Technical Note

TN_167

FIFO Basics (USB2.0)

Version 1.0

Issue Date: 2016-10-05

This document explains how to configure and implement the FIFO interface in FTDI devices.

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 document explains the two FIFO modes available with FTDI full speed and hi-speed USB devices, what devices support these modes, and how to implement FIFO mode in software and hardware. FIFO mode uses a byte wide data bus for high speed data transfer between a PC host and an FPGA or microcontroller.

1.1 Scope

This document applies to the following FTDI devices: FT245B, FT245R, FT240X, FT2232D, FT232H, and FT2232H. All of these devices function as FIFO slaves.

FIFO code examples in C++ and Verilog are available in the appendix and can be downloaded from the FTDI website.

Note: FTDI USB3.0 solutions also include FIFO interfaces but not included in the scope of this document.

2 What is FIFO Communication?

FIFO is an acronym for "First In, First Out", and is designed for much higher speed communication than UART serial. Using FTDI devices, a FIFO can be implemented as an 8, 16, or 32 bit parallel interface; in this document, the focus will be on 8 bit FIFO. There are two types of FIFO communication, Asynchronous and Synchronous. The target devices for FIFO communication are usually microcontrollers or FPGAs.

2.1 FIFO Performance and Mode Select

The following tables show the performance of Asynchronous FIFO and Synchronous FIFO with FTDI USB Full Speed and Hi-Speed devices, and how FIFO mode is selected.

FTDI Device	# Channels	Asynchronous FIFO Performance	Synchronous FIFO Performance
FT245B	1	1 Mbyte/sec	NA
FT245R	1	1 Mbyte/sec	NA
FT240X	1	1 Mbyte/sec	NA
FT2232D	2	1 Mbyte/sec	NA
FT232H	1	8 Mbyte/sec	40 Mbyte/sec
FT2232H	2 Asynchronous or 1 Synchronous	8 Mbyte/sec	40 Mbyte/sec (single channel)

Table 2.1 FIFO Performance

FTDI Device	Asynchronous FIFO Mode	Synchronous FIFO Mode
FT245B	Default operation	NA
FT245R	Default operation	NA
FT240X	Default operation	NA
FT2232D	Requires EEPROM	NA
FT232H	Requires EEPROM	Requires EEPROM and d2xx FT_SetBitMode = 0x40
FT2232H	Requires EEPROM	Requires EEPROM and d2xx FT_SetBitMode = 0x40

Table 2.2 FIFO Mode Selection

2.2 Asynchronous FIFO I/O

With Asynchronous FIFO, data is written to/read from the chip's 8 bit data bus when the WR# or RD# inputs toggle. TXE# and RXF# are the internal buffer status flags. Asynchronous FIFO mode can be accessed with either the VCP or D2XX driver. No special bit mode setting is required for d2xx applications in asynchronous mode; just open a handle to the device and read/write data to the chip. When using the VCP driver for asynchronous FIFO, a simple TTY application such as TeraTerm can be used. Note, there is no baud rate setting required for this mode of operation, the value may be set, but is ignored.

FT245BL Pins	FT245RL Pins	FT240XS Pins	FT2232D Pins	FT232HQ Pins	FT2232HQ Pins	Name/ Dir	Asynchronous FIFO Function
18-25	1, 5, 3, 11, 2, 9, 10, 6	24, 4, 2, 9, 1, 7, 8, 5	24-19, 17, 16	13-20	16-24	D0-D7 (bidi)	8 Bit Data
12	23	21	15	21	26	RXF# (output)	Receive Buffer Full flag
14	22	20	23	25	27	TXE# (output)	Transmit Buffer Empty flag
16	13	11	12	26	28	RD# (input)	Read strobe
15	14	12	11	27	29	WR# (input)	Write strobe

Table 2.3 Asynchronous FIFO Pins, All Devices

The FT245B, FT245R, FT240X, FT2232D, FT232H and FT2232H support asynchronous FIFO mode.

Figure 2.1 is a generic USB FIFO block diagram showing key signals.

The HDL code or firmware in the FPGA/Microcontroller manages flow control with the FTDI FIFO slave.

The application code running on the USB Host reads/writes data to the FIFO slave. In Sync FIFO mode, bit mode must be set to 0x40

In Synchronous FIFO mode, the dashed light blue signals, OE# and CLK_60 MHz, are added to the existing connections.

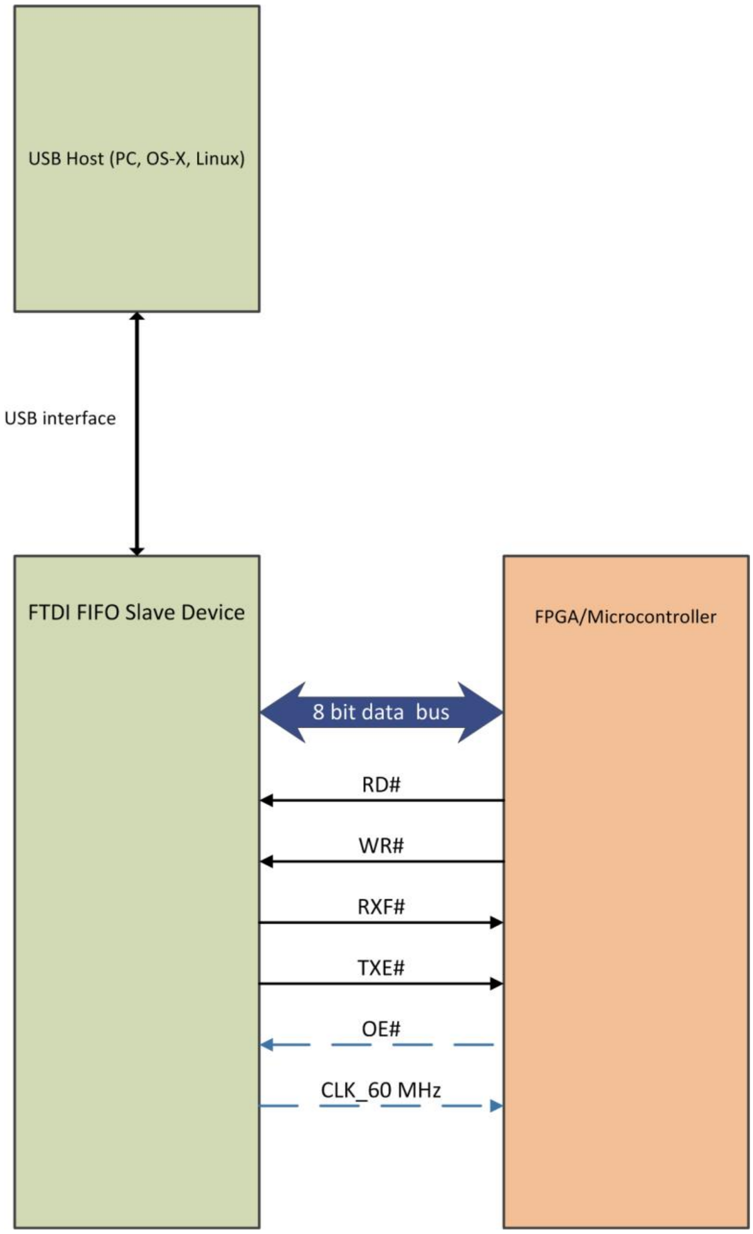

Figure 2.1 Generic FTDI FIFO Slave block diagram

2.2.1 What are Buffer Flags? (TXE# and RXF#)

The TXE# and RXF# outputs are the buffer status pins (flags).

For both Asynchronous and Synchronous FIFO modes, the status of the internal transmit and receive buffers must be monitored by the external FPGA or microcontroller to avoid buffer over run and data loss.

The TX buffer is used by data sent from the FIFO pins back to the host (write operation)

The RX buffer is used by data sent from the host to the FIFO output pins (read operation)

When the TXE# flag is low, this indicates there is enough internal transmit buffer space available for writing data back to the host. The USB host application code (VCP or D2XX for Async FIFO, D2XX for Sync FIFO) must constantly read incoming data from the device to keep the buffer from filling up.

When the RXF# flag is low, this indicates there is still unread data in the internal receive buffer remaining to be read by the downstream FPGA or micro. Instead of interpreting this flag as "receive buffer full", the RXF# flag can best be thought of as "receive buffer not empty yet". When the RXF# flag stays high, the last byte of data in the buffer remains on the data bus and does not change.

In normal asynchronous FIFO read/write operations, it is normal for the RXF# and TXE# flags to toggle briefly during each read/write cycle. The minimal pulse duration is approximately 80 nSec. These runt pulses can be ignored by the firmware/HDL code running on the downstream micro or FPGA.

However, the RD# or WR# strobe inputs **must** be throttled when the TXE# or RXF# buffer flags stay high for over 400 nSec.

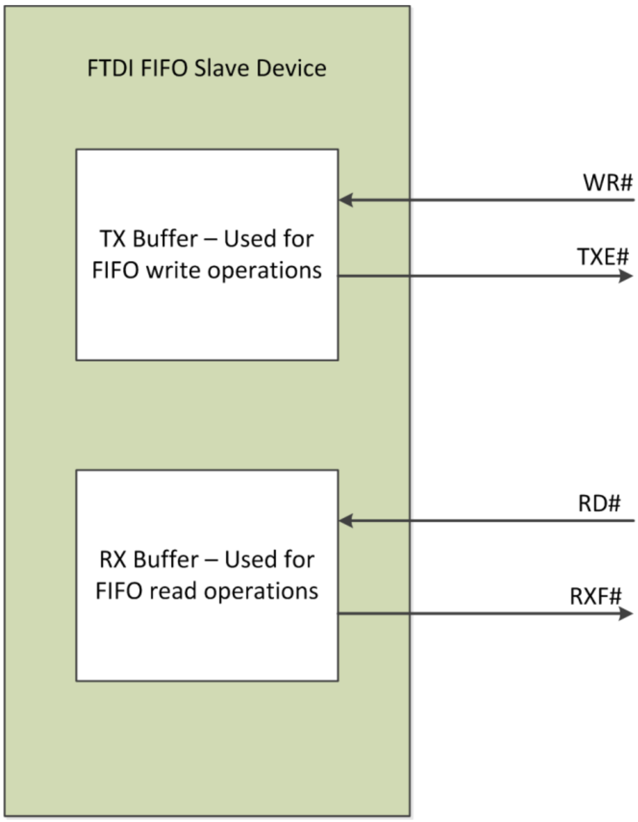

Figure 2.2 FIFO Internal Buffers, R/W Strobes, and Status Flags

2.3 Synchronous FIFO I/O

Synchronous FIFO is only available on the FT232H and FT2232H devices. With synchronous FIFO mode selected, a 60 MHz clock is generated by the FTDI device. This signal clocks data from the downstream device. In contrast to Asynchronous mode, the WR# or RD# pins are held low during data transfer. There is an output enable pin (OE#) that needs to be driven low when data is being read from the FIFO interface. Data loss is prevented by monitoring the TXE# and RXF# flags, as with Asynchronous FIFO mode. Synchronous FIFO mode can only be accessed by the D2XX driver, with Bit Mode set to 0x40.

FT232H Pins	FT2232H Pins	Name	Synchronous FIFO Function
13-20	16-24	ADBUS0-ADBUS7	8 Bit Data (bidirectional)
21	26	ACBUS0	RXF# (output)
25	27	ACBUS1	TXE#(output)
26	28	ACBUS2	RD# (input)
27	29	ACBUS3	WR#(input)
28	30	ACBUS4	SIWU# (input)
29	32	ACBUS5	CLKOUT (60 MHz clock output)
30	33	ACBUS6	OE# (input)

Table 2.4 FT232H/FT2232H Synchronous FIFO Interface Pins

Refer to Figure 2.1 for Synchronous FIFO connections.

2.4 FTDI Drivers

The following table shows what FTDI drivers are used with Asynchronous and Synchronous FIFO modes.

Device	FIFO Mode	Driver (VCP or D2XX)
FT245B/FT245R/FT2232D/FT240X	Asynchronous	VCP or D2XX
FT232H/FT2232H	Asynchronous	VCP or D2XX
FT232H/FT2232H	Synchronous	D2XX only

Table 2.3 Driver Applications

3 Asynchronous FIFO Read/Write Operation

3.1 Asynchronous FIFO Read Timing

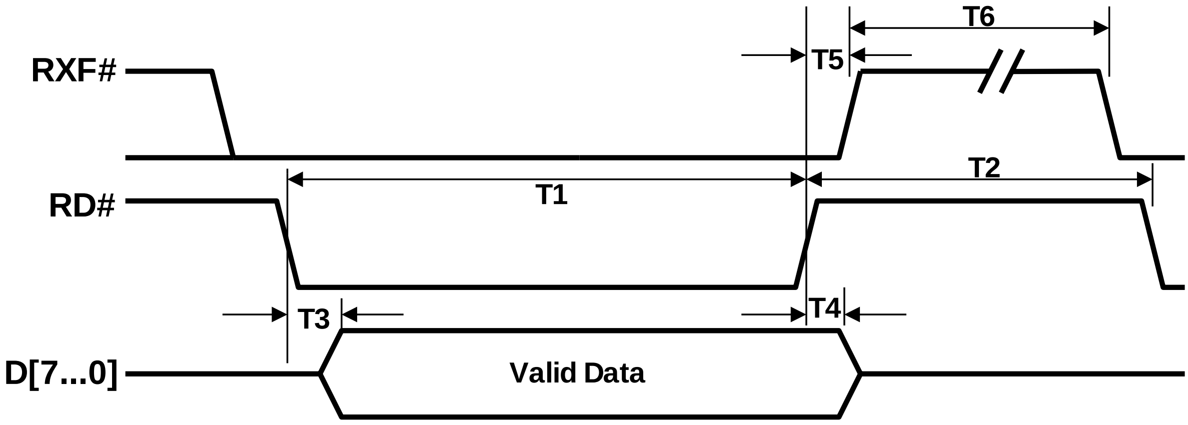

Figure 3.1 Asynchronous FIFO Read Cycle (Full Speed)

Time	Description	Minimum	Maximum	Unit
T1	RD# Active Pulse Width	50	-	ns
T2	RD# to RD# Pre-Charge Time	50 + T6	-	ns
T3	RD# Active to Valid Data*	20	50	ns
T4	Valid Data Hold Time from RD# Inactive*	0	-	ns
T5	RD# Inactive to RXF#	0	25	ns
T6	RXF# Inactive After RD Cycle	80	-	ns

Table 3.1 Async FIFO Read Timing (Full Speed)

3.2 Asynchronous FIFO Read Scope Capture

In this example, a 30 character string of data from the host PC was transmitted to a FT245R module. Note that the RXF# signal toggles for 300 nSec during each read. RD# signal is clocked at 750 KHz. When all the data has been transferred, the RXF# line drives high.

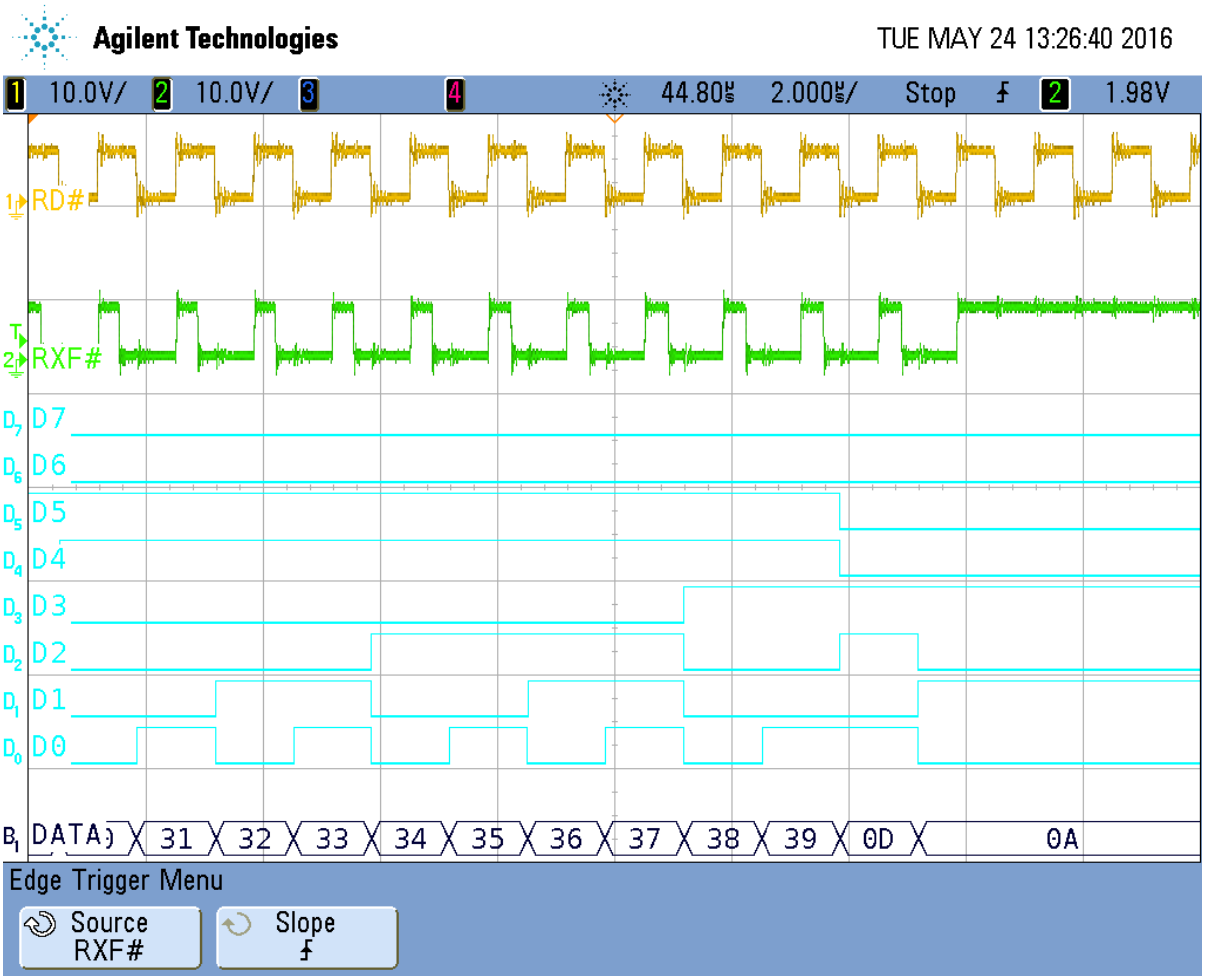

Figure 3.2 Asynchronous FIFO read scope capture from FT245R

3.4 Asynchronous FIFO Write Timing (Full Speed Devices)

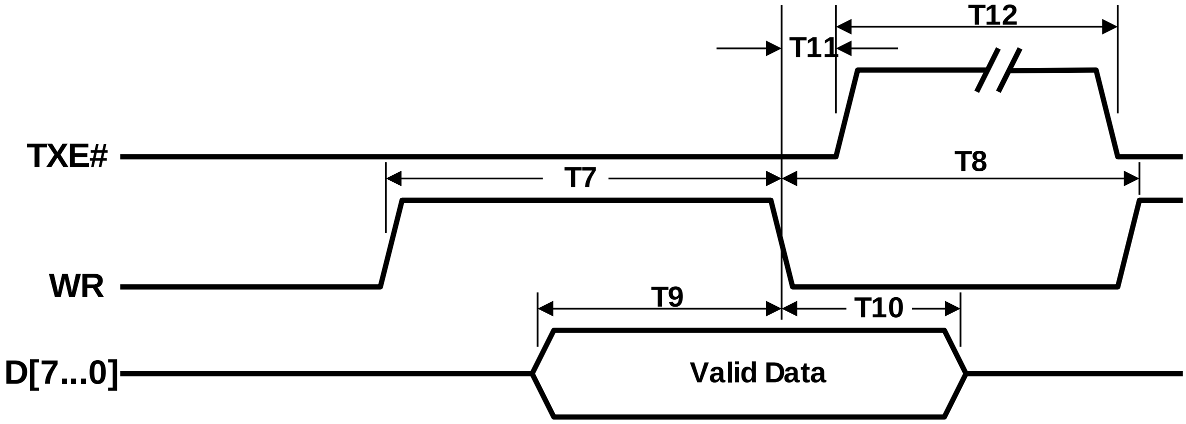

Figure 3.3 Asynchronous FIFO Write Cycle (Full Speed)

Time	Description	Minimum	Maximum	Unit
T7	WR Active Pulse Width	50	-	ns
T8	WR to WR Pre-Charge Time	50	-	ns
T9	Valid data setup to WR falling edge*	20	-	ns
T10	Valid Data Hold Time from WR Inactive*	0	-	ns
T11	WR Inactive to TXE#	5	25	ns
T12	TXE# Inactive After WR Cycle	80	-	ns

Table 3.2 Async FIFO Write Timing (Full Speed)

3.5 Asynchronous FIFO Write Scope Capture

In this example, channel A of a FT2232D module is configured in bit bang mode to simulate an 8 bit data bus. The ADBUS and BCBUS pins on the FT2232D are connected by ribbon cable. Channel B of the FT2232D module is configured in Async FIFO mode to receive data from Channel A and send this data back to the PC host.

A simple TTY application is used to receive incoming FIFO data.

Figure 3.4 shows how a DLP-2232M module is connected to demonstrate an asynchronous FIFO write operation.

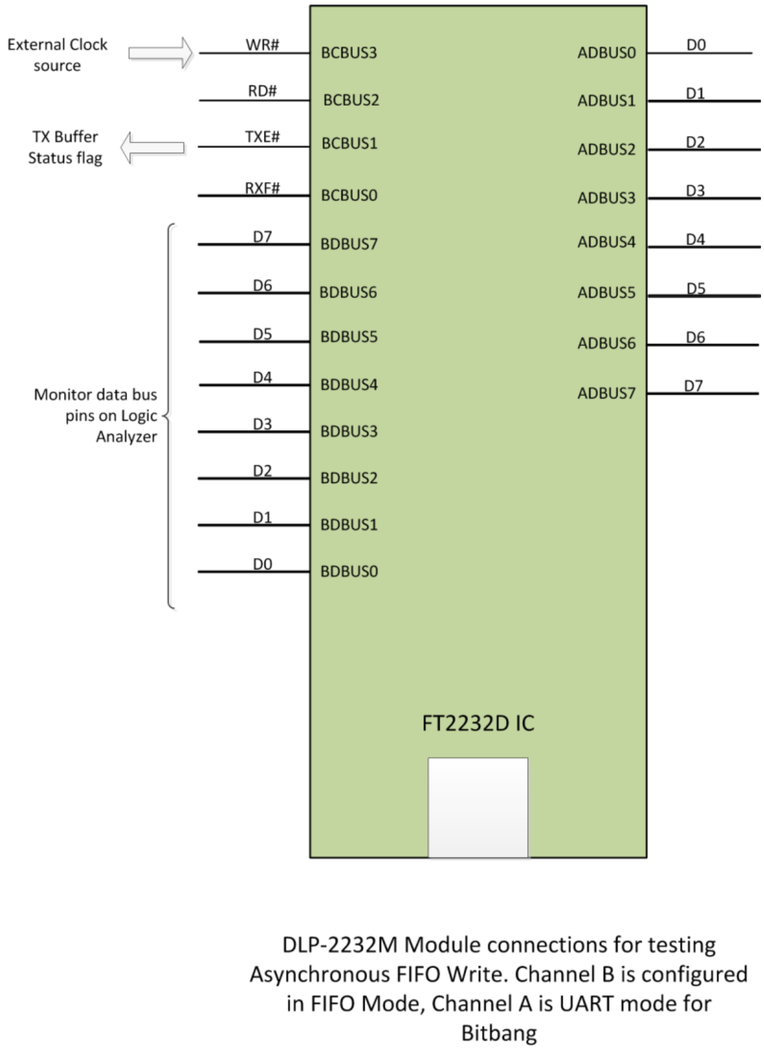

Figure 3.4 Asynchronous FIFO write hardware setup

Figure 3.5 shows the scope triggering on 0xAA. WR# signal is clocked at 100 KHz.

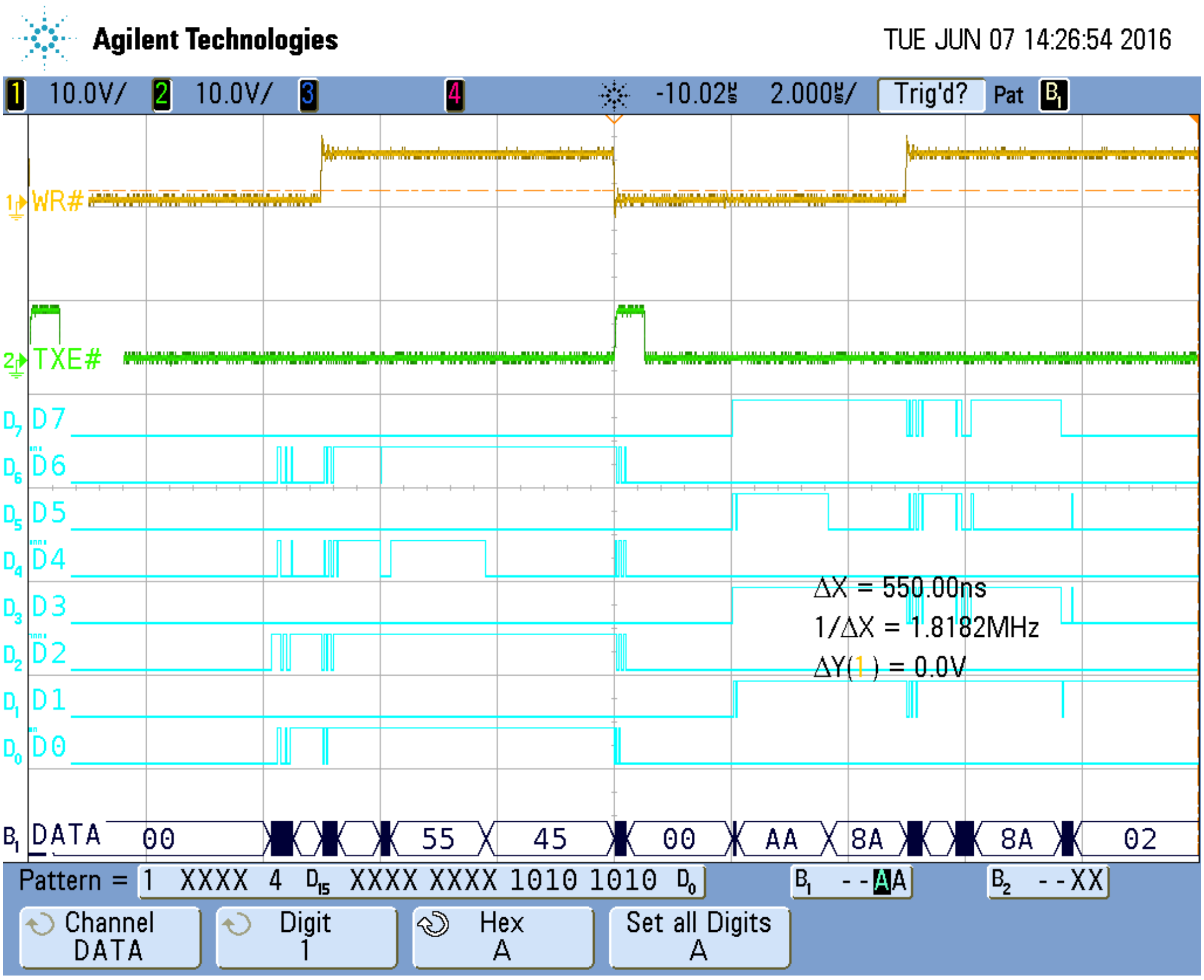

Figure 3.5 Asynchronous FIFO write scope capture from FT2232D

Figure 3.5 shows the same scope trace zoomed in. Note that TXE# high pulse is 500 nSec.

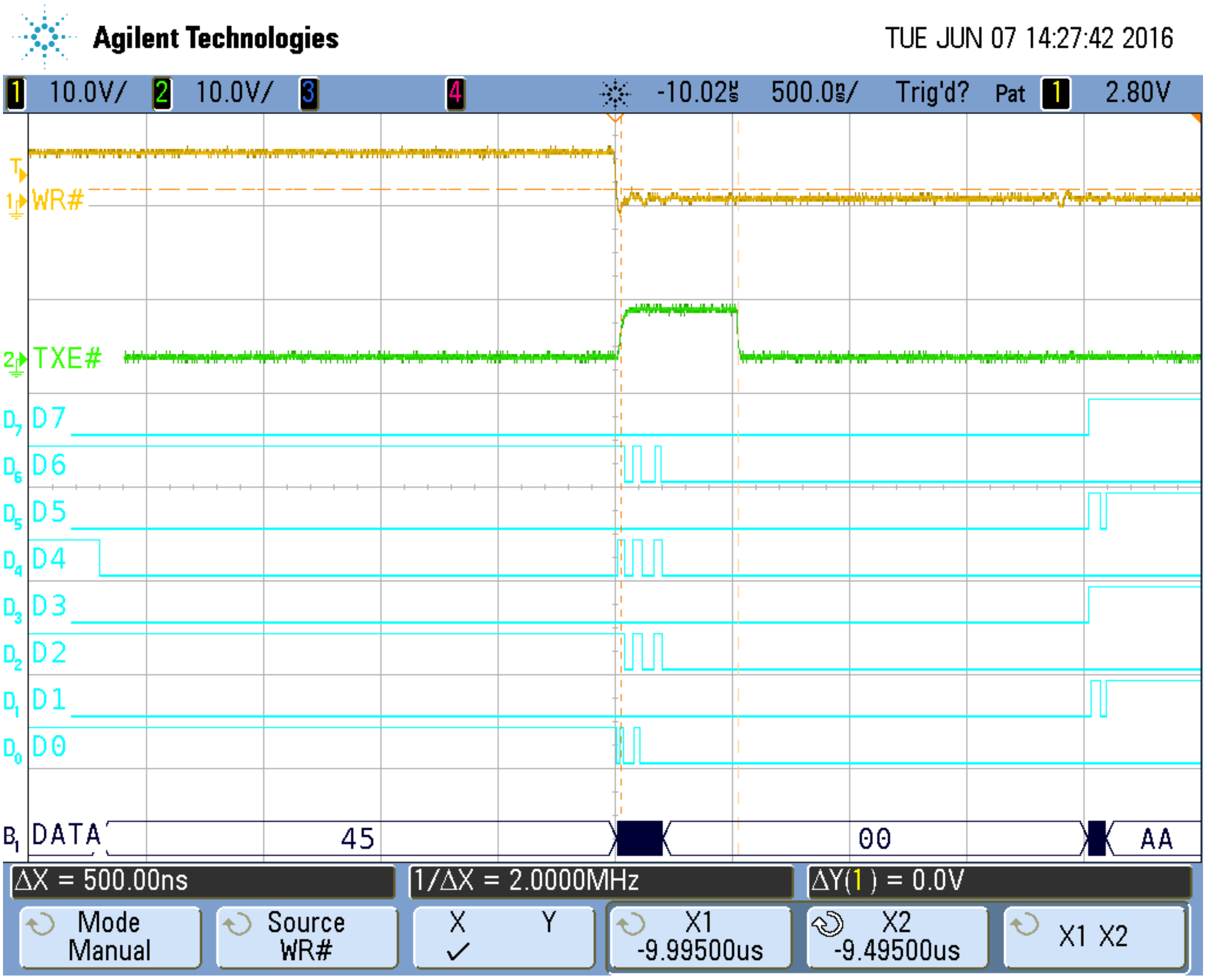

Figure 3.6 Detail of Asynchronous FIFO write scope capture from FT2232D

4 Synchronous FIFO Read/Write Operation

4.1 Synchronous FIFO Read/Write Timing (High Speed Devices)

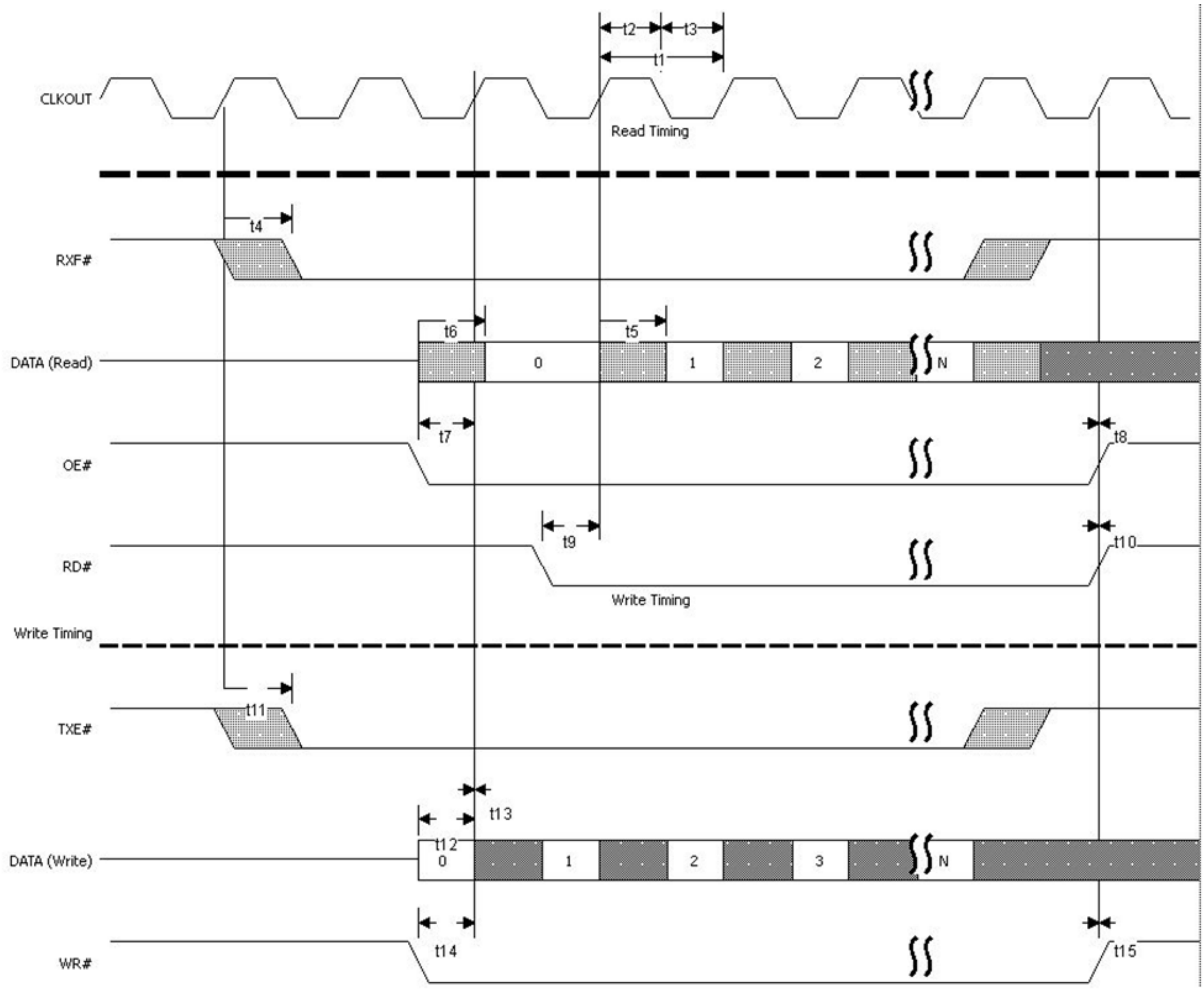

Figure 4.1 Synchronous FIFO Read/Write Cycle (High Speed)

Name	Min	Nom	Max	Units	Comments
t1		16.67		ns	CLKOUT period
t2	7.5	8.33	9.17	ns	CLKOUT high period
t3	7.5	8.33	9.17	ns	CLKOUT low period
t4	0		9	ns	CLKOUT to RXF#
t5	0		9	ns	CLKOUT to read DATA valid
t6	0		9	ns	OE# to read DATA valid
t7	7.5		16.67	ns	OE# setup time
t8	0			ns	OE# hold time
t9	7.5		16.67	ns	RD# setup time to CLKOUT (RD# low after OE# low)
t10	0			ns	RD# hold time
t11	0		9	ns	CLKOUT TO TXE#
t12	7.5		16.67	ns	Write DATA setup time
t13	0			ns	Write DATA hold time
t14	7.5		16.67	ns	WR# setup time to CLKOUT (WR# low after TXE# low)
t15	0				WR# hold time

Table 4.1 Synchronous FIFO Read/Write Timing (High Speed)

4.2 Synchronous FIFO Read Scope Capture (High Speed)

In this example, 16 bytes of data from a d2xx application running on the host PC was transmitted to a FT232H module. Note the RXF# signal stays low until all 16 characters have been received. The RD# signal is held low during this operation, and the data is clocked out of the FT232H by the internally generated 60 MHz clock.

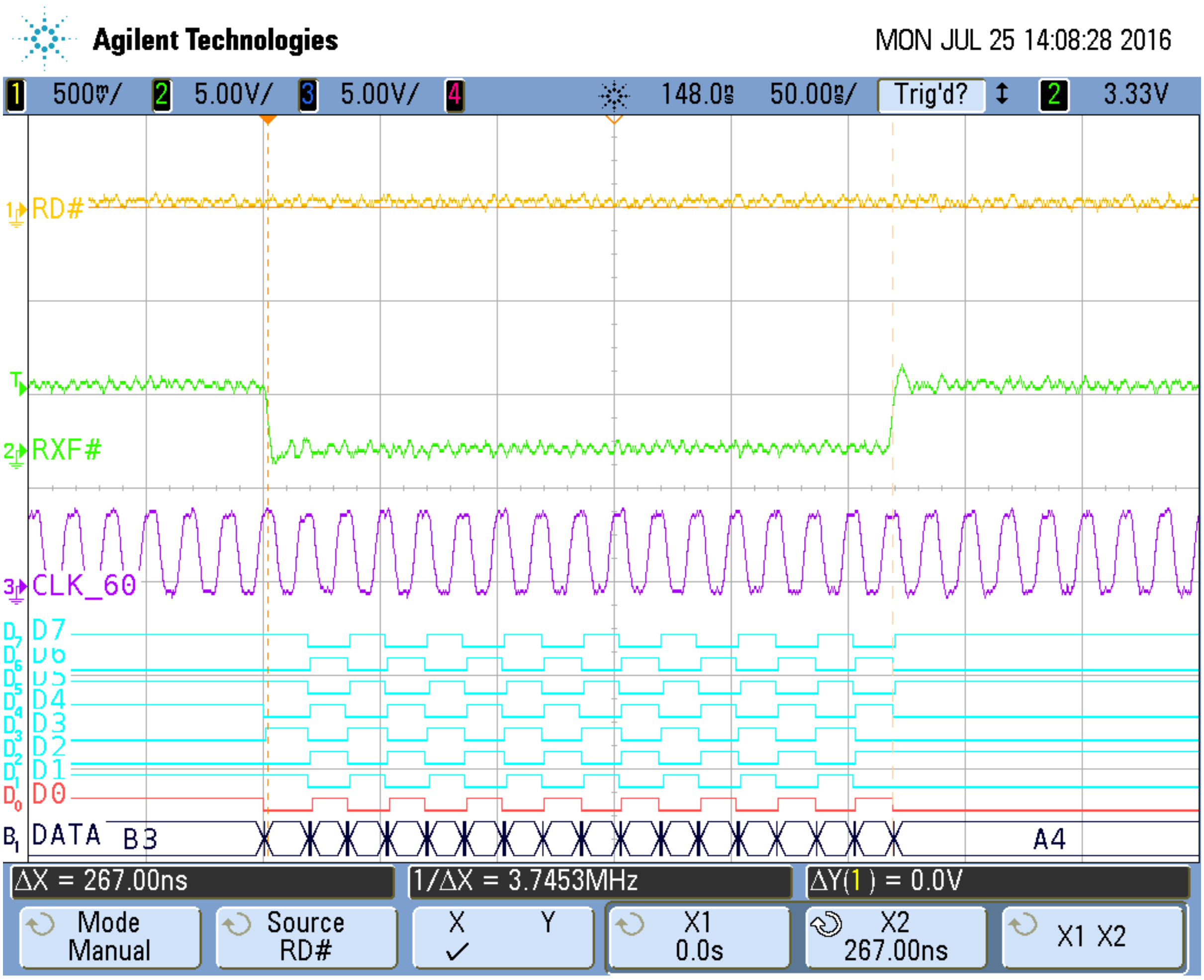

Figure 4.2 Synchronous FIFO read scope capture from FT232H

Figure 4.4 shows the same trace zoomed in to show data being read.

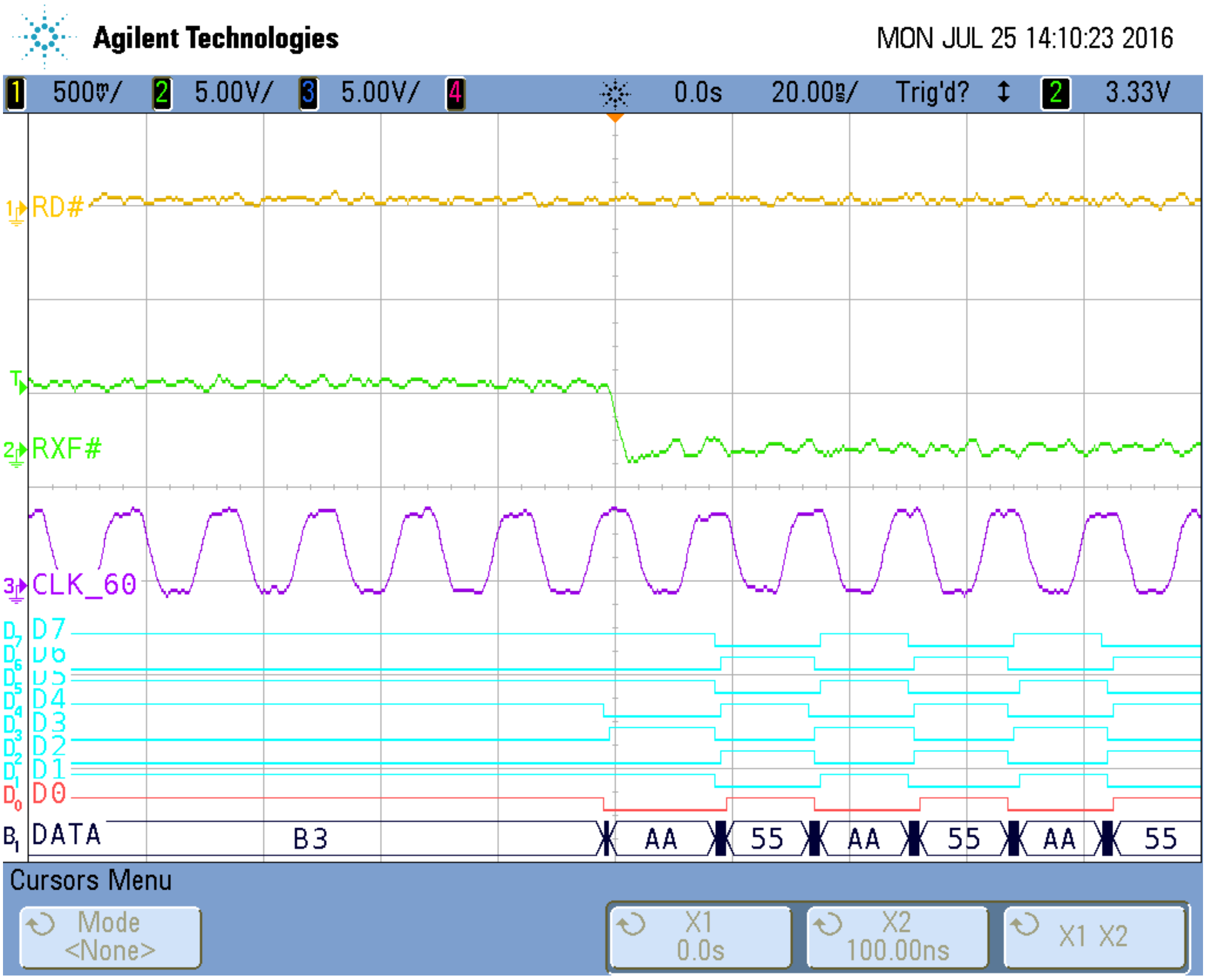

Figure 4.3 Detail of Synchronous FIFO read scope capture from FT232H

4.3 Synchronous FIFO Write Scope Capture (High Speed)

In this example, the FTDI Morph-IC II demo board is used. This board is equipped with a FT2232H chip and an Altera Cyclone 2 FPGA, and is pre-wired to use the Synchronous FIFO I/O pins. An 8 bit synchronous counter with a count enable input is implemented in the FPGA, and the 60 MHz clock out signal from the FT2232H is used as a master clock. The TXE# flag drives the count enable input of the counter. The host PC is running a d2xx application performing a Synchronous FIFO read operation. In this scope capture, you can observe the TXE# flag going high when the internal buffers fill to capacity, halting the counter until the buffers have sufficient storage space available for data transfer.

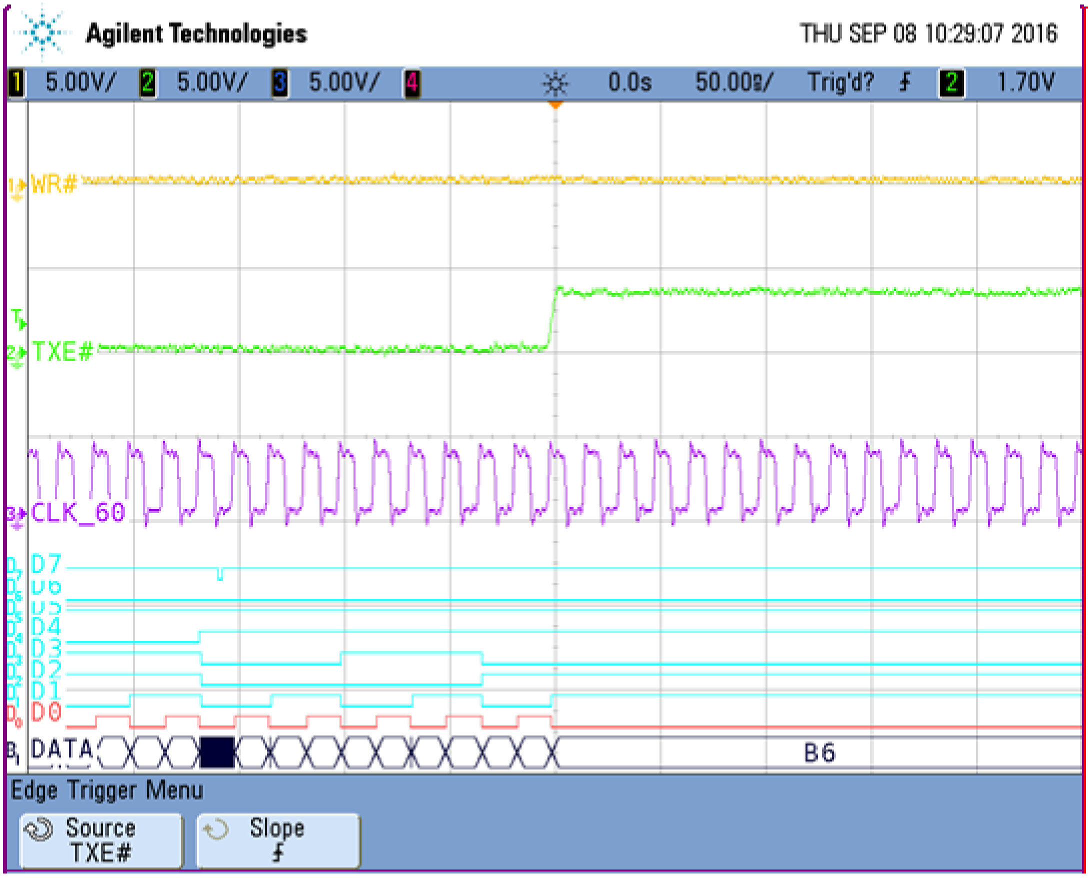

Figure 4.4 Synchronous FIFO write operation scope capture from FT2232H

5 Conclusion

This document demonstrates how simple hardware and code examples can be used to implement Asynchronous and Synchronous FIFO communication with a variety of FTDI devices. Our FIFO interfaces enable very high throughput applications, with a minimum of effort on the part of the designer. FTDI makes FIFO design easy.

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 – 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 8797 1330
Fax: +886 (0) 2 8751 9737

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

AN_146 USB Hardware Design Guides for FTDI ICs

AN_130 FT2232H Used in a FT245 Style Synchronous FIFO Mode

AN_165 Establishing Synchronous 245 FIFO Operations using a Morph-IC-2

FT245BL USB FIFO IC Data Sheet

FT245R USB FIFO IC Data Sheet

FT240X Full Speed USB to 8-Bit FIFO IC Datasheet

FT2232D Dual USB UART/FIFO IC Data Sheet

FT2232H Hi-Speed Dual USB UART/FIFO Data Sheet

FT232H Single Channel Hi-Speed USB Multipurpose UART/FIFO IC

Acronyms and Abbreviations

Terms	Description
USB	Universal Serial Bus
USB-IF	USB Implementers Forum
FIFO	First in First Out 8 bit interface
Asynchronous FIFO	FIFO Read/Write operations controlled by toggling RD# or WR# inputs. No external clock is used.
Synchronous FIFO	High performance version of FIFO interfaces. Data is streamed by holding RD# or WR# low and using 60 MHz clock from FT232H/FT2232H to stream data

Appendix B – Source Code for FIFO examples

Synchronous FIFO Read d2xx Example

(This code transfers data from host to the Synchronous FIFO interface)

```
#include <windows.h>
#include <stdio.h>
#include "ftd2xx.h"

int main(int argc, char* argv[])
{
        FT_HANDLE fthandle;
        FT_STATUS status;
        status = FT_Open(0, &fthandle);

                                if(status != FT_OK)
                                {
                                printf("open status not ok %d\n", status);
                                return 0;
                                }

 status = FT_ResetDevice(fthandle);
              if(status != FT_OK)
              printf("reset status not ok %d\n", status);

UCHAR Mask = 0xFF; // Set data bus to outputs
UCHAR mode = 0;
UCHAR mode1 = 0x40; // Configure FT2232H into 0x40 Sync FIFO mode

 status = FT_SetBitMode(fthandle, Mask, mode);    // reset MPSSE
 status = FT_SetBitMode(fthandle, Mask, mode1);   // configure FT2232H into Sync FIFO mode

                if(status != FT_OK)
               printf("mode status not ok %d\n", status);

                DWORD data_out = 0xAA;
                DWORD data_written;
                INT loop;

                UCHAR data_buf[16] = {0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55,
0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55};

for (loop=1;loop<1000000;loop++)

{
status = FT_Write(fthandle, &data_buf, 16, &data_written);

        printf("loop number  %d\n", loop);
}
                           if(status != FT_OK)
                           printf("status not ok %d\n", status);
                           else
                           printf("output data \n");
        status = FT_Close(fthandle);

                   return 0;

}
```

Synchronous FIFO Write d2xx Example

(this code transfers data from the synchronous FIFO interface back to the host)

```
#include <windows.h>
#include <stdio.h>
#include "ftd2xx.h"

int main(int argc, char* argv[])
{
        FT_HANDLE fthandle1;
        FT_STATUS status;

status = FT_Open(0, &fthandle1);
                                if(status != FT_OK)
                                {
                                printf("open device status not ok %d\n", status);
                                return 0;
                                }

 status = FT_SetTimeouts(fthandle1,500,500);

               if(status != FT_OK)
               printf("timeout device status not ok %d\n", status);

UCHAR MaskA = 0x00; // Set data bus to inputs
UCHAR modeA = 0x40;  // Configure FT2232H into 0x40 Sync FIFO Mode

 status = FT_SetBitMode(fthandle1, MaskA, modeA);

                           if(status != FT_OK)
               printf("mode A status not ok %d\n", status);

Sleep(500);

DWORD RxBytes;
DWORD TxBytes;
DWORD EventDword;

status = FT_GetStatus(fthandle1, &RxBytes, &TxBytes, &EventDword);

                        printf("bytes in RX queue %d\n", RxBytes);

                        printf("\n")

                UCHAR data_in[65536]; // declare a large buffer for incoming data
                DWORD r_data_len = RxBytes;
                DWORD data_read;
                memset(data_in,0,1028);

 status = FT_Read(fthandle1, data_in, r_data_len, &data_read);

                           if(status != FT_OK)
               printf("status not ok %d\n", status);

                           else {
                                        printf("bytes read %d\n", data_read);
                                        printf("data read %x\n", data_in[0]);
                                        printf("data read %x\n", data_in[1]);
                                        printf("data read %x\n", data_in[2]);
                                        printf("data read %x\n", data_in[3]);
                                }
getchar()

status = FT_Close(fthandle1);

                   return 0;

}
```

Asynchronous FIFO Read d2xx Example

(This code transfers data from the host to the asynchronous FIFO interface)

- Code is almost identical to synchronous FIFO read, except there is no reference to bit mode.
- Since asynchronous FIFO mode also uses the VCP driver, a TTY application such as TeraTerm can be used for asynchronous FIFO communication.

```
#include <windows.h>
#include <stdio.h>
#include "ftd2xx.h"

int main(int argc, char* argv[])
{
        FT_HANDLE fthandle;
        FT_STATUS status;
        status = FT_Open(0, &fthandle);

                                if(status != FT_OK)
                                {
                                printf("open status not ok %d\n", status);
                                return 0;
                                }

 status = FT_ResetDevice(fthandle);
              if(status != FT_OK)
              printf("reset status not ok %d\n", status);

                DWORD data_out = 0xAA;
                DWORD data_written;
                INT loop;

                UCHAR data_buf[16] = {0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55,
0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55};

for (loop=1;loop<1000000;loop++)

{
status = FT_Write(fthandle, &data_buf, 16, &data_written);

        printf("loop number  %d\n", loop);
}
                           if(status != FT_OK)
                           printf("status not ok %d\n", status);
                           else
                           printf("output data \n");
        status = FT_Close(fthandle);

                   return 0;

}
```

Asynchronous FIFO Write d2xx Example

(this code transfers data from the asynchronous FIFO interface back to the host)

- Code is almost identical to synchronous FIFO write, except there is no reference to bit mode.
- Since asynchronous FIFO mode also uses the VCP driver, a TTY application such as TeraTerm can be used for asynchronous FIFO communication.

```
#include <windows.h>
#include <stdio.h>
#include "ftd2xx.h"

int main(int argc, char* argv[])
{
        FT_HANDLE fthandle1;
        FT_STATUS status;

status = FT_Open(0, &fthandle1);
                                if(status != FT_OK)
                                {
                                printf("open device status not ok %d\n", status);
                                return 0;
                                }

 status = FT_SetTimeouts(fthandle1,500,500);

               if(status != FT_OK)
               printf("timeout A status not ok %d\n", status);


DWORD RxBytes;
DWORD TxBytes;
DWORD EventDword;

status = FT_GetStatus(fthandle1, &RxBytes, &TxBytes, &EventDword);

                        printf("bytes in RX queue %d\n", RxBytes);

                        printf("\n")

                UCHAR data_in[65536]; // declare a large buffer for incoming data
                DWORD r_data_len = RxBytes;
                DWORD data_read;
                memset(data_in,0,1028);

 status = FT_Read(fthandle1, data_in, r_data_len, &data_read);

                           if(status != FT_OK)
               printf("status not ok %d\n", status);

                           else {
                                        printf("bytes read %d\n", data_read);
                                        printf("data read %x\n", data_in[0]);
                                        printf("data read %x\n", data_in[1]);
                                        printf("data read %x\n", data_in[2]);
                                        printf("data read %x\n", data_in[3]);
                                }
getchar()

status = FT_Close(fthandle1);

                return 0;
}
```

Verilog Example for 8 Bit Counter

```
module Counter8   (
out      , // Output of the counter
enable   , // Enable counting, driven by TXE# from FT2232H
clk      , // clock input from FT2232H
read_n   , // output to FT2232H RD#
write_n  , // output to FT2232H WR#
out_en   , // output to FT2232H OE#
send_im  , // output to FT2232H SI/WUA#
clk_pin  , // monitor CLK 60 from Morphic IO pins
);
//----------Output Ports--------------
output [7:0] out;
output clk_pin;
output reg read_n = 1'b1;
output reg write_n = 1'b0;
output reg send_im = 1'b1;
output reg out_en = 1'b1;

//------------Input Ports--------------

input clk, enable;

//------------Internal Variables--------
reg [7:0] out;
reg reset = 1'b1;
//reg enable = 1'b0;
reg [7:0] data = 8'b0;
reg load = 1'b0;
//-------------Code Starts Here-------
buf B1 (clk_pin, clk); //brings out CLK_60 to Morphic pin J1-28 (IOK4)
always @(posedge clk)
if (reset == 0) begin
  out <= 8'b0 ;
end else if (load == 1) begin
  out <= data;
end else if (enable == 0) begin
  out <= out + 1;
end

endmodule
```

Appendix C – List of Tables & Figures

List of Tables

List of Figures

Appendix D – Revision History

Document Title: TN_167 FIFO Basics (USB2.0)

Document Reference No.: FT_001328

Clearance No.: FTDI# 511

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

Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial Release	2016-10-05