Application Note AN232BM-01

# FT232BM /FT245BM BIT BANG MODE

## Definition

The FT232BM and FT245BM chips can be set up in a special mode where the normal function of the chips are replaced. This mode changes the 8 data lines on the FT245BM or the RS232 data and control lines of the FT232BM to an 8 bit bi-directional bus. The purpose of this mode was intended to be used to program FPGA devices. It may also be used to talk to serial EEPPROMs or load a data latch.

## Pin Definitions

| FT245BM | FT232BM | Bit-Bang Data bit |
|---|---|---|
| Data0 | TXD | Data0 |
| Data1 | RXD | Data1 |
| Data2 | RTS | Data2 |
| Data3 | CTS | Data3 |
| Data4 | DTR | Data4 |
| Data5 | DSR | Data5 |
| Data6 | DCD | Data6 |
| Data7 | RI | Data7 |

## Mode of Operation

Any data written to the device in the normal manner will be self clocked onto the data pins (for the pins that have been programmed as outputs). Each pin can be set as an input or an output independant of the other pins. The rate of clocking out the data is controlled by the baud rate generator.  This exists in both the FT245BM as well as the FT232BM.

For the data to change, there has to be new data written and the baud clock has to tick. If you do not write any new data to the device then the pins will hold the last value written.

The commands of interest are :

1) FT_SetBaudRate(ftHandle : Dword ; BaudRate : Dword) : FT_Result;

This controls the rate of transferring bytes that have been written to the device onto the pins. It also sets the sampling of the pins and transferring the result back to the Read path of the chip. The maximum baud rate is 3 MegaBaud.  The clock for the Bit Bang mode is actually 16 times the baudrate. A value of  9600 baud would transfer the data at (9600 x 16) = 153600 bytes per second or 1 every 6.5 uS.

2) FT_SetBitMode (ftHandle : Dword ; ucMask , ucEnable : Byte) : FT_Result;

This sets up which bits are input and which are output. The ucMask byte sets the direction. A '1' means the corresponding bit is to be an output. A '0' means the corresponding bit is to be an input. When read data is passed back to the PC, the current pin state for both inputs and outputs will used. The ucEnable byte will turn the bit bang mode off and on. Setting bit 0 to '1' turns it on. Clearing bit 0 to '0' turns it off.

3) FT_GetBitMode ( ftHandle : Dword ; pucData : pointer ) : FT_Result;

This function does an immediate read of the 8 pins and passes back the value. This is useful to see what the pins are doing now. The normal Read pipe will contain the same result but it has also been sampling the pins continuously (up until its buffers become full). Therefor the data in the Read pipe will be old.

## Example : Programing an Altera FLEX10K FPGA

This example interfaces to an Altera FLEX10K FPGA. A cable was made up in the following manner :

| Bit Bang bit | Mode | ALTERA Pin |
|---|---|---|
| 0 | OUT | CLK |
| 1 | OUT | nCONFIG |
| 2 | OUT | DATA0 |
| 3 | IN | nSTATUS |
| 4 | IN | CONF_DONE |
| 5 | IN | not used |
| 6 | IN | not used |
| 7 | IN | not used |

This code was written in Delphi Pascal.

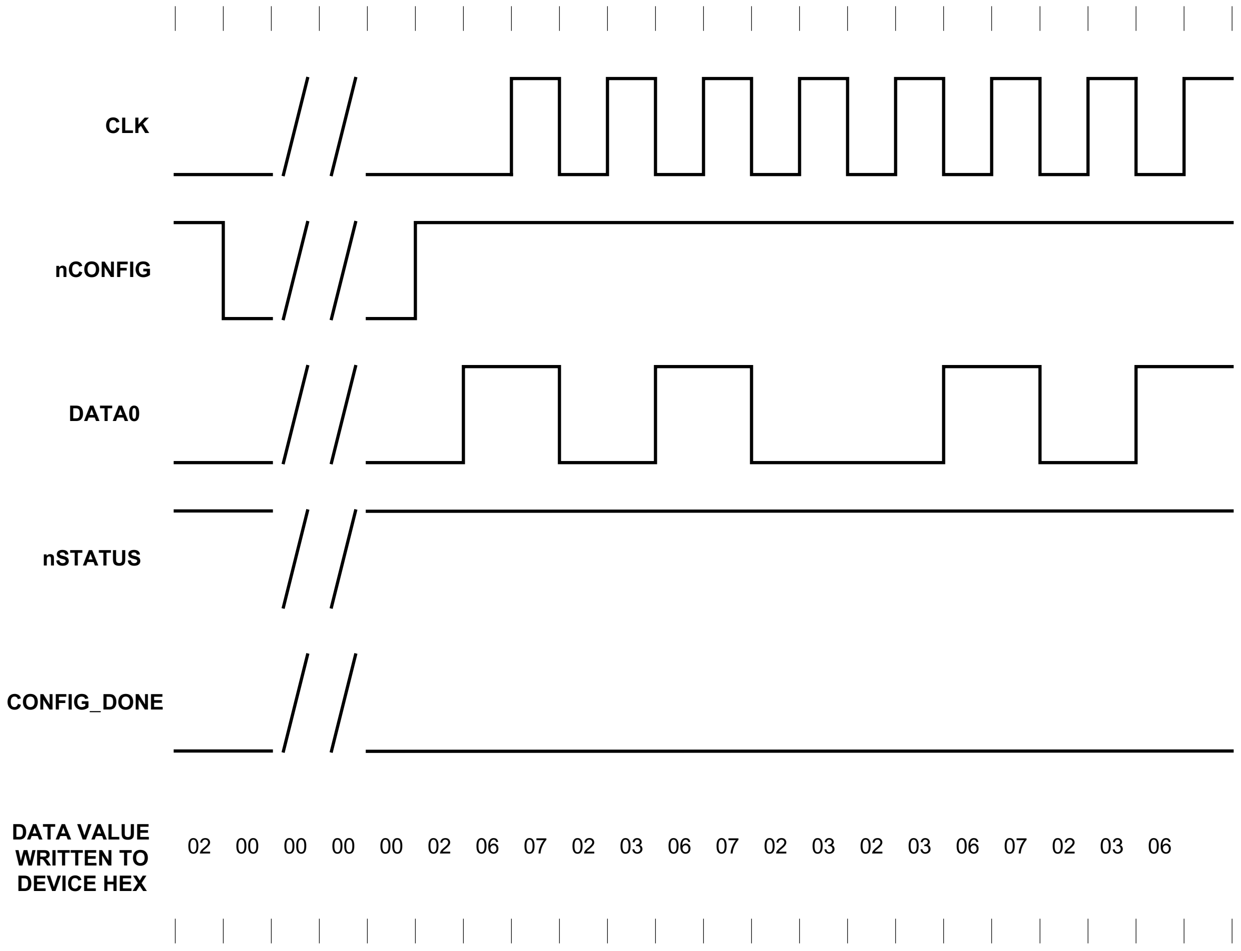
```
function CheckStatus : boolean;

//------------------------------------------------
// returns true unless there is an error.
//
// This is reading the nSTATUS line from the FPGA
//------------------------------------------------

var res : FT_Result;
NStat : byte;
i,j : integer;
begin
CheckStatus := True;
res := Get_USB_Device_BitMode(NStat);

NStat := NStat AND $08;
if NStat = 0 then CheckStatus := False;
end;

procedure SendBitByte2(ByteToSnd : byte; Offset : integer);

//-----------------------------------------------------------------
// This takes a byte to send serialy to the FPGA and converts
// it into 16 bytes to be transmitted down USB. Each bit of the
// byte is put onto data bit 2 of the device and clocked using data
// bit 0.
//-----------------------------------------------------------------

var SendingByte : byte;
begin
// bit 0
SendingByte := $02; // clock low config high
if (ByteToSnd AND $01) <> 0 then SendingByte := SendingByte + $04;
FT_Out_Buffer[Offset + 0]:= SendingByte;
```

```
FT_Out_Buffer[Offset + 1]:= SendingByte + $01; // clock high
// bit 1
SendingByte := $02; // clock low config high
if (ByteToSnd AND $02) <> 0 then SendingByte := SendingByte + $04;
FT_Out_Buffer[Offset + 2]:= SendingByte;
FT_Out_Buffer[Offset + 3]:= SendingByte + $01; // clock high
// bit 2
SendingByte := $02; // clock low config high
if (ByteToSnd AND $04) <> 0 then SendingByte := SendingByte + $04;
FT_Out_Buffer[Offset + 4]:= SendingByte;
FT_Out_Buffer[Offset + 5]:= SendingByte + $01; // clock high
// bit 3
SendingByte := $02; // clock low config high
if (ByteToSnd AND $08) <> 0 then SendingByte := SendingByte + $04;
FT_Out_Buffer[Offset + 6]:= SendingByte;
FT_Out_Buffer[Offset + 7]:= SendingByte + $01; // clock high
// bit 4
SendingByte := $02; // clock low config high
if (ByteToSnd AND $10) <> 0 then SendingByte := SendingByte + $04;
FT_Out_Buffer[Offset + 8]:= SendingByte;
FT_Out_Buffer[Offset + 9]:= SendingByte + $01; // clock high
// bit 5
SendingByte := $02; // clock low config high
if (ByteToSnd AND $20) <> 0 then SendingByte := SendingByte + $04;
FT_Out_Buffer[Offset + 10]:= SendingByte;
FT_Out_Buffer[Offset + 11]:= SendingByte + $01; // clock high
// bit 6
SendingByte := $02; // clock low config high
if (ByteToSnd AND $40) <> 0 then SendingByte := SendingByte + $04;
FT_Out_Buffer[Offset + 12]:= SendingByte;
FT_Out_Buffer[Offset + 13]:= SendingByte + $01; // clock high
// bit 7
SendingByte := $02; // clock low config high
if (ByteToSnd AND $80) <> 0 then SendingByte := SendingByte + $04;
FT_Out_Buffer[Offset + 14]:= SendingByte;
FT_Out_Buffer[Offset + 15]:= SendingByte + $01; // clock high

end;

procedure SendBitByte(ByteToSnd : byte; SendNow : Boolean );

//---------------------------------------------------------------
// This routine calls the routine to serialise the byte and
// then decides if it should send the packet down to USB. It does
// this to maximise band width on the USB bus.
//---------------------------------------------------------------

var SendingByte : byte;
begin
SendBitByte2(ByteToSnd,Offset);
OffSet := OffSet + 16;
if (SendNow OR (OffSet = 4096)) then
  begin
  SendBytes(OffSet);
  OffSet := 0;
  end;

end;

procedure CompleteProg;

//-----------------------------------------------------
// Clock the FPGA 10 times to complete its programming
//-----------------------------------------------------

var SendingByte : byte;
begin
SendingByte := $02; // clock low config high
FT_Out_Buffer[0]:= SendingByte;
FT_Out_Buffer[1]:= SendingByte OR $03; // clock high
SendingByte := $02; // clock low config high
FT_Out_Buffer[2]:= SendingByte;
FT_Out_Buffer[3]:= SendingByte OR $03; // clock high
SendingByte := $02; // clock low config high
FT_Out_Buffer[4]:= SendingByte;
FT_Out_Buffer[5]:= SendingByte OR $03; // clock high
SendingByte := $02; // clock low config high
FT_Out_Buffer[6]:= SendingByte;
FT_Out_Buffer[7]:= SendingByte OR $03; // clock high
SendingByte := $02; // clock low config high
FT_Out_Buffer[8]:= SendingByte;
FT_Out_Buffer[9]:= SendingByte OR $03; // clock high
SendingByte := $02; // clock low config high
FT_Out_Buffer[10]:= SendingByte;
FT_Out_Buffer[11]:= SendingByte OR $03; // clock high
SendingByte := $02; // clock low config high
FT_Out_Buffer[12]:= SendingByte;
FT_Out_Buffer[13]:= SendingByte OR $03; // clock high
SendingByte := $02; // clock low config high
FT_Out_Buffer[14]:= SendingByte;
FT_Out_Buffer[15]:= SendingByte OR $03; // clock high
SendingByte := $02; // clock low config high
FT_Out_Buffer[16]:= SendingByte;
FT_Out_Buffer[17]:= SendingByte OR $03; // clock high
SendingByte := $02; // clock low config high
FT_Out_Buffer[18]:= SendingByte;
FT_Out_Buffer[19]:= SendingByte OR $03; // clock high

SendBytes(20);

end;

function CheckDone : boolean;

//---------------------------------------------------
// Returns false unless it is finished.
```

```
//
// This checks the CONF_DONE line from the FPGA.
//--------------------------------------------------

var res : FT_Result;
NStat : byte;
begin
CheckDone := False;
res := Get_USB_Device_BitMode(NStat);
NStat := NStat AND $10;
if NStat <> 0 then CheckDone := True;
end;

function StartConfig : boolean;
//----------------------------------------------------------
// returns true if confdone is low after pulsing nConfig
//
// It sets up a pulse on nCONFIG by keeping the nCONFIG line
// low over several bytes.
//----------------------------------------------------------
var started : boolean;
begin
FT_Out_Buffer[0]:= $02;
FT_Out_Buffer[1]:= $00; // pulse config
FT_Out_Buffer[2]:= $00; // pulse config
FT_Out_Buffer[3]:= $00; // pulse config
FT_Out_Buffer[4]:= $00; // pulse config
FT_Out_Buffer[5]:= $00; // pulse config
FT_Out_Buffer[6]:= $00; // pulse config
FT_Out_Buffer[7]:= $00; // pulse config
FT_Out_Buffer[8]:= $00; // pulse config
FT_Out_Buffer[9]:= $00; // pulse config
FT_Out_Buffer[10]:= $00; // pulse config
FT_Out_Buffer[11]:= $00; // pulse config
FT_Out_Buffer[12]:= $00; // pulse config
FT_Out_Buffer[13]:= $00; // pulse config
FT_Out_Buffer[14]:= $02;
SendBytes(15);
started := CheckDone;
if started then StartConfig := False else StartConfig := True;

end;

procedure ProgramFPGA;

//---------------------------------------------------------
// This is the main entry point for programming the device.
// It sets up the device and opens the file to use. It then
// reads the file and sends the data to the FPGA checking for
// errors on the way.
//---------------------------------------------------------

var RBFFile : file;
```

```
iBytesRead,BytesWritten : integer;
PageData : Array[0..511] of Byte;
i : integer;
Complete,Passed,ImmSend : Boolean;
res : FT_Result;

begin
// Bit Mode On
OpenComPort;
SetupComPort;
res := Set_USB_Device_BitMode($07,$01);

AssignFile(RBFFile, form1.Edit2.Text);    // Open input file
Reset(RBFFile,1);                 // Record size = 1
Complete := false;
OffSet := 0;
res := Reset_USB_Device;

FT_Current_Baud := 57600; // = 16 * 57600 = 1.08 uS period
res := Set_USB_Device_BaudRate;

BytesWritten := 0;
DisplayClr;

if not( EOF(RBFFile)) then
  begin
  Complete := StartConfig; // start sequence - will return success
  if Complete then Complete := False else Complete := True;
    repeat
    BlockRead(RBFFile, PageData, 256, iBytesRead);
    if (iBytesRead > 0) then
      begin
      for i := 0 to (iBytesRead - 1) do
//        for i := 0 to iBytesRead do
        begin
        ImmSend := False;
        if (i = (iBytesRead - 1)) then
          begin
          if iBytesRead < 256 then
            ImmSend := True else ImmSend := False;
          end;
        SendBitByte(PageData[i],ImmSend);
        BytesWritten := BytesWritten + 1;
        end;
      end;
    Complete := CheckDone;
    Passed := CheckStatus;
    until (iBytesRead = 0) OR Complete OR NOT Passed;
  end;
if (iBytesRead = 0) then
  begin
  end;
if complete then
  begin
  CompleteProg; // for last 10 clocks
  DisplayMsg(‘ Programming Passed ‘);
  end
else
  begin
  if NOT Passed then
    begin
    DisplayMsg(‘ Programming Failed - nStatus’);
    DisplayMsg(‘ Bytes Written = ‘+ inttostr(BytesWritten));
    end
  else
    begin
    DisplayMsg(‘ Programming Failed - ran out of file’);
    DisplayMsg(‘ Bytes Written = ‘+ inttostr(BytesWritten));
    CompleteProg; // for last 10 clocks
    end;
  end;
CloseFile(RBFFile);
res := Set_USB_Device_BitMode($07,$00); // turn bit bang mode off
CloseComPort;
end;
```

## Waveform diagram for the FPGA example


CLK
nCONFIG
DATA0
nSTATUS
CONFIG_DONE
DATA VALUE WRITTEN TO DEVICE HEX
02 00 00 00 00 02 06 07 02 03 06 07 02 03 02 03 06 07 02 03 06