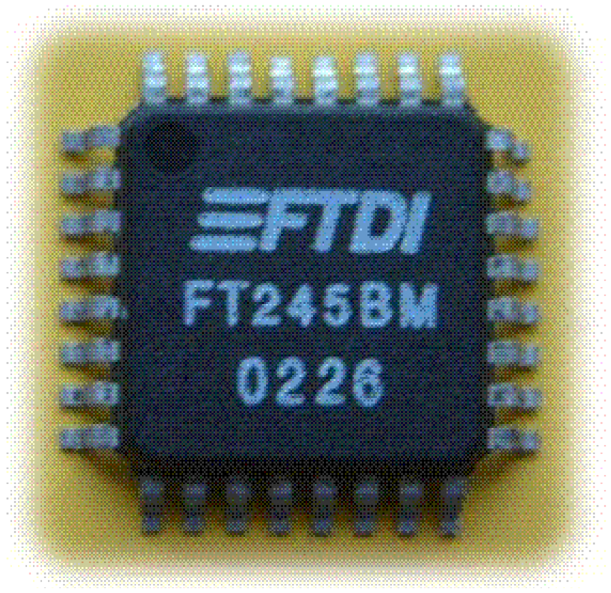
A normalized USB interface based on the FT245B chip from FTDI.



D.BRETON, LAL ORSAY

With the valuable help of C.CHEIKALI and B.LAVIGNE

The current document seeks to describe a USB interface implementation proposal based on the FTDI FT245B circuit. The latter in fact being only a byte pipe, a framing protocol for distinguishing bytes is also proposed. The objective is to obtain a « plug and play » combination comprising the interface part of the board, the decoding block to integrate in a FPGA on the user side and the graphic interface software running on PC and MacIntosh. Furthermore, the user side FPGA interface seeks to be the simplest possible, while offering all of the different current modes of functioning in data acquisition systems. To this end, three types of interfaces are offered.

Evolutions of the document :

- **V2.0**: overall rewriting of version V1.6. The former version of the interface is named simplified. Introduction of the standard version of the interface with n_wait input for slow peripheral, and of the extended version with read_req input and busy output.
- **V2.1**: a few upgrades.
- **V2.2**: modification of the interrupts format. N_write has a width of 2T and n_sync of T. Busy surrounds the data in all modes.
- **V2.3**: translation from French version by Layla Breton. New modification of the interrupts format. The values of the different timings are refined.

1. Description of the FT245B circuit.

The diagram of the FT245B chip is shown on Figure 1. On the user side, it appears as a combination of two FIFOs accessible in a common, bi-directional way via an 8-bit data bus and only 4 control signals, two for the write access and two for the read access. One of these two FIFOs (RX) of a depth of 128 bytes receives the data from the USB bus and the other (TX) of a depth of 384 bytes makes it possible to transmit data to the USB bus.

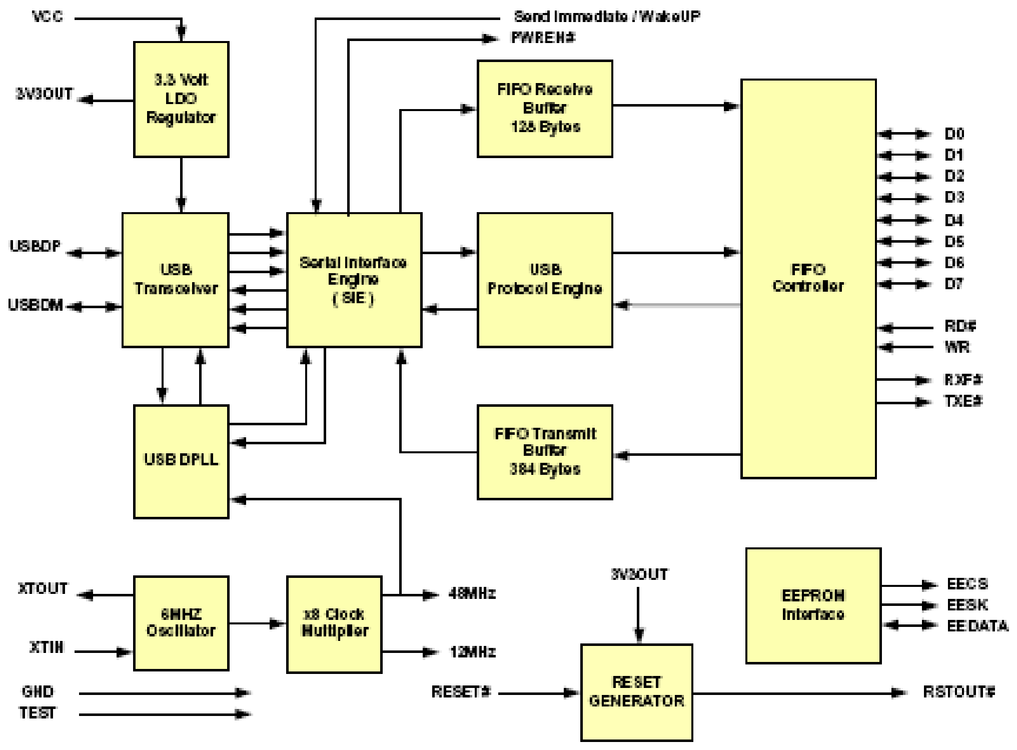


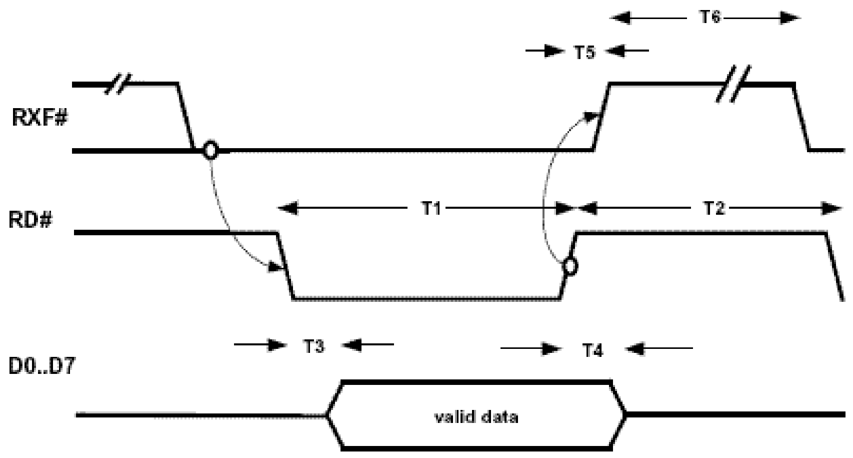
Figure 1 : simplified block diagram of the FT245B.

One must be attentive to the terminology used here, particularly if one is referring to the common definition of a write or read operation. Indeed, we mean by write a transfer of data from a program in execution on the host machine towards the peripheral, which in the present case will appear as a read operation by the peripheral of the circuit interface RX FIFO. In the same manner, a read operation corresponding to a data transfer from the peripheral towards the program in execution on the host machine will appear as a write by the peripheral in the interface circuit TX FIFO.

In the general case in which the program in execution is the only director of the transfer operations, and in which we wish for the USB protocol to be totally transparent, the block interface must execute these access operations in an autonomous fashion in order to only give simple, classic orders on the user side. This will therefore be the case with operations referred to as « standard ».

The four control signals are composed of two flags linked to the state of the FIFOs, RXF* and TXE*, and of two command signals RD* and WR. They function in the following manner:

- When the host machine sends data to the peripheral via the USB, the circuit will pull the RXF* signal to 0 to indicate to the peripheral that at least one byte of data is available. The reading of the RX FIFO will be done with the help of RD*. RXF* returns to 1 after the reading of each byte (see Figure 2).
- **Note** : the rise and fall times of the data on the bus can be very long, approximately 200ns for a charge of approximately 50pF (it appears that the output buffers of the FT245B on the latter are quite minimal). Therefore the real width of the RD* signal must be at least 200ns in order to be able to properly sample the data at their destination. Similarly, it is preferable to detect the falling edge of the RXF* rather than its low level to generate a reading, because the delay preceding its rise it rather unpredictable.

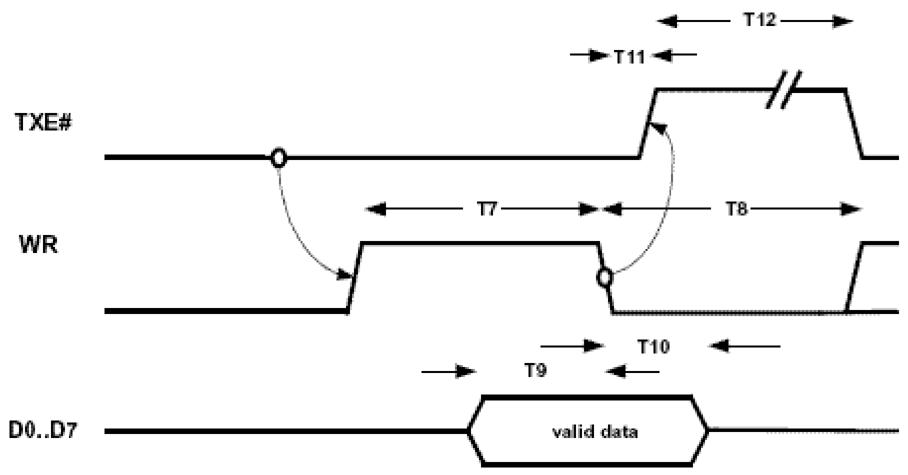


Time	Description	Min	Max	Unit
T1	RD Active Pulse Width	50		ns
T2	RD to RD Pre-Charge Time	50 + T6		ns
T3	RD Active to Valid Data *** Note 4	20	50	ns
T4	Valid Data Hold Time from RD Inactive *** Note 4	0		ns
T5	RD Inactive to RXF#	0	25	ns
T6	RXF inactive after RD cycle	80		ns

*** Note 4 - Load 30 pF

Figure 2 : reading cycle of the RX FIFO.

- To send data from the peripheral towards the host machine via the USB, we will use the WR signal to write in the TX FIFO if TXE* is at 0. If the buffer is busy transferring the data towards the FIFO or if the FIFO is full, TXE* will be held at 1. TXE* rises to 1 after the writing of each byte (see Figure 3).



Time	Description	Min	Max	Unit
T7	WR Active Pulse Width	50		ns
T8	WR to WR Pre-Charge Time	50		ns
T9	Data Setup Time before WR inactive	20		ns
T10	Data Hold Time from WR inactive	0		ns
T11	WR Inactive to TXE#	5	25	ns
T12	TXE inactive after WR cycle	80		ns

Figure 3 : writing cycle of the TX FIFO.

In fact, all of these operations will be transparent to the user. They are described here to explain what the USB-side interface will have to do.

2. Encapsulation of the data frames on the USB bus.

As mentioned above, the FT245B circuit only acts as a byte pipe. To synchronize and secure the transfers, a framing protocol is therefore necessary. This format was however chosen to be the simplest possible and to generate a minimal amount of dead time. The principles are the following :

- A fixed header byte.
- A control byte with the eight least significant bits (LSB) of the number of data bytes to transfer (less one to authorize all of the values from 1 to 256).
- A control byte to indicate the direction of the transfer and to give the sub-address.
- N data bytes.
- A fixed trailer byte.

This format (see Tabs 1A, 1B and 1C) is the same for the writing and reading transfers, which simplifies the data identification. For the writes, the number of data words is limited to 256 (8 bits). For the read requests, the frame will still comprise 5 bytes, but the « data » field will be used to define the most significant bit (MSB) of the number of words to read, which will therefore be able to reach 65536 (16 bits).

Byte\bit	b7	b6	b5	b4	b3	b2	b1	b0
Header	1	0	1	0	1	0	1	0
Control 1	Nb of words – 1 (N – 1)							
Control 2	R/W=0	Subadd						
Data x N**	Data byte x N (N < 256)							
Trailer	0	1	0	1	0	1	0	1

Tab 1A : format of write data frame.

Byte\bit	b7	b6	b5	b4	b3	b2	b1	b0
Header	1	0	1	0	1	0	1	0
Control 1	Nb of words – 1 (N – 1) => LSB							
Control 2	R/W=1	Subadd						
Data	Nb of words – 1 (N – 1) => MSB							
Trailer	0	1	0	1	0	1	0	1

Tab 1B : format of read demand data frame.

Byte\bit	b7	b6	b5	b4	b3	b2	b1	b0
Header	1	0	1	0	1	0	1	0
Control 1	Nb of words – 1 (N – 1) => LSB							
Control 2	R/W=1	Subadd						
Data x N**	Data byte x N (N < 65536)							
Trailer	0	1	0	1	0	1	0	1

Tab 1C : format of read data frame.

Thus, during each transfer, the reception state machine synchronizes itself on the header. If the machine is waiting for a header and finds another byte, an interrupt is emitted towards the USB to signal the error. The sending of interrupts for a header error is then inhibited until the reception of the next header. Once this header is found, and knowing the length of the frame which is encoded in the next byte, the machine can verify that the trailer is present at the expected place, and that the transfer was actually well performed. In the opposite case, an interrupt is also emitted towards the USB to signal the error. The user can also generate an interrupt.

The format of the interrupt is the following (Tab 2) :

Byte\bit	b7	b6	b5	b4	b3	b2	b1	b0
Header	1	0	1	0	1	0	1	0
Control 1	0	0	0	0	0	0	0	0
Control 2	0	Subadd						
Data	Status byte*							
Trailer	0	1	0	1	0	1	0	1

* depending upon the type of interrupt

Tab 2 : format of an interrupt.

It therefore involves a frame with a data byte but where the R/W bit sent back equals 0, which is impossible during a normal reading (where it equals 1). The sub-address will be the last used for an interrupt following a transmission error (header or trailer) or will equal 0x7F in the case of an interrupt generated by the user. The status word depends upon the type of interface and of interrupt (see the following chapters).

There is an order of priority for the interrupts :

1) Header
2) Trailer
3) User

Note :

- the sub-address 0x7F is reserved for the interrupts and must therefore not be utilized by the program running on the host machine.
- the sub-address 0x7E is reserved for the interface version number. A reading of this sub-address will give a byte to translate into 2 quartets (0x23 for version 2.3). This number will be used by the software to recognize it and choose the adapted library. The oldest versions do not comprise this version number.

3. Simplified interface block : usb_light_interface.

The objective of the project is to define a generic interface block between the FT245B and the user. It currently involves an ALTERA block to integrate in the projects. A generic VERILOG version is in the course of development and will be available later on. The user side interface is limited to the minimum useful signals, and is compatible with the other GPIB and VME interface blocks already in use in the lab.

As we also wish to be able to be compatible with the older FPGAs without integrated RAM blocks, the proposed interface has a size reduced to the strict necessity. It is thus a blend of graphic input for the schematics and of AHDL for the state machines. It will be suitable in fact for most « standard » cases, where the program in the course of execution is the only director of the transfers, as for instance with VME or with GPIB, and where the access times to the peripheral are less than 100ns for writing or reading.

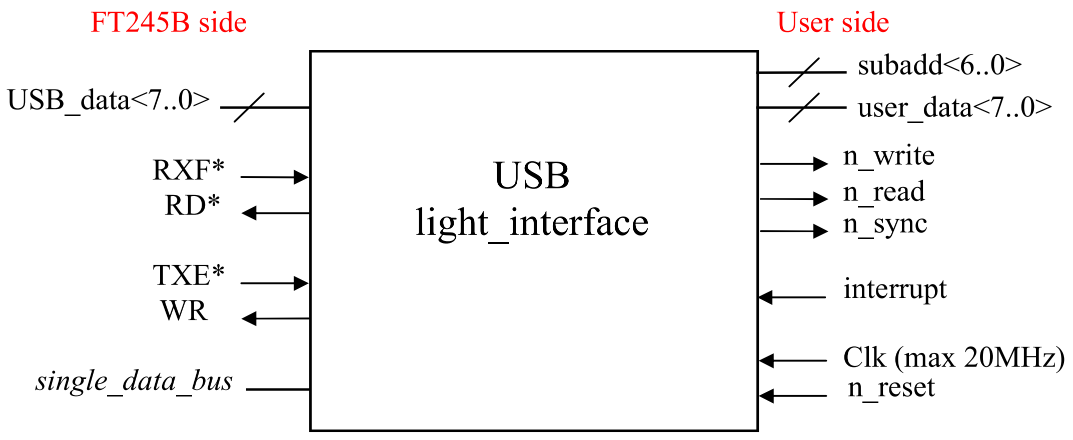


Figure 4 : simplified interface block.

The interface block appears on Figure 4, and its internal block diagram on Figure 5.

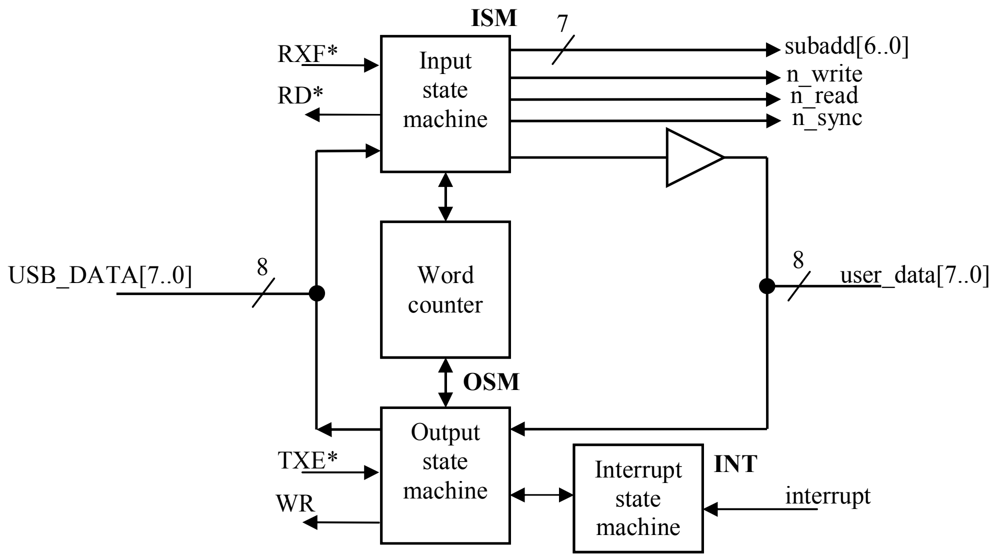


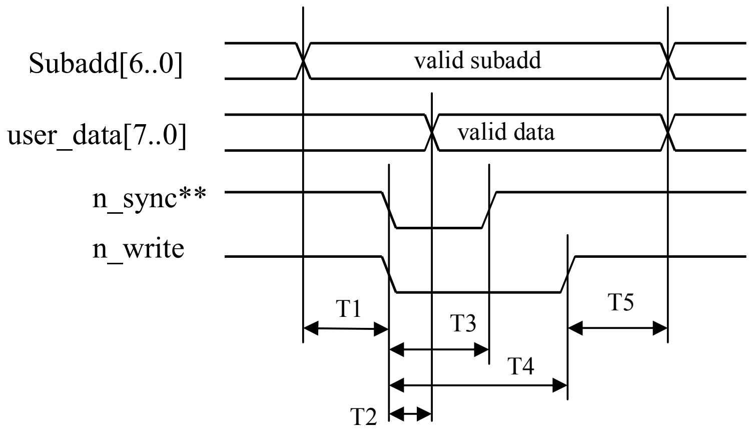
Figure 5 : simplified interface block diagram.

We see in the schematics that two machines directly linked to the FT245B appear in the block. A third machine permits managing the interrupts. The maximum clock frequency to send to the interface is 20MHz in order to respect the temporal constraints of the FT245B. The period T mentioned in the tables is therefore at least 50ns.

The functioning principle is the following :

- When a data byte arrives in the RX FIFO of the USB, the RXF* flag signals it to the ISM input state machine. The latter is destined to manage all of the transfers from the USB towards the peripheral, therefore the user world. It manufactures from the USB frames the user_data[7..0] and subadd[6..0] vectors, as well as the two n_write and n_read orders. In parallel, ISM will of course generate the reading of the USB's RX FIFO.

- When this ISM machine is in its idle state, it searches for a header. If the byte received is not a header, it produces a pulse on its « header_error » output and returns to the idle state. This pulse is immediately transmitted as a prioritary interrupt to the OSM machine which sends an interrupt frame towards the USB with the R/W bit at 0, all bits of the data byte at 0 except bit 0. The sending of interrupts for a header error is then inhibited until the reception of the next header, but the « header_error » output continues to signal the errors.



Time	Description	Typ	Unit
T1	Subadd valid to n_write active	1.5 x T*	ns
T2	n_write active to data valid	5	ns
T3	n_sync active pulse width	T*	ns
T4	n_write active pulse width	2 x T*	ns
T5	n_write inactive to subadd and data invalid	T/2	ns

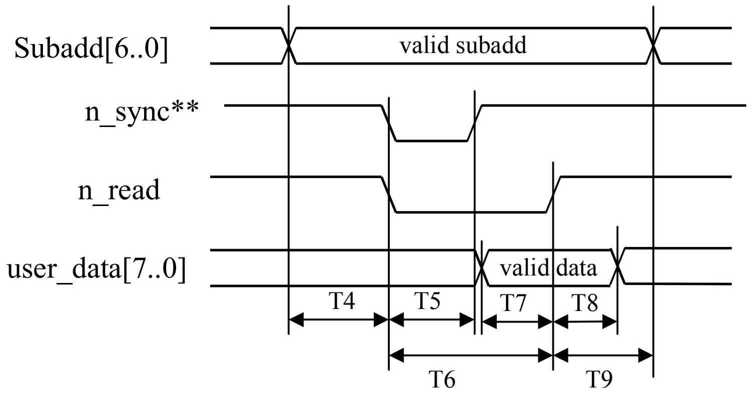
* T is the clock period sent to the interface (>50ns)

** n_sync is only present during the first write pulse of a frame.

Figure 6 : chronogram of a write operation.

- If the frame is a write request, then the sub-address will be presented on the subadd[6..0] bus, the data on the user_data [7..0] bus and the n_write signal triggered successively for each of the presented bytes (see Figure 6). The n_sync signal is presented at the same time as the first n_write and rises one clock period earlier in order to be able to synchronize the block access towards wide registers or self-incremented sub-addresses. The peripheral must use the n_write rising edge to memorize the data and to post-increment if necessary a RAM (NTA type) address counter for example. Thus, the n_sync signal can be sent directly on the n_clear or n_load inputs of a counter or of a shift register and the n_write on the clock of the same counter or register. This sequence will end when the total number of bytes to write (limited to 256) has been attained.

- If the frame is a read demand, then the number of bytes will be stored and utilized by the OSM read machine to present the sub-address on the subadd[6..0] bus, and activate the n_read signal successively for each one of the required bytes (see Figure 7). The n_sync signal is activated in the same fashion as with writing. The Data is memorized in the interface block on the rising edge of n_read, and the peripheral can thus utilize this edge to post-increment a RAM (type NTA) address counter for example. This sequence will end when the total number of bytes to read (limited to 65536) has been attained. In parallel, OSM will generate the write in the TX FIFO of the USB.



Time	Description	Min	Typ	Unit
T4	Subadd Valid to n_read Active		16 x T*	ns
T5	n_sync Pulse Width		T*	ns
T6	n_read Pulse Width		2 x T*	ns
T7	data Valid to n_read Inactive	T*		ns
T8	n_read Inactive to data Invalid	0		ns
T9	n_read Inactive to Subadd Invalid		T*	ns

* T is the clock period sent to the interface (>50ns)

** n_sync is only present during the first read pulse of a frame.

Figure 7 : chronogram of a standard read operation.

- The USB_data[7..0] bus being shared between the inputs and the outputs, the OSM output machine will always have to give priority of access to the ISM input machine, except for interrupts.
- In the case in which the ISM machine detects an error on the trailer of the frame in reception, which could flag a loss of synchronization of the transfers, it emits a pulse on its « trailer_error » output and immediately transmits a prioritary interrupt to the OSM machine which sends an interrupt frame towards the USB with the R/W bit at 0, all bits of the data byte at 0 except bit 1.

- If the interface receives an interrupt request from the peripheral, it sends an interrupt frame towards the USB with the R/W bit at 0 and the data byte equal to 0x4D.

The format of the interrupt with the simplified interface is the following (Tab 3a) :

Byte\bit	b7	b6	b5	b4	b3	b2	b1	b0
Header	1	0	1	0	1	0	1	0
Control 1	0	0	0	0	0	0	0	0
Control 2	0	Subadd						
Data	Status byte							
Trailer	0	1	0	1	0	1	0	1

Tab 3a : format of an interrupt with the simplified interface.

It involves a frame with the last sub-address utilized for an interrupt following a transmission error (header or trailer) or the sub-address 0x7F for an interrupt generated by the user. The R/W bit in return equals 0, which is impossible during a normal reading (where it equals 1). The data byte contains a status word. Table 3b describes the content of this status byte :

Type	b7	b6	b5	b4	b3	b2	b1	b0
Header error	0	0	0	0	0	0	0	1
Trailer error	0	0	0	0	0	0	1	0
User int	0	1	0	0	1	1	0	1

Tab 3b : content of the status byte of an interrupt with the simplified interface.

The interrupts following a transmission error (header or trailer) will be recognized by their sub-address which is different from 0x7F. The bit 0 flags in this case a header error and the bit 1 a trailer error. The status byte of the user interrupt is invariable and equals 0x4D.

We can imagine utilizing this interface with a single data bus, the one linked to the FT245B (USB_data[7..0]). The latter is in fact tri-state, and can be sent directly towards the blocks of the peripheral. One must therefore however make sure that this bus is constantly left floating outside of readings by the USB. This explains the presence of the « single_data_bus » signal at the interface input side :

Value of the « single_data_bus » bit	Usage of the data bus
0	A bus on the USB side and a bus on the peripheral side. The two are separate.
1	A single bus common to the USB and to the peripheral (USB_data[7..0]).

With this version of the interface, there is no handshake with the peripheral. In order to be able to manage the slow peripherals, an « n_wait » input was added to be able to access the latter. It is the version referred to as « standard » of the interface described in Chapter 4.

Moreover, a continuous read mode can be necessary for certain detector applications. This evolution is presented in Chapter 5 which describes the version of the interface referred to as « extended ». It also permits transmitting information with the interrupts generated by the user.

4. Standard interface : usb_interface.

This version of the interface is an evolution of usb_light_interface which makes its utilization more general and permits working with slow peripheral, for example the EEPROM or the microcontroller interfaces. For the user, it is identical to the usb_light_interface, with the exception of the peripheral side n_wait input (see Figures 8 and 9) which permits handshaking between the interface and the latter.

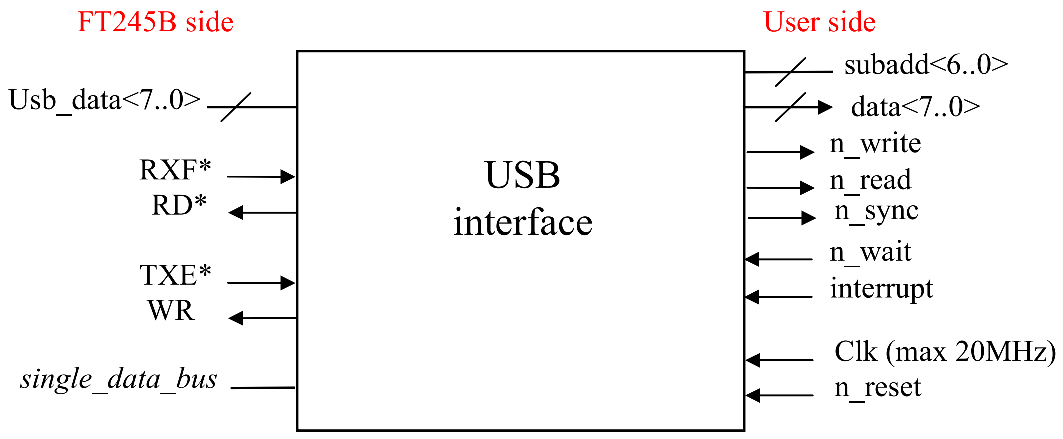


Figure 8 : standard interface block.

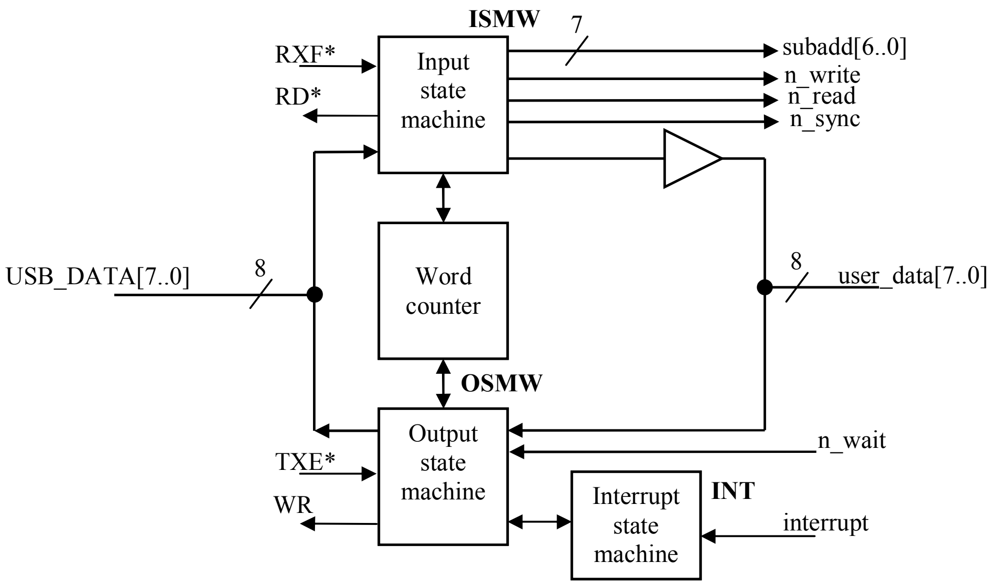
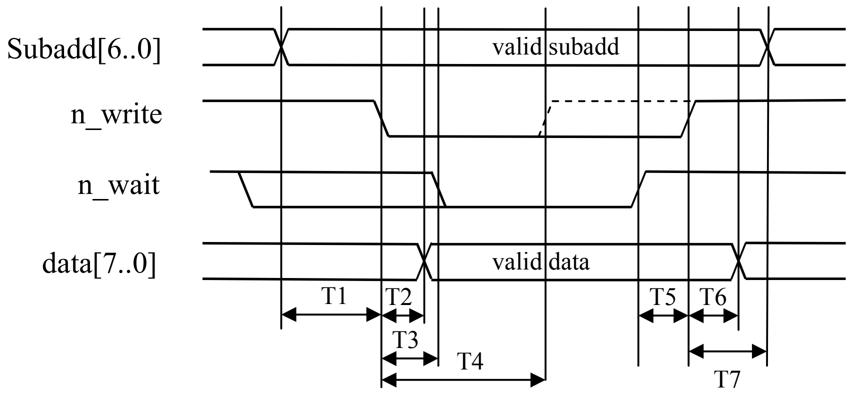


Figure 9 : standard interface block diagram.

The n_wait signal permits blocking the response of the interface as long as the peripheral has not validated the write or read request. For the user, it suffices to pull n_wait to zero as long as necessary. The end of the byte write or read will only take place on the rise of n_wait (cf Figures 10 and 11).

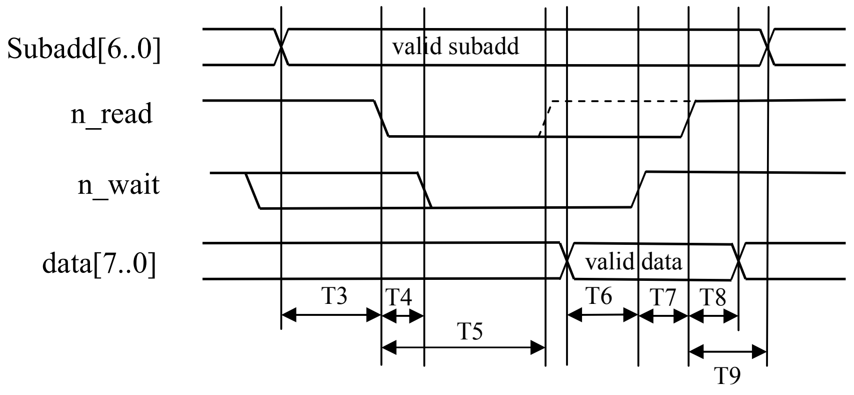
If the n_wait signal is not utilized, it must be connected to VCC.



Time	Description	Min	Max	Unit
T1	Subadd Valid to n_write Active	T*	2 x T*	ns
T2	n_write Active to data Valid	0	10	ns
T3	n_write Active to n_wait Active	-	T*	ns
T4	n_write Pulse Width without n_wait		2 x T*	ns
T5	n_wait Inactive to n_write Inactive	0	T*	ns
T6	n_ write Inactive to data Invalid	10	20	ns
T7	n_ write Inactive to Subadd Invalid	10	20	ns

* T is the clock period sent to the interface (>50ns)

Figure 10 : chronogram of a write operation delayed by n_wait.



Time	Description	Min	Max	Unit
T3	Subadd Valid to n_read Active		T*	ns
T4	n_read Active to n_wait Active	0	T*	ns
T5	n_read Pulse Width without n_wait		2 x T*	ns
T6	data Valid to n_wait Inactive	T*		ns
T7	n_wait Inactive to n_read Inactive	0	T*	ns
T8	n_read Inactive to data Invalid	0		ns
T9	n_read Inactive to Subadd Invalid	T*		ns

* T is the clock period sent to the interface (>50ns)

Figure 11 : chronogram of a read operation delayed by n_wait.

5. Extended interface: usb_full_interface.

This version of the interface permits a continuous read mode triggered by the peripheral for the detector applications with random data flow. It comprises the same elements as the standard interface, to which we added an internal register which permits authorizing the extended read mode, an input signal (read_req) which permits triggering a read on the request of the peripheral, and an output signal (busy) which signals to the peripheral that the interface is busy transmitting data (see Figure 12), even for the standard access. The interrupt input remains available in a normal fashion in the case of continuous reading. This version also offers the sending of status bits by the peripheral inside the interrupts.

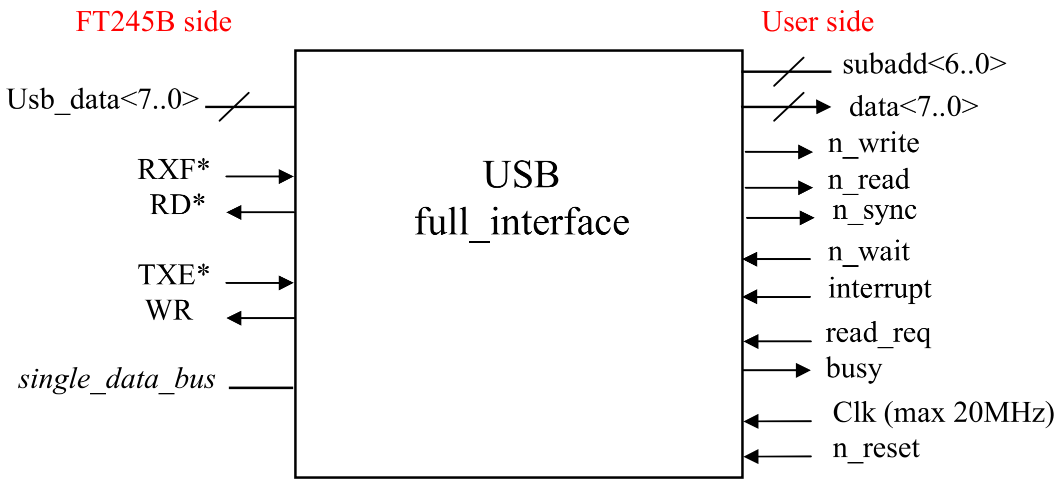


Figure 12 : extended interface block.

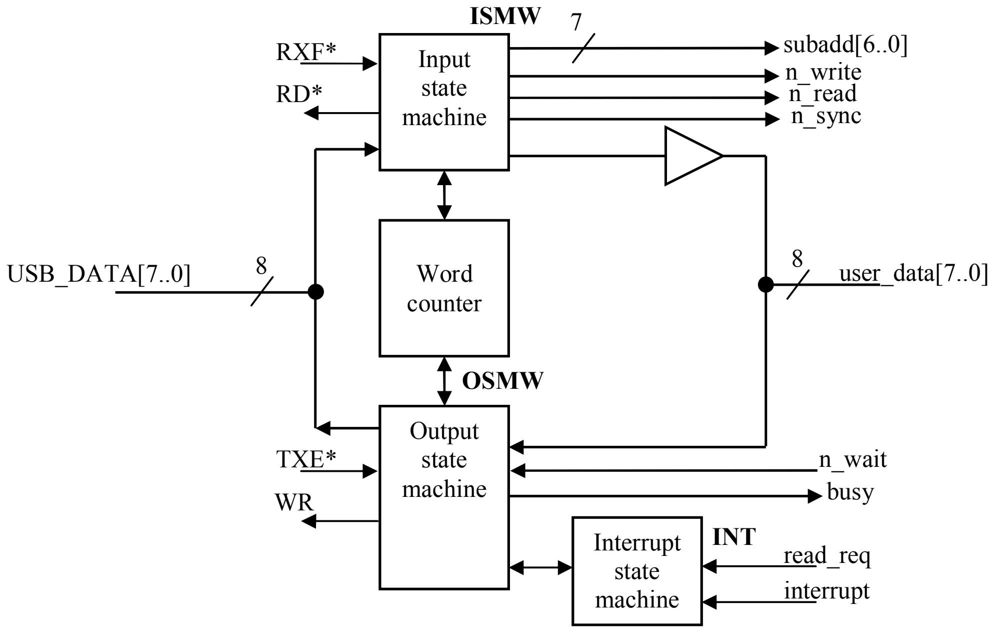
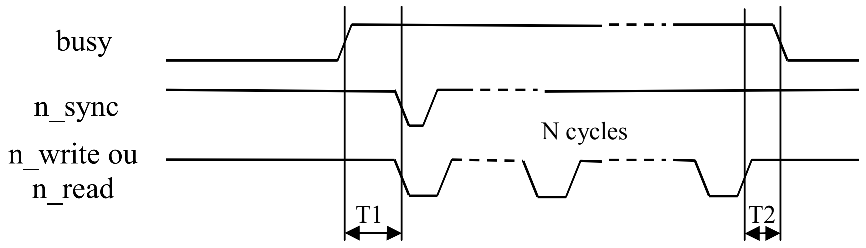


Figure 13 : block diagram of the extended interface.

The internal block schematics of the extended interface appears on Figure 13 and the corresponding simplified chronogram for the write or read operations on Figure 14.

The busy signal is used during all of the types of access. During standard write or read operations, it surrounds the phase of access towards the peripheral (see Figure 14).



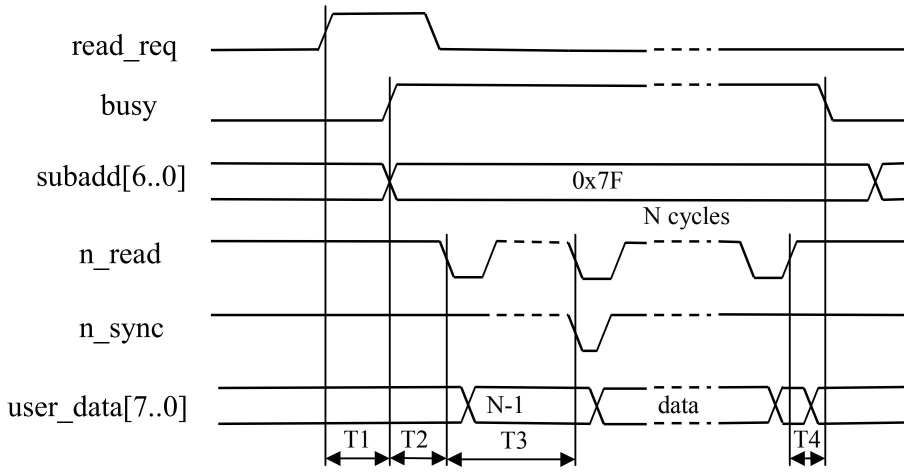
Time	Description	Min	Max	Unit
T1	Busy Active to n_write or n_read Active	T*		ns
T2	Last n_write or n_read Active to busy Inactive	0		ns

* T is the clock period sent to the interface (>50ns)

Figure 14 : chronogram of a standard write or read operation.

The functioning principle of the read_request is the following (see Figure 15) :

- The read_req signal is an extended read request, which will be interpreted on its rising edge.
- In response, the busy signal will rise to 1.



Time	Description	Min	Typ	Unit
T1	read_req Rising Edge to busy and subadd Active		2 x T*	ns
T2	busy Active to first n_read Active		T*	ns
T3	first n_read Active to data n_read Active		44 x T*	ns
T4	last n_read Active to busy Inactive		10 x T*	ns

* T is the clock period sent to the interface (>50ns)

Figure 15 : chronogram of an extended read operation.

- The interface will then send a first n_read to read on the data bus of the peripheral the number N-1 of bytes to send towards the USB. The user_data[7..0] bus will of course be utilized if the single_data_bus input is at 0 and if not, it will be the usb_data[7..0] bus.

- Next, the interface will send N n_read read orders to the peripheral as during a standard read demand by the USB. N_sync will also be present with the first among them.
- When the N bytes have been transmitted towards the USB, the busy signal will fall again to 0 to signal to the peripheral that the interface has finished the transfer of the block and is ready to receive a new extended read request.
- The write access remains always prioritary in relation to the extended read requests in order to be sure that the program in execution on the host machine can continue to keep the hand on the operations in course.

The format of the extended read frame is presented in Table 4. The difference with a standard read return frame is in the Control 2 word where the sub-address is fixed at 0x7F as for the user interrupt. It is presented on the Subadd bus so that the peripheral can recognize that it is indeed a response to the read_req.

The number of data bytes in a frame is limited to 256. This is due to the fact that we only have 8 bits for coding it. But as the peripheral can perform as many extended read requests as necessary one after another, one can attain with this the maximum theoretical speed from the USB link, or 1,1MByte/s.

Byte\bit	b7	b6	b5	b4	b3	b2	b1	b0
Header	1	0	1	0	1	0	1	0
Control 1	Nb of words – 1 (N – 1)							
Control 2	1	Subadd = 0x7F						
Data x N*	Data byte x N (N < 256)							
Trailer	0	1	0	1	0	1	0	1

Tab 4 : format of an extended read frame.

When one is in the extended mode, all of the interrupts continue to function in the same manner. They will be emitted at the end of the extended read cycle, with the same order of priority as described in Chapter 2. However, the format of the user interrupt is more developed, because the peripheral can add information in the status byte.

The interrupt format with the extended interface is the following (Tab 5a) :

Byte\bit	b7	b6	b5	b4	b3	b2	b1	b0
Header	1	0	1	0	1	0	1	0
Control 1	0	0	0	0	0	0	0	0
Control 2	0	Subadd						
Data	Status byte							
Trailer	0	1	0	1	0	1	0	1

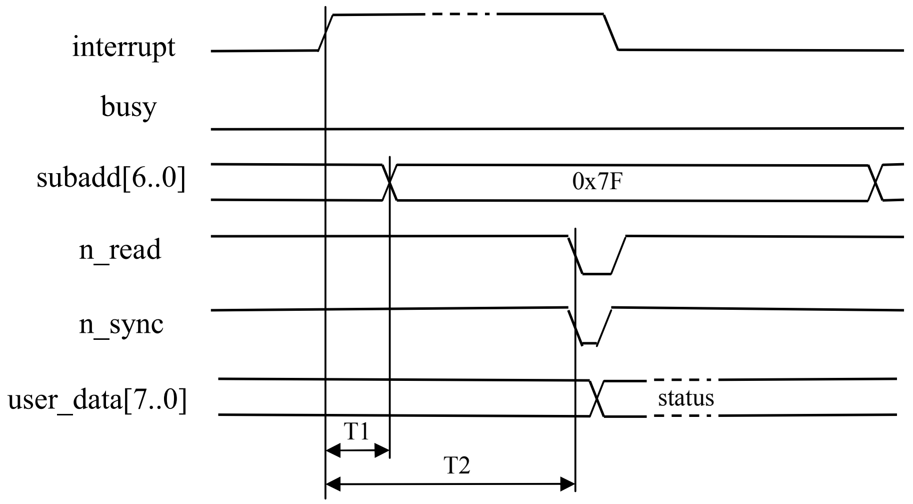
Tab 5a : format of an interrupt with the extended interface.

It involves a frame with the last sub-address utilized for an interrupt following a transmission (header or trailer) error or the sub-address 0x7F for an interrupt generated by the user. The R/W bit in return equals 0, which is impossible during a normal reading (where it equals 1). The data byte contains a status word. Table 5b describes the content of this status byte :

Type	b7	b6	b5	b4	b3	b2	b1	b0
Header error	R	R	R	R	R	R	0	1
Trailer error	R	R	R	R	R	R	1	0
User int	X	X	X	X	X	X	X	X

Tab 5b : content of the status byte of an interrupt with the extended interface.

The interrupts following a transmission (header or trailer) error will be recognized by their sub-address which is different from 0x7F. The bit 0 flags in this case a header error and bit 1 a trailer error. The bits marked « R » are reserved. For an interrupt generated by the user (see Figure 16), the peripheral will be able to furnish the data byte, which will be requested from it by the n_read order as during a standard read. This status request following the interrupt will be recognized by the user thanks to its sub-address 0x7F presented on the subadd bus and by the absence of the busy signal during the n_read.



Time	Description	Min	Typ	Unit
T1	interrupt Rising Edge to subadd Active		2 x T*	ns
T2	interrupt Rising Edge to n_read Active		20 x T*	ns

* T is the clock period sent to the interface (>50ns)

Figure 16 : chronogram of a user interrupt with the extended interface.

Note :

- the sub-address 0x7F is reserved for the interrupts and must therefore not be utilized by the program running on the host machine.
- the sub-address 0x7E is reserved for the interface version number. A reading of this sub-address will give a byte to translate into 2 quartets (0x23 for version 2.3). This number will be used by the software to recognize it and choose the adapted library. The oldest versions do not comprise this version number.

6. Implementation of the interface.

Figure 17 shows the implementation of the components of the interface on the board. The correspondence between the names of the different signals linked to the interface on this schematic and those of the interface block in the ALTERA is described in Table 6.

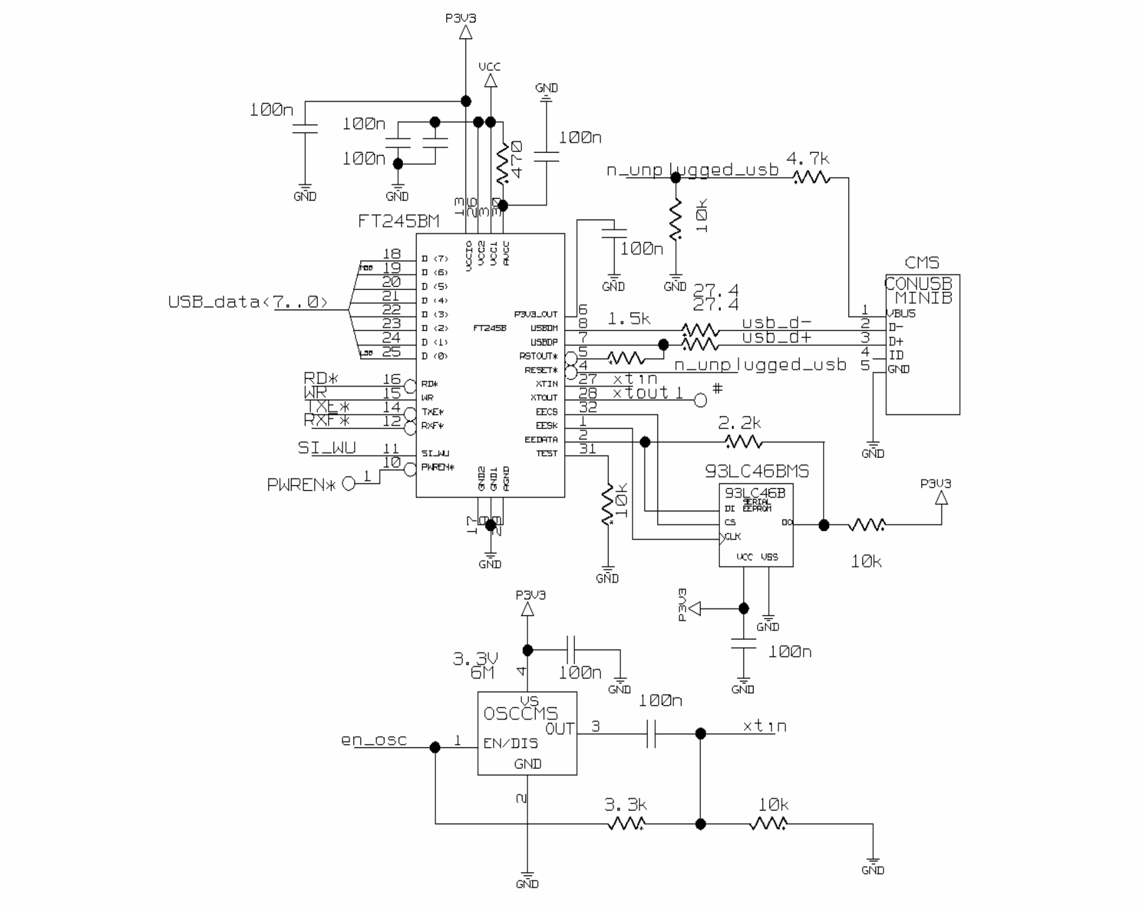


Figure 17 : implementation of the interface on the board.

The signal PWREN* serves to signal when the USB is active (at 0).

Pulling the SI_WU to 0 serves to force the emission of the data contained in TX FIFO in the next USB « bulk-in request » whatever the number of bytes present. This could serve to improve the read debit in certain cases. By default, pull this pin to VCCIO (here P3V3).

The en_osc signal is used to start or stop the oscillator for the FT245B. It must be at 3.3V to authorize the functioning and in this case, it puts the average level of xtin at +2.5V through the resistor bridge, as required in the FT245B data sheet. If en_osc is at GND, xtin then well pulled down to GND. If we use a strap, we must implement the oscillator as in Figure 18. If en_osc uses a level other than 3.3V, it must be taken into account to connect to the oscillator and to the bridge for the +2.5V : there is an example with +5V in Figure 19.

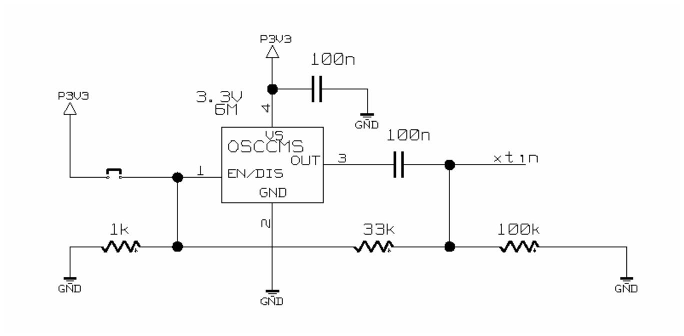


Figure 18 : example of an implementation of the oscillator en/dis input from a strap at + 3.3V.

The EEPROM, which is empty at the beginning, will by loaded by USB during the first connection and will then furnish the information which was previously stored inside during the future connections. **Note :** different versions of this EEPROM exist, some of which, such as he Microchip 93C46B, do not function under 3.3V (with Microchip, one must use 93LC46B).

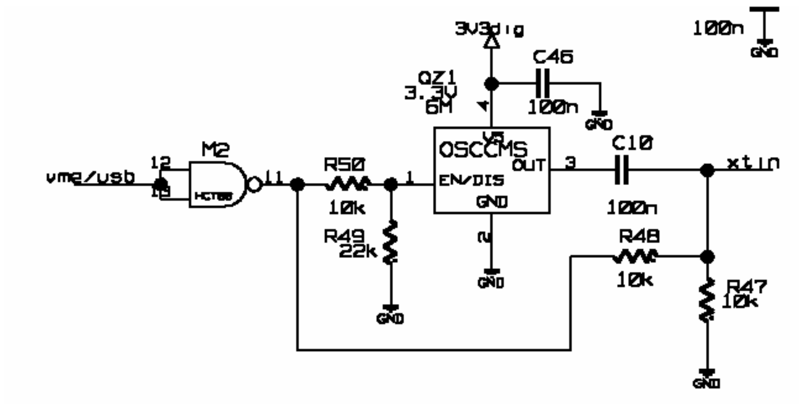


Figure 19 : example of an implementation of the oscillator en/dis input from a signal at +5V.

Board	ALTERA
RD*	n_rd
WR	wr
TXE*	n_txe
RXF*	n_rxf

Tab 6 : correspondence of the signal names.

Figure 20 describes the two possible implementations of the simplified interface block inside ALTERA depending on the number of data buses utilized. The external input-outputs correspond to the signals linked to FT245B.

- If we have a USB-side data bus which is independent from the peripheral side data bus, we must use the first solution (single_data_bus at GND).
- If we have a single common data bus, we must use the second solution (single_data_bus at VCC).
- The usb_clk signal must have a maximum frequency of 20MHz. It can be the 6MHz clock of the FT245B. To ensure the maximum baud rate of 1,1MByte/s, the frequency must be greater than 10MHz.

- The n_reset (negative pulse) signal entirely resets the interface. It must be done at least once during the power-up sequence.
- The interrupt signal is a positive pulse of an open length, memorized on its rising edge.

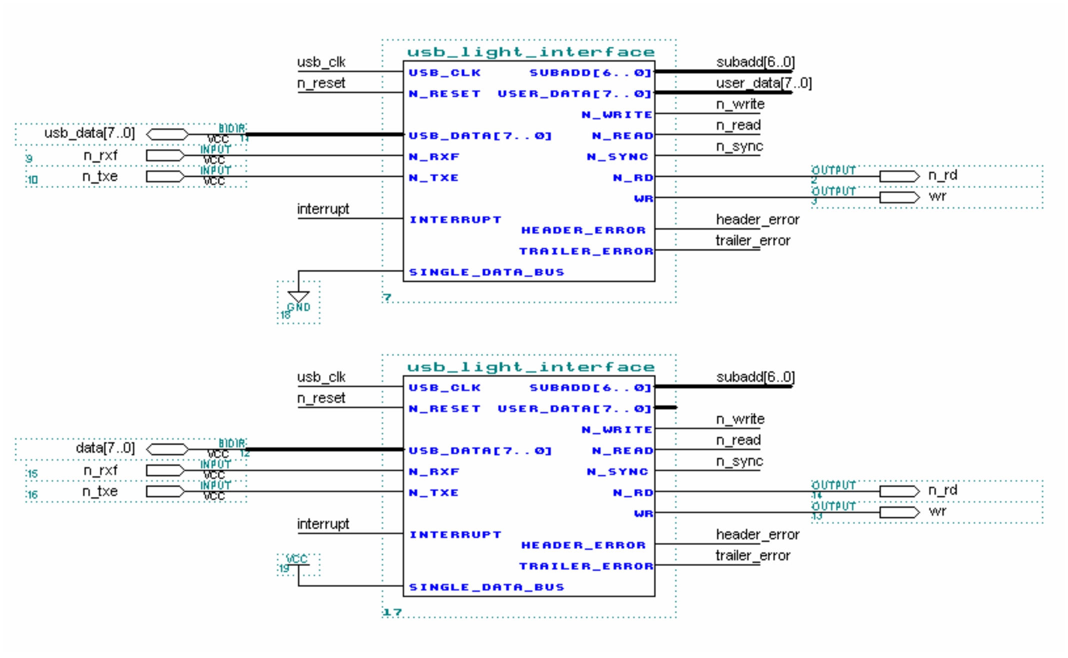


Figure 20 : implementation of the simplified interface block in ALTERA.

- The header_error signal is a positive pulse lasting one period of usb_clk.
- The trailer_error signal is a positive pulse lasting 2 periods of usb_clk.

The files necessary for the compilation of the simplified block comprise :

- the *usb_light_interface* graphic page and its symbol
- the *word_counter* graphic page and its symbol
- the three state machiness *ism_machine*, *osm_machine* and *int_machine* and their respective symbols.

The files necessary for the compilation of the standard block comprise :

- the *usb_interface* graphic page and its symbol
- the *word_counter* graphic page and its symbol
- the three state machiness *ismw_machine*, *osmw_machine* and *int_machine* and their respective symbols.

The files necessary for the compilation of the extended block comprise :

- the *usb_full_interface* graphic page and its symbol
- the *word_counter, version_nb* and *byte_builder* graphic pages and their respective symbols
- the four state machiness *ismw_machine*, *osmw_machine, read_req_machine* and *int_machine* and their respective symbols.

These files are available in Maxplus2 and Quartus versions.