;
    }
}
```

For compatibility with existing designs, Timers 0, 1 and 2 do ‘tick’ every 12 clock periods, so the formula to convert microseconds to timer ticks is:

\[
timer \text{ ticks} = \frac{\text{time (\mu s)} \times \text{clock frequency (MHz)}}{12}
\]

This formula applies to the FT51A as well as the 8051 and its derivatives, so carefully-written timer code might not need to be changed. For an example, see Section 11.
11 Sample code

FTDI have ran this sample on both an Atmel AT89S8253 and a UMFT51AA module, connected to the mikroBoard for 8051 40-pin daughtercard hosted on the UNI-DS6 Development Board by MikroElektronika.

This sample toggles bit 0 of Port 2 every second. On the UNI-DS6, this bit is connected to the LED for RC0.

The full source code for this application is in examples\AN_351_FT51A_Compatibility_Module of the FT51A SDK.

To compile and program the UMFT51AA:
```
sdcc -D_FT51 second-toggle.c
ft51prg.exe second-toggle.ihx
```

To compile and program the Atmel AT89S8253:
```
sdcc second-toggle.c
8051Flash.exe -w -pAT89S8253 -q -f "second-toggle.ihx"
```

Note that the UMFT51AA is programmed with (and powered by) an FTPD-1 module connected to CN3 on the UMFT51AA. The AT89S8253 is programmed with (and powered by) a USB cable connected to CN3 on the mikroBoard.

The program begins by defining macros used later.
```
// Helper macro to get most-significant byte of a 2-byte variable.
#ifndef MSB
#define MSB(x) ((unsigned char)(((unsigned int)(x) >> 8) & 0x00ff))
#endif // MSB

// Helper macro to get least-significant byte of a 2-byte variable.
#ifndef LSB
#define LSB(x) ((unsigned char)((unsigned int)(x) & 0x00ff))
#endif // LSB

#ifdef __FT51
  // Default clock frequency for FT51 is 48 MHz
  #define CLK_FREQ_MHZ 48
#else
  // Atmel mikroBoard has 10 MHz clock
  #define CLK_FREQ_MHZ 10
#endif // __FT51

// Compute timer starting value for given number of microseconds.
// Scale is clock frequency in MHz / clock-cycles per timer tick (12).
// 16-bit timer counts up to 65536, so subtract duration from 65536.
#define TIMER_START_VALUE(us) ((unsigned int)(65536UL - (unsigned long)(us) * CLK_FREQ_MHZ / 12 ))
```

Also, the program defines Special Function Registers and bit-addressable variables.
```
__sbit __at (0xA0) P2_0; // Bit 0 of Port 2
__sbit __at (0xB0) P3_0; // Bit 0 of Port 3
__sbit __at (0xA9) ET0; // Timer 0 interrupt-enable
__sbit __at (0xAF) EA; // Master interrupt-enable

__sfr __at (0x88) TCON; // Timer Control
__sfr __at (0x89) TMOD; // Timer Mode
__sfr __at (0x8A) TL0; // Timer start value (low byte)
__sfr __at (0x8C) TH0; // Timer start value (high byte)
__sfr __at (0x8E) IE; // Interrupt Enable
```
The ISR (interrupt service routine) for Timer 0 increments a global variable and reloads the timer with a value which will take 10 milliseconds to overflow. Note that there is no need to clear Timer 0’s Overflow flag; this happens automatically when the ISR is executed.

```c
volatile unsigned char interrupt_counter = 0;

// Interrupt Service Routine to be run every 10 milliseconds.
void Timer0_ISR(void) __interrupt (1)
{
    interrupt_counter++;
    // Reload the 10 ms timer
    TH0 = MSB(TIMER_START_VALUE(10000));
    TL0 = LSB(TIMER_START_VALUE(10000));
}
```

The main function configures Timer 0 to interrupt every 10 milliseconds, then enters an infinite loop which toggles bit 0 of Port 2 every 100 interrupts (every second). Note that the timer interrupt is disabled while accessing the global variable interrupt_counter, which might otherwise be altered by the ISR.

```c
void main(void)
{
    EA = 1; // Enable interrupts in general
    ET0 = 1; // Enable interrupt specifically for Timer0
    P2_0 = 1; // Start with LED off

    // 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 will roll over to zero
    // after 10000 microseconds (one hundred times per second).
    TH0 = MSB(TIMER_START_VALUE(10000));
    TL0 = LSB(TIMER_START_VALUE(10000));

    TCON &= 0xCF; // Clear Timer0’s Overflow and Run flags
    TCON |= 0x10; // Start Timer0 (set its Run flag)

    for (; ;) 
    {
        // Endless loop, interrupted periodically by Timer0_ISR
        ET0 = 0; // Disable timer 0 interrupt while we access counter
        if (interrupt_counter >= 100)
        {
            interrupt_counter = 0;
            // 100 x 10 ms == 1 second, so toggle bit 0 of Port 2
            P2_0 = !P2_0;
        }
        ET0 = 1; // Re-enable timer 0 interrupt
    }
}
12 Initial firmware

FTDI ships the UMFT51AA module with a firmware which is essentially inert except for DFU (Device Firmware Upgrade) capabilities. This means that the firmware may be upgraded by connecting the module to a host PC’s USB port and running a suitable application on the host. This topic is covered in more detail in AN_344_FT51A_DFU_Sample.pdf in the FT51A SDK’s application notes directory.

To help verify at a very basic level that the module functions correctly, FTDI provides another firmware which is the same as the initial firmware but toggles P3.0 periodically. An LED connected to this pin should blink every half-second.

This extra firmware is blink_dfu.ihx in the examples\AN_351_FT51A_Compatibility_Module directory. The command to load it onto the module is:

```bash
ft51dfu blink_dfu.ihx
```

Note that the procedure in AN_344_FT51A_DFU_Sample.pdf must be followed first to install the suitable DFU drivers.
13 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, 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,
200052
Shanghai, P.R. 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.

System and equipment manufacturers and designers are responsible to ensure that their systems, and any Future Technology Devices International Ltd (FTDI) devices incorporated in their systems, meet all applicable safety, regulatory and system-level performance requirements. All application-related information in this document (including application descriptions, suggested FTDI devices and other materials) is provided for reference only. While FTDI has taken care to assure it is accurate, this information is subject to customer confirmation, and FTDI disclaims all liability for system designs and for any applications assistance provided by FTDI. Use of FTDI devices in life support and/or safety applications is entirely at the user’s risk, and the user agrees to defend, indemnify and hold FTDI harmless from any and all damages, claims, suits or expense resulting from such use. This document is subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Neither the whole nor any part of the information contained in, or the product described in this document, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. Future Technology Devices International Ltd, Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH, United Kingdom. Scotland Registered Company Number: SC136640
Appendix A – References

Document References

http://www.ftdichip.com/Products/ICs/FT51.html

[2] FT51A Software Development Kit
http://www.ftdichip.com/Firmware/FT51ARegistration.htm

[3] AN_344 FT51A DFU Sample

Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Terms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analog-to-Digital Converter</td>
</tr>
<tr>
<td>DFU</td>
<td>Device Firmware Upgrade</td>
</tr>
<tr>
<td>GPIO</td>
<td>General-Purpose Input-Output</td>
</tr>
<tr>
<td>I2C</td>
<td>Inter-Integrated Circuit</td>
</tr>
<tr>
<td>IOMUX</td>
<td>Input-Output Multiplexer</td>
</tr>
<tr>
<td>ISR</td>
<td>Interrupt Service Routine</td>
</tr>
<tr>
<td>MTP</td>
<td>Multiple-Time Programmable</td>
</tr>
<tr>
<td>SDCC</td>
<td>Small Device C Compiler</td>
</tr>
<tr>
<td>SFR</td>
<td>Special Function Register</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver/Transmitter</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
</tbody>
</table>
Appendix B – List of Tables & Figures

List of Figures
Figure 1: UMFT51AA pin-out matches the classic 8051. ........................................... 3
## Appendix C – Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Changes</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>First Release</td>
<td>2015-12-21</td>
</tr>
</tbody>
</table>