

# **USB\_DEVTRSAC**

## **User guide**

author: Maksim Shaklunov

email: [maxseeking@ya.ru](mailto:maxseeking@ya.ru)

2014

## 1 Pins

Name	Dir	Description
System		
<b>clk_4xrate</b>	input	Clock. Frequency must be 48 MHz for full-speed device and must be 6 MHz for low-speed device.
<b>rst0_async</b>	input	0: asynchronous reset. 1: idle.
<b>rst0_sync</b>	input	0: synchronous reset. It can be used by software reset. 1: idle.
Transceiver		
<b>drx_plus</b>	input	Receiver single-ended D+ signal.
<b>drx_minus</b>	input	Receiver single-ended D- signal.
<b>drx</b>	input	Receiver differential signal.
<b>dtx_plus</b>	output	Transmitter single-ended D+ signal
<b>dtx_minus</b>	output	Transmitter single-ended D- signal
<b>dtx_oe</b>	output	0: idle. 1: transmitter output enable.
Transaction (synchronous with <b>clk_4xrate</b> )		
<b>trsac_req</b>	output	Host request. 1: ACTIVE. USB_DEVTRSAC received or need to send data for particular endpoint. <b>trsac_ep</b> , <b>trsac_type</b> have valid values. 0: OK. Transaction is successful completed. 2: FAIL. Transaction failed.
<b>trsac_reply</b>	input	Device reply. 0: ACK/DATA. USB_DEVTRSAC sends ACK packet at OUT/SETUP transaction and sends DATA packet at IN transaction. 1: NAK. USB_DEVTRSAC sends NAK packet. 2: STALL. USB_DEVTRSAC sends STALL packet.
<b>trsac_ep</b>	output	Endpoint address.
<b>trsac_type</b>	output	Transaction type. 0: SETUP. 1: OUT. 2: IN.
<b>rfifo_rd</b>	input	RFIFO is place where USB_DEVTRSAC save data received from host. RFIFO is 64 byte length. 0: idle. 1: RFIFO read enable.
<b>rfifo_empty</b>	output	0: RFIFO is not empty. 1: RFIFO is empty.
<b>rfifo_rdata</b>	output	RFIFO data byte.
<b>wfifo_wr</b>	input	RFIFO is place where USB_DEVTRSAC read data which will be sent to host. WFIFO is 64 byte length. 0: idle. 1: WFIFO write enable.

Name	Dir	Description
wfifo_full	output	0: WFIFO is not full. 1: WFIFO is full.
wfifo_wdata	input	WFIFO data byte.
Endpoint (synchronous with <b>clk_4xrate</b> )		
ep_enable	input	Bus has 15 lines each line from 1 to 15 correspond endpoint from 1 to 15. 0: disable transactions and reset toggle bit for particular endpoint. 1: enable transactions for particular endpoint.
ep_isoch	input	Bus has 15 lines each line from 1 to 15 correspond endpoint from 1 to 15. 0: idle. 1: enable isochronous transaction behavior for particular endpoint.
ep_intnorettry	input	Bus has 15 lines each line from 1 to 15 correspond endpoint from 1 to 15. 0: idle. 1: enable interrupt with no retry transaction behavior for particular endpoint.
Start of frame (synchronous with <b>clk_4xrate</b> )		
sof_tick	output	0: idle. 1: USB_DEVTRSAC received SOF packet. High pulse duration is one clock cycle.
sof_value	output	Latched 11-bit SOF value.
Device (synchronous with <b>clk_4xrate</b> )		
device_state	output	b_000: POWERED state. b_001: DEFAULT state. b_010: ADDRESSED state. b_011: CONFIGURED state. b_1xx: SUSPENDED state.
device_speed	input	0: low-speed. 1: full-speed.
device_wakeup	input	0: idle. 1: enable wakeup signaling.

## 2 Purpose

USB\_DEVTRSAC accelerate USB Full/Low-speed peripheral development. It covers USB protocol up to transaction level:

- USB device states control;
- transaction control;
- data toggle bit managing;
- packet encoding/decoding;
- remote wakeup signaling.

Using USB\_DEVTRSAC, device designer focus on transfers and requests processing without implementing transaction level of USB protocol.

The figure 2 demonstrates functional parts of the abstract device, which use USB\_DEVTRSAC: device specific logic (endpoints set, standard/class-specific request processing), USB\_DEVTRSAC, transceiver.

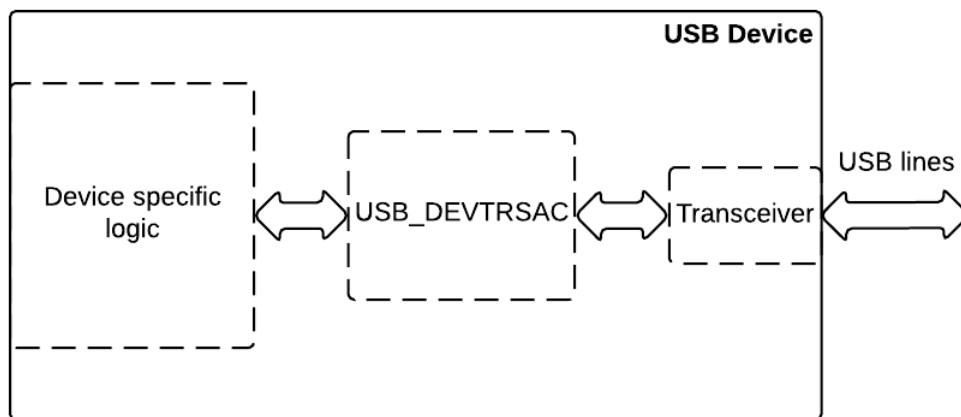


Figure 2 – USB device.

### 3 Transaction control

All USB transfers consist of one or multiple transactions. Transactions can be divided in three general types: IN, OUT, SETUP.

In all transactions USB\_DEVTRSAC manages endpoints toggle bit by itself. But after standard request SetInterface() device must reset toggle bit of endpoints related to the new alternative setting. USB\_DEVTRSAC has no information about alternative settings and related endpoints. Because of that, device specific logic must reset toggle bit with **ep\_enable** bus: low pulse at line that correspond endpoint.

#### 3.1 IN transaction

Next phases take place in IN transaction of control, bulk, interrupt transfers:

- host sends TOKEN IN packet;
- device sends DATA packet. If endpoint is unable to send data – device sends NAK packet, if endpoint halted – device sends STALL packet;
- host sends ACK packet if it has received DATA packet.

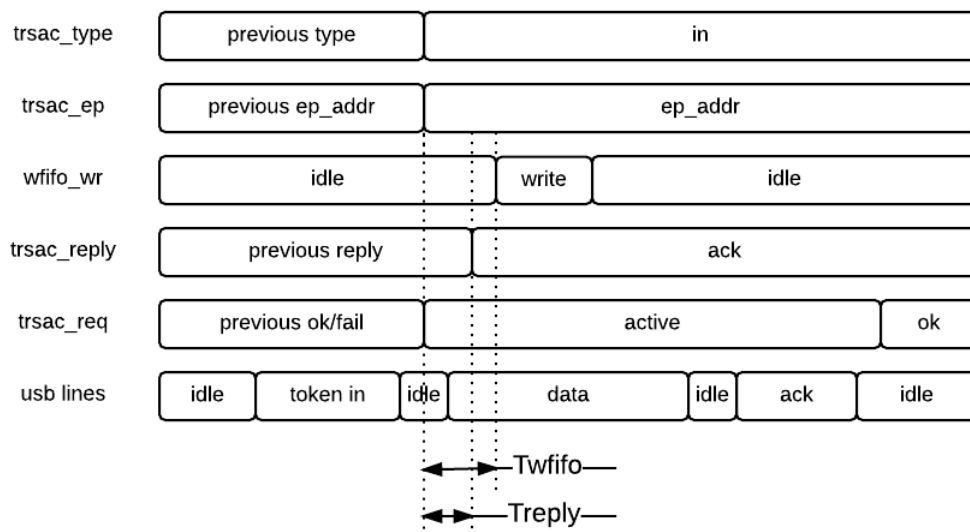


Figure 3.1 – IN transaction.

The figure 3.1 demonstrates example of the IN transaction. **Treply**=30 clock cycles is time during which device specific logic have to answer with **trsac\_reply**. **Twfifo**=50 clock cycles is time during which device specific logic have to begin to write first data byte in the WFIFO.

At the first phase USB\_DEVTRSAC receives TOKEN IN packet and validates token address versus device address. If it is successful, USB\_DEVTRSAC sets:

- **trsac\_req**=**ACTIVE** indicating beginning of transaction;
- **trsac\_ep**=**EP\_ADDR** indicating endpoint address;
- **trsac\_ep**=**IN** indicating transaction type.

At the second phase USB\_DEVTRSAC begins to send SYNC field of the answer packet. And after Treply USB\_DEVTRSAC samples **trsac\_reply** signal:

- **trsac\_reply=ACK/DATA**, it begins to send PID field of the DATA packet. And after Twfif USB\_DEVTRSAC begins to send data from the WFIFO. If WFIFO has no data, zero length DATA packet will be sent. At the third phase USB\_DEVTRSAC waits ACK packet from host. And if USB\_DEVTRSAC receives packet, it sets **trsac\_req=OK**; if does not, it sets **trsac\_req=FAIL**;
- **trsac\_reply=NAK** it sends NAK packet. After that USB\_DEVTRSAC sets **trsac\_req=OK**;
- **trsac\_reply=STALL** it send STALL packet. After that USB\_DEVTRSAC sets **trsac\_req=OK**.

Next phases take place in IN transaction of the isochronous transfer:

- host sends TOKEN IN packet;
- device sends DATA packet.

Difference from previous transfers is that isochronous transfer have no handshake. So device specific logic must always answer with **trsac\_reply=ACK/DATA**.

### 3.2 OUT transaction

Next phases take place in IN transaction of control, bulk, interrupt transfers:

- host sends TOKEN OUT packet;
- host sends DATA packet;
- device sends ACK packet. If endpoint is unable to receive data – device sends NAK packet, if endpoint halted – device sends STALL packet.

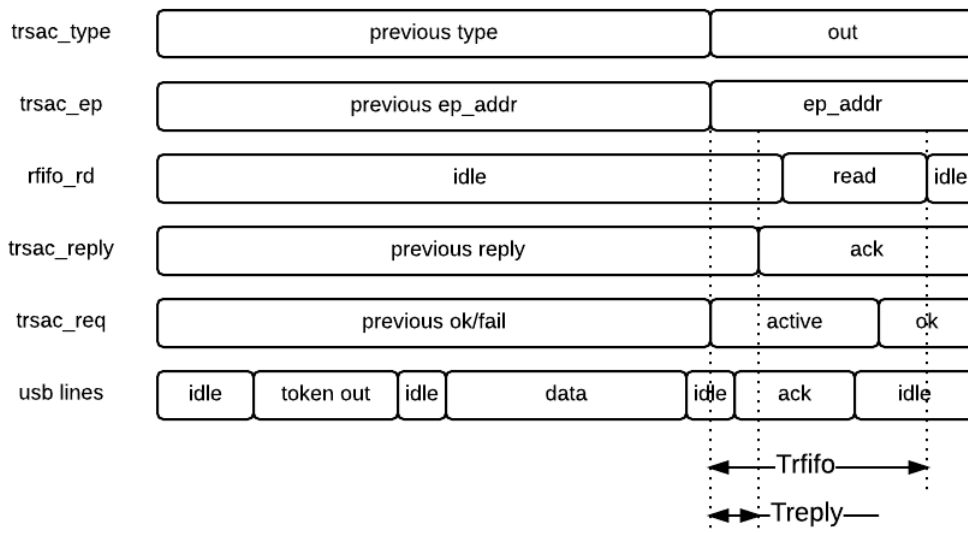


Figure 3.2 – OUT transaction.

The figure 3.2 demonstrates example of the OUT transaction. Treply=30 clock cycles is time during which device specific logic have to answer with **trsac\_reply**. Trfifo=120 clock cycles is time during which device specific logic have to read all data bytes from the RFIFO before new transaction will be began.

At the first phase USB\_DEVTRSAC receives TOKEN OUT packet and validates token address versus device address.

At the second phase USB\_DEVTRSAC receives DATA packet and sets:

- **trsac\_req=ACTIVE** indicating beginning of transaction;
- **trsac\_ep=EP\_ADDR** indicating endpoint address;
- **trsac\_ep=OUT** indicating transaction type.

Device specific logic begin to read data from RFIFO. If USB\_DEVTRSAC received zero-length DATA packet then RFIFO has no data and **rfifo\_empty=1**.

At the third phase USB\_DEVTRSAC begins to send SYNC field of the answer packet. After Treply USB\_DEVTRSAC samples **trsac\_reply** signal:

- **trsac\_reply=ACK/DATA**, it sends ACK packet. After that USB\_DEVTRSAC sets **trsac\_req=OK**;
- **trsac\_reply=NAK** it sends NAK packet. After that USB\_DEVTRSAC sets **trsac\_req=OK**;
- **trsac\_reply=STALL** it sends STALL packet. After that USB\_DEVTRSAC sets **trsac\_req=OK**.

Next phases take place in IN transaction of the isochronous transfer:

- host sends TOKEN IN packet;
- host sends DATA packet.

Difference from previous transfers is that isochronous transfer have no handshake. So device specific logic just read data from the RFIFO.

### 3.3 SETUP transaction

Setup transaction can only be in the control transfer:

- host sends TOKEN OUT packet;
- host sends DATA packet;
- device always sends ACK packet if it had no packet errors.

USB\_DEVTRSAC has same behavior as in OUT transaction.

## 4 Standard/class-specific requests and enumeration

USB\_DEVTRSAC processing only three standard requests at default pipe (endpoint 0):

- **SetAddress().** USB\_DEVTRSAC use this request for managing device address and for transition to ADDRESSED state. This request don't pass on transaction interface (device specific logic don't see request).
- **SetConfiguration().** This request pass on transaction interface and device specific logic must processing it. When status stage of this request is successful, USB\_DEVTRSAC goes to CONFIGURED state and reset all endpoints toggle bits.
- **ClearFeature(endpoint halt).** This request pass on transaction interface and device specific logic must processing it. When status stage of this request is successful, USB\_DEVTRSAC reset toggle bit of the corresponded endpoint.

All other requests pass on transaction interface for device specific logic processing.

While enumeration process, USB\_DEVTRSAC indicates device state with **device\_state** signal:

- After system reset USB\_DEVTRSAC is placed in POWERED state.
- After USB reset USB\_DEVTRSAC is placed in DEFAULT state. Now it can process endpoint 0 transaction.
- After SetAddress() USB\_DEVTRSAC is placed in ADDRESSED state. Now device has unique address
- After SetConfiguration() USB\_DEVTRSAC is placed in CONFIGURED state. Now it can process transactions for endpoints from 1 to 15. Device specific logic must setup next signals for endpoints 1-15:
  - **ep\_enable[15:1].** USB\_DEVTRSAC enable/disable corresponded endpoint transactions;
  - **ep\_isoch[15:1].** USB\_DEVTRSAC apply isochronous transaction behavior for corresponded endpoint.
  - **ep\_intnorettry[15:1].** USB\_DEVTRSAC apply interrupt transaction behavior with no retry mechanism for corresponded endpoint. USB\_DEVTRSAC always toggle DATA PID in spite of receiving ACK from host. This is for IN interrupt endpoint only.



5 Remote wakeup

The figure 5 demonstrate remote wakeup signaling. When bus idle continue ~5.5 ms USB\_DEVTRISAC goes in SUSPENDED state. Then if USB\_DEVTRISAC detect **device\_wakeup**=1, it generate ~3 ms K state on USB lines. High pulse duration of the **device\_wakeup** must be at least one clock cycle.

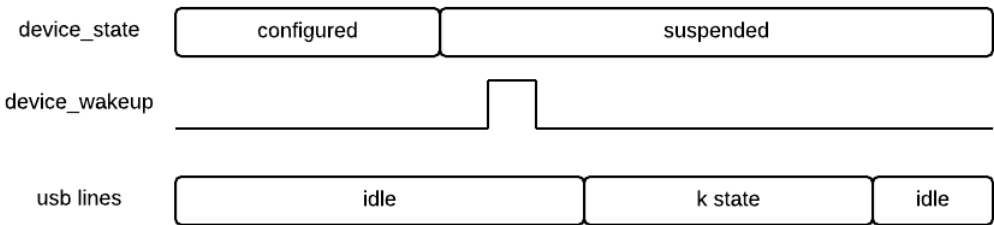


Figure 5 – Remote wakeup signaling.

# 6 Start of frame

The figure 6 demonstrate SOF packet receiving. High pulse duration of the **sof\_tick** is one clock cycle.

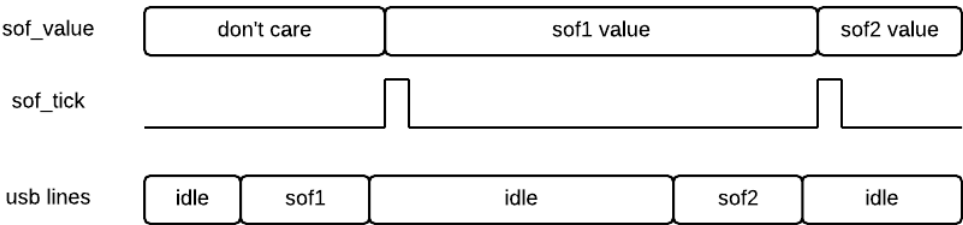


Figure 6 – SOF packet receiving.