**Line listener:**

Rising edge sensitive -> save debugger and device lines into a register to be read on the falling edge.

**Protocol engine:**

* falling edge sensitive
* Start in IDLE

**IDLE:**

Bus direction: debugger -> device

Action: What until line listener detects a HIGH state -> transition to START state.

**START:**

Bus direction: debugger -> device

Action: If next cycle after HIGH = LOW -> transition to read/write state

Else transition to CONNECT.

**CONNECT:**

Bus direction: debugger -> device

Action: Line has to be HIGH for at least 50 cycles. Then wait for two LOW cycle -> transition to IDLE.

**READ/WRITE:**

Bus direction: debugger -> device

Action: Read second bit from line listener to set read or write operation variable. Transition to the ACK state after the 7th bit.

**ACK:**

Bus direction: device -> debugger

Action: (Transition cycle to ACK is used to switch the bus direction; in VHDL: pin <= HIGH\_Z; state <= ACK; result: first cycle of ACK state is already a valid write) Read 3 bits (ACK bits) on the falling edge and check if ACK = OK. If not OK -> transition to IDLE or do dummy read/write depending on protocol type (probably go to IDLE with J-Link Segger)   
else ->  
transition to read or write data depending on bit from read/write state.

**Write data:**

Bus direction: debugger -> device

Action: (Transition cycle to write data is used to switch the bus direction; in VHDL: pin <= HIGH\_Z; state <= WRITE; result: first cycle of WRITE state is already a valid write) Wait for 33 bits of data then -> transition to IDLE.

**Read data:**

Bus direction: device -> debugger

Action: Wait for 33 bits on the falling edge then on the next cycle -> transition to IDLE

**How to handle TrN (Turn around phase):**

Starts on the falling edge before ACK and after reading data and starts on the rising edge after ACK for writing. On the falling edge the before the start of the TRN period set all the control signals for the gate generator signal length start edge and duration. Hold all signals like this till the next falling edge occurs (automatically handled by likely switching these signals in the falling edge triggered protocol engine state machine). In order to start the edge on the rising edge a separate duplicate of the generator with switched edge triggering is need.

Before ACK: One falling edge before the TRN set the control lines for the half cycle gate generator. Hold till next falling edge. (Dedicated half cycle gate generator)

After reading: in the protocol engine state machine set the pin as HIGH Z and switch state to IDLE. Next falling edge state IDLE executes and drives the line. (No dedicated hardware)

Before writing: On the falling edge before the TRN set the control lines for the 3 half cycle gate generator. Hold till next falling edge. (Dedicated half cycle gate generator)´

Use in combination with a multiplexer that switches between drive and HIGHZ depending on the generated gate signal.

**Half cycle gate generator for the HIGH-Z transitions:**

library ieee;

use ieee.std\_logic\_1164.all;

entity half\_or\_full\_cycle\_pulse is

port (

clk : in std\_logic;

trigger : in std\_logic;

out\_signal : out std\_logic

);

end entity;

architecture rtl of half\_or\_full\_cycle\_pulse is

signal trigger\_prev : std\_logic := '0';

signal pulse\_rise : std\_logic := '0';

signal pulse\_fall : std\_logic := '0';

begin

-- Rising edge process: detect trigger rising edge and start pulse

process(clk)

begin

if rising\_edge(clk) then

if (trigger = '1' and trigger\_prev = '0') then

pulse\_rise <= '1';

end if;

if (pulse\_fall=’1’) then

pulse\_rise=’0’;

end if;

trigger\_prev <= trigger;

end if;

end process;

-- Falling edge process: used for ending the half-cycle or full-cycle pulse

process(clk)

begin

if falling\_edge(clk) then

if (pulse\_rise = '1') then

pulse\_fall <= '1';

else

pulse\_fall <= '0'; -- Reset fall flag

end if;

end if;

end process;

-- Output signal is high only while pulse is active

out\_signal <= '1' when (pulse\_rise = '1' and pulse\_fall = '0') else '0';

end architecture;

library ieee;

use ieee.std\_logic\_1164.all;

entity three\_half\_cycle\_pulse is

port (

clk : in std\_logic;

trigger : in std\_logic;

out\_signal : out std\_logic

);

end entity;

architecture rtl of three\_half\_cycle\_pulse is

signal trigger\_prev : std\_logic := '0';

signal half\_cycle\_1 : std\_logic := '0';

signal half\_cycle\_2 : std\_logic := '0';

signal half\_cycle\_3 : std\_logic := '0';

signal pulse\_active\_r : std\_logic := '0';

signal pulse\_active\_f : std\_logic := '1';

begin

-- Rising edge process: start pulse and control half cycles 1 and 3

process(clk)

begin

if rising\_edge(clk) then

trigger\_prev <= trigger;

if (trigger = '1' and trigger\_prev = '0') then

pulse\_active\_r <= '1';

half\_cycle\_1 <= '1';

end if;

if pulse\_active\_r = '1' then

half\_cycle\_3 <= half\_cycle\_2;

end if;

if(pulse\_active\_f = '0') then

half\_cycle\_1 <= '0';

half\_cycle\_3 <= '0';

pulse\_active\_r <= '0';

end if;

end if;

end process;

-- Falling edge process: control half cycle 2

process(clk)

begin

if falling\_edge(clk) then

if pulse\_active\_r = '1' then

half\_cycle\_2 <= half\_cycle\_1;

pulse\_active\_f <= not half\_cycle\_3;

else

half\_cycle\_2 <= '0';

pulse\_active\_f <= '1';

end if;

end if;

end process;

-- Output is high if any half cycle is active

out\_signal <= pulse\_active\_r and pulse\_active\_f;

end architecture;