The test of writing a big amount of data to the FT2232-analog prototype.

The test of writing a big amount of data to the FT2232-analog prototype (further Prototype) via the synchronous FIFO interface (further SFIFO).

An external microcontroller is connected to the Prototype via USB and SFIFO interfaces.
In the beginning the microcontroller generates the sequence of USB-packets that corresponds to the Windows 7 enumeration packets (not included in the program text).
Then the microcontroller starts the test of "writing via SFIFO" to the Prototype.

In the beginning of the test microcontrollers pins are switched to the "writing via SFIFO" mode.
Then the microcontroller performs the following actions :
1) It clears the SFIFO if it isn't free in the beginning of the test
2) It writes a random amount of data to the SFIFO
3) It writes the same amount of the same data to the software FIFO for future comparison
4) It reads a random amount of data from the SFIFO
5) It reads the same amount of data from the program FIFO
6) It compares the data from the SFIFO with the data from the program FIFO
Writing and reading of data are performed until the summary amount of written data is less than a set value.

```
//The test of writing a big amount of data to the FT2232-analog prototype
unsigned int I5_TestSFIFO_WriteToI5_ManyBytes(   USB_TypeDef* usb,
                                                 uint8_t UsbAddrTarget,
                                                 uint8_t EndpSourceIn,
                                                 uint8_t EndpSourceOut,
                                                 uint8_t EndpTargetIn,
                                                 uint8_t EndpTargetOut,
                                                 uint8_t InterfaceNumber)  {
//Data to write
uint8_t SFIFO_TxData[I5_SFIFO_WRITE_THROUGHT_USB_OVERFLOW_LEVEL] = {0};
uint32_t SFIFO_NBytesToWrite = 0;  //Number of bytes to write
uint32_t SFIFO_TxBytes = 0;        //Number of bytes that were actually written
uint32_t SFIFO_CntrBytesCurr = 0;  //SFIFO byte counter
uint32_t SFIFO_TxBytesMax = 0;     //Full number of written bytes in the test

//Read data
uint8_t USB_RxData[I5_SFIFO_WRITE_THROUGHT_USB_OVERFLOW_LEVEL] = {0};
uint32_t USB_NBytesToRead = 0;     //Number of bytes to read
uint32_t USB_CntrRxBytes = 0;      //Number of bytes that were actually read
uint8_t USB_RxBytes = 0; //Number of bytes that were actually read in IN token

//Bytes that were read from software FIFO
uint8_t PFIFO_RxData[I5_SFIFO_WRITE_THROUGHT_USB_OVERFLOW_LEVEL] = {0};

unsigned int ResCmd = 0;
unsigned int ResIN = USB_RXCSR1_ERROR_RXSTALL;
unsigned int ResCheck = 0;
unsigned int ResWr = 0;

uint16_t DivInt = 0;
uint8_t DivFrac = 0;

unsigned int i = 0;

unsigned int RxFullNum = 0;
unsigned int RxFullCntr = 0;

unsigned int CleaningTokensNum = 0;//Number of tokens to clean the software FIFO

Buf8_t PFIFO;  //Software FIFO

// Software FIFO initialization
Buf8Init (&PFIFO);

//*********************************************************************
//                                SFIFO
//*********************************************************************
//Set microcontroller pins to output. Transmission I5 -> external device
I1_GpioInitSfifo_Device_To_I5();
I5_SFIFO_RDN_DEACTIVATE(InterfaceNumber);
I5_SFIFO_OEN_DEACTIVATE(InterfaceNumber);
I5_SFIFO_WRN_DEACTIVATE(InterfaceNumber);
```

```
//--------Set clock divider---------
//SetDiv SIU0
//Without optimization
//DivInt = 29999;   //1,kHz
DivInt = 2999; //10,kHz
//DivInt = 1999;    //15,kHz <<<----Write
//DivInt = 1763;    //17,kHz
//DivInt = 1499;    //20,kHz <<<----Read
//DivInt = 1199;    //25,kHz
//DivInt = 999;     //30,kHz

//-O1
//DivInt = 749;     //40,kHz
//DivInt = 599;     //50,kHz <<<---- Write
//DivInt = 499;     //60,kHz
//DivInt = 427;     //70,kHz <<<---- Read
//DivInt = 399;     //75,kHz
//DivInt = 374;     //80,kHz
DivFrac = 0;
//I5_Cmd_SetDiv
ResCmd = I5_Cmd_SetDiv(  usb,                //USB_TypeDef* usb,
                         UsbAddrTarget,      //uint8_t UsbAddrNew,
                         InterfaceNumber,    //uint8_t InterfaceNumber,
                         DivInt,             //uint16_t DivInt,
                         DivFrac);           //uint8_t DivFrac);
if(ResCmd!=1)  {
     TEST_ERROR("I5_Cmd_SetDiv SIU0");
}
else  {
     Menie_printf("I5_Cmd_SetDiv SIU0        OK\n\r");
}


//SetSiuCfg SIU0
ResCmd = I5_Cmd_SetSiuCfg(    usb,
                              UsbAddrTarget,
                              I5_INTERFACE_TYPE_SFIFO,
                              InterfaceNumber);
if(ResCmd!=1)  {
     TEST_ERROR("I5_Cmd_SetSiuCfg SIU0");
}
else  {
     Menie_printf("I5_Cmd_SetSiuCfg SIU0     OK\n\r");
}


//--------Clean FIFO---------
Menie_printf("Cleaning SFIFO\n\r");
CleaningTokensNum = (2*I5_SFIFO_SIZE_BYTES)/USB_ENDPOINT_BULK_PKT_LEN_MAX;
ResIN=USB_ERROR_OK;
for(i=0; i<CleaningTokensNum; i++)  {
```

```
     ResIN = UsbSendIN(  usb,           //USB_TypeDef* usb,
                         UsbAddrTarget, //uint8_t UsbAddrTarget,
                         EndpSourceIn,  //uint8_t EndpSource,
                         EndpTargetIn,  //uint8_t EndpTarget,
                         &USB_RxData[USB_CntrRxBytes], //uint8_t* RxData,
                         &USB_RxBytes); //uint8_t* RxBytes )
     if(ResIN!=USB_ERROR_OK)  {
          if(ResIN==USB_RXCSR1_ERROR_NAK_TIMEOUT)  {
               break;
          }
          else  {
               TEST_ERROR("UsbSendIN reply error at the %uth cleaning token", i);
               return 0;
          }
     }

     if(USB_RxBytes!=USB_ENDPOINT_BULK_PKT_LEN_MAX)  {
          break;
     }
}


//Data counters initialization
SFIFO_TxBytesSum = 0;
SFIFO_CntrBytesCurr = 0;


//---------Exchange of a big amount of data. External device -> I5--------
Menie_printf("Read USB <- Write SFIFO\n\r");
//For debugging
Menie_printf("Transaction  TransmittedBytes  CurrentFilling:\n\r");

//Random generator initial seed
//srand(123);

//SFIFO_TxBytesMax = 8*1024;
//SFIFO_TxBytesMax = 128*1024;
SFIFO_TxBytesMax = 1024*1024;
while(SFIFO_TxBytesSum<SFIFO_TxBytesMax)  {
     //Create data fo fill SFIFO ( The amount of data is random )
     SFIFO_NBytesToWrite = rand() & I5_SFIFO_SIZE_M;
     for(i=0; i<SFIFO_NBytesToWrite; i++)  {
          SFIFO_TxData[i]=(uint8_t) (rand() & 0xFF);
     }

     //Write data to SFIFO via external pins
     SFIFO_TxBytes = 0;
     if(SFIFO_NBytesToWrite>0)  {
          if(!I5_SFIFO_IsFull(InterfaceNumber))  {
           ResWr = I5_SFIFO_WriteData( InterfaceNumber,//uint8_t InterfaceNumber,
                                       SFIFO_NBytesToWrite,//uint32_t NBytesToWrite,
                                       SFIFO_TxData,       //uint8_t* I1_TxData,
```

```
                                        &SFIFO_TxBytes);    //uint32_t* TxBytes)
                if(ResWr==0)  {
                     TEST_ERROR("I5_SFIFO_WriteData");
                     return 0;
                }
                SFIFO_TxBytesSum += SFIFO_TxBytes;
                SFIFO_CntrBytesCurr += SFIFO_TxBytes;
                //For debugging
Menie_printf("WR    %8u    %4u\n\r", SFIFO_TxBytesSum, SFIFO_CntrBytesCurr);
     }
}


//Write data to software FIFO
for(i=0; i<SFIFO_TxBytes; i++)  {
     if(!Buf8Write (&PFIFO, SFIFO_TxData[i]))  {
          break;
     }
}
//Compare amounts of data in SFIFO and PFIFO
if(SFIFO_CntrBytesCurr!=PFIFO.Count)  {
     TEST_ERROR("FIFO's unsynchronization : SFIFO=%u  PFIFO=%u",
                 SFIFO_CntrBytesCurr, PFIFO.Count);
     return 0;
}

//Read data from SFIFO via USB
USB_NBytesToRead = rand() & I5_SFIFO_SIZE_M;

//Calculate the maximum amount of packets to receive
RxFullNum = ((unsigned int) USB_NBytesToRead/USB_ENDPOINT_BULK_PKT_LEN_MAX) + 1;

//Sending IN tokens
USB_CntrRxBytes = 0;
USB_RxBytes = 0;
for(RxFullCntr=1; RxFullCntr<=RxFullNum; RxFullCntr++)  {
     ResIN = UsbSendIN(  usb,           //USB_TypeDef* usb,
                         UsbAddrTarget, //uint8_t UsbAddrTarget,
                         EndpSourceIn,  //uint8_t EndpSource,
                         EndpTargetIn,  //uint8_t EndpTarget,
                         &USB_RxData[USB_CntrRxBytes], //uint8_t* RxData,
                         &USB_RxBytes); //uint8_t* RxBytes );
     //If the sending of the IN token was successful, update the data counters
     if(ResIN==USB_ERROR_OK)  {
          USB_CntrRxBytes += USB_RxBytes;
          SFIFO_CntrBytesCurr -= USB_RxBytes;
     }
     //If there were errors during the IN token sending
     else  {
          //If the error is NAK, which could arise during reading from the empty
          //SFIFO, then stop sending IN tokens
```

```
            if(ResIN==USB_RXCSR1_ERROR_NAK_TIMEOUT)   {
                  break;
            }
            else  {
                  //If there is any other error, then stop the test
                  TEST_ERROR("UsbSendIN at %uth token", RxFullCntr);
                              return 0;
                  }
            }

            if(USB_RxBytes!=USB_ENDPOINT_BULK_PKT_LEN_MAX)   {
                  break;
            }
      }
      //For debugging
      Menie_printf("RD    %8u   %4u\n\r", SFIFO_TxBytesSum, SFIFO_CntrBytesCurr);


      //Read data from the software FIFO
      for(i=0; i<USB_CntrRxBytes; i++)   {
            if(!Buf8Read (&PFIFO, (uint8_t*) &PFIFO_RxData[i]))   {
                  break;
            }
      }
      //Compare amounts of data in SFIFO and PFIFO
      if(SFIFO_CntrBytesCurr!=PFIFO.Count)   {
            TEST_ERROR("FIFO's unsynchronization : SFIFO=%u  PFIFO=%u",
                        SFIFO_CntrBytesCurr, PFIFO.Count);
            return 0;
      }


      //Data checking
      ResCheck = I5_TestSFIFO_WriteToI5_ManyBytes_CheckData(USB_RxData,
                                                            PFIFO_RxData,
                                                            USB_CntrRxBytes);
      if(ResCheck!=1)   {
            TEST_ERROR("CheckData");
            Menie_printf("PFIFO.Count          =%u\n\r", PFIFO.Count);
            Menie_printf("USB_NBytesToRead     =%u\n\r", USB_NBytesToRead);
            Menie_printf("RxFullNum            =%u\n\r", RxFullNum);
            Menie_printf("RxFullCntr           =%u\n\r", RxFullCntr);
            Menie_printf("SFIFO_NBytesToWrite  =%u\n\r", SFIFO_NBytesToWrite);
            Menie_printf("SFIFO_TxBytes        =%u\n\r", SFIFO_TxBytes);
            return 0;
      }
}

return 1;
}
```