Wykorzystanie IP-core AXI UART Lite do komunikacji z komputerem poprzez konwersję UART – USB za pomocą FTDI

AXI UART Lite v2.0 LogiCORE IP

AXI - Advanced eXtensible Interface Bus

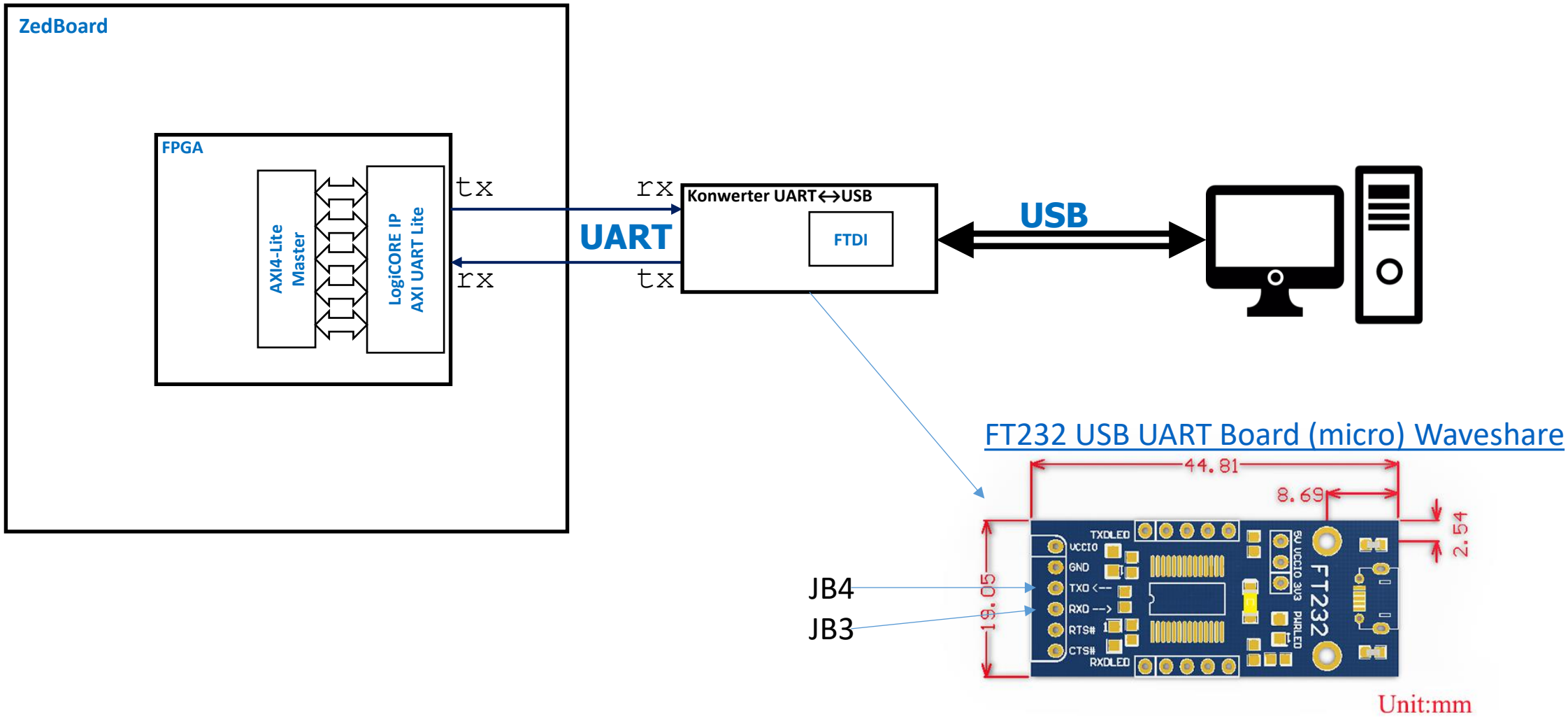
FTDI - Future Technology Devices International

Cel:
Wprowadzenie do projektowania układu cyfrowego za pomocą układu FPGA, języka SystemVerilog, i z wykorzystaniem IP-core'a opartego na magistrali AXI4-Lite.

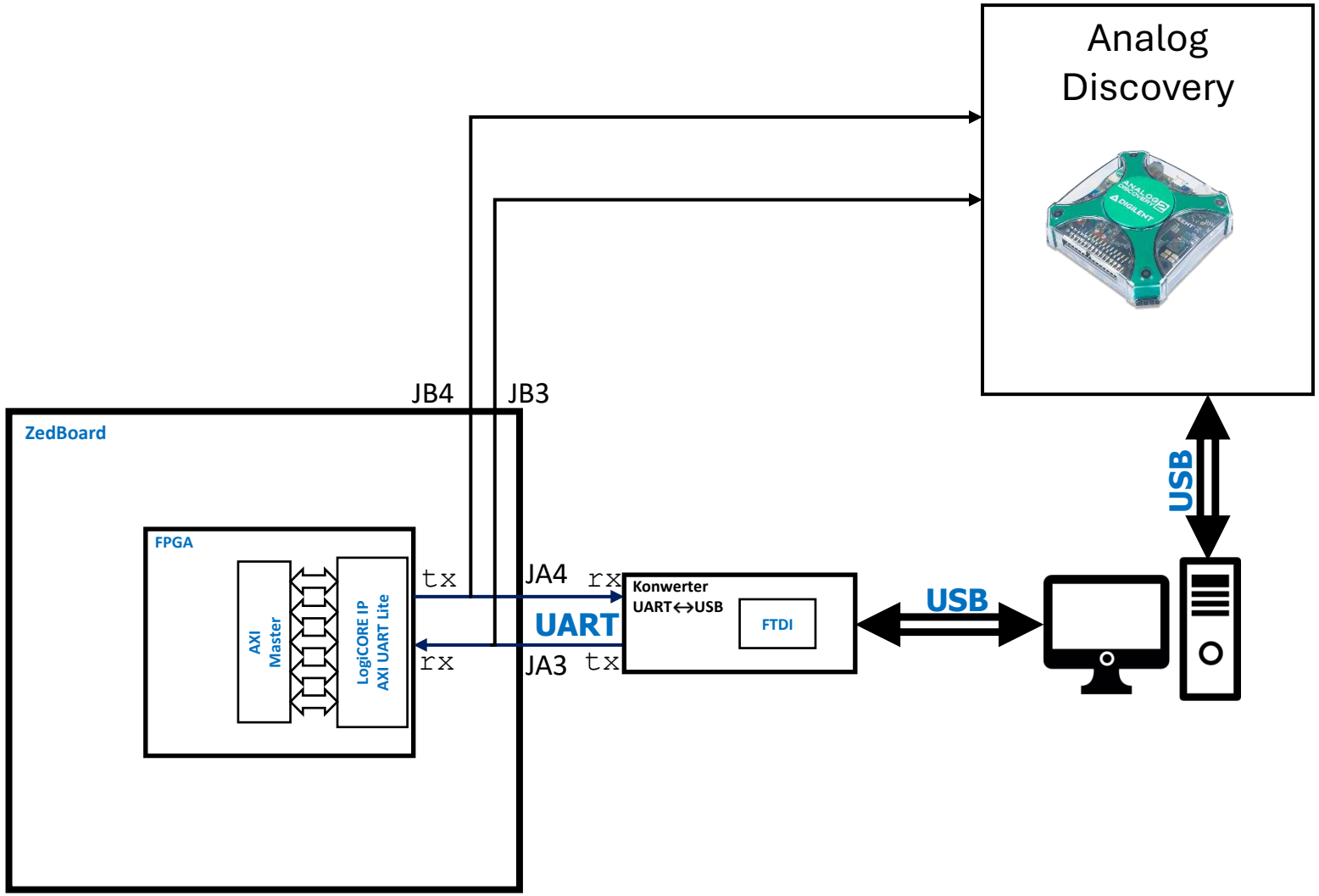
Metoda:
Przygotowanie modułu szczytowego i symulacyjnego. Właczenie IP-core'a.

Laboratorium Języków Opisu Sprzętu AGH WFiIS

Architektura komunikacji: UART w FPGA ↔ USB w komputerze



Obserwacja komunikacji UART

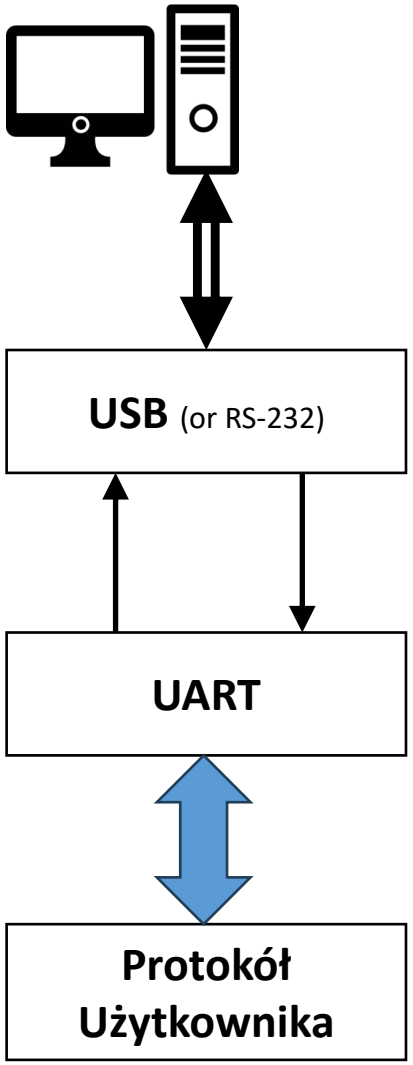


Future Technology Devices International

FTDI Chip to brytyjska firma bez linii technologicznej (*fabless*), która opracowuje innowacyjne rozwiązania krzemowe, które zwiększają możliwości komunikacyjne z najnowszymi światowymi technologiami. Głównym celem firmy jest „łączenie technologii” w celu wspierania inżynierów za pomocą wysoce wyrafinowanych, bogatych w funkcje, solidnych i prostych w użyciu platform produktowych.

1. Zainstaluj pakiet **pyftdi** https://eblot.github.io/pyftdi/features.html
   za pomocą: `pip3 install pyftdi`
2. Podłącz do gniazda USB płytkę FT232 Waveshare
3. Dowiedz się jaki jest adres URL podłączonej płytki za pomocą: `python3 ftdi_urls.py`
4. Poznany adres wprowadź do pozostałych skryptów.
5. Uruchom Vivado w trybie *Hardware Manager* i załaduj do ZedBoard dostarczony strumień bitowy `top.bit`
6. Podłącz na złącze JA (górne piny) płytkę konwertera FT232.
   UWAŻAJ na kolejność pinów zasilania!
7. Podłącz na złącze JB do pinów JB3 i JB4 Analog Discovery w trybie analizatora stanów logicznych. Obserwuj pracę łącza UART.
8. Uruchom skrypt `uart.py` lub `loop_uart.py` aby sprawdzić czy komunikacja między Zedboard a komputerem poprzez UART-USB działa poprawnie.
   UWAŻAJ: konieczne może być naciśnięcie przycisku reset (BTNC na ZedBoard) i/lub wyjęcie włożenie wtyczki USB (UART)
9. Modyfikuj skrypty by przetestować różne zachowania projektu.

Poziomy komunikacji



Zestaw poleceń i danych przesyłanych w ramkach UART

Domyślnie układ jest w trybie **Odbiornika** `rec_trn=1'b1` i w trybie **Komend** `cmdm=1'b1` .
Po otrzymaniu komendy `StartFrameIn` układ przechodzi do trybu **Danych** `cmdm=1'b0`, a po otrzymaniu komendy znów powraca do trybie **Komend** `cmdm=1'b1`.

Struktura komendy

1	d	m	m	m	m	m	m
Kod operacji		**Ilość słów M następująca po komendzie**					

Mniej znaczący bit kodu operacji d oznacza kierunek transmisji `d=1'b0` – odbiór, `d=1'b1` – wysyłanie.

Nazwa	**Kod operacji**	**Binarnie**	**Opis**
`StartFrameIn`	`2'b10`	`8'b10mmmmmm`	Rozpoczęcie ramki transmisji wejściowej, po której nastąpią dane w ilości słów **M** określonej wartością `6'bmmmmmm`. Odebrane słowa umieszczane są w pamięci wewnętrznej od jej początku.
`StopFrameIn`	`2'b10`	`8'b10000000`	Kończy transmisję wejściową. Jest to właściwie ta sama komenda co `StartFrameIn` tylko pole `6'bmmmmmm` jest wyzerowane co oznacza, że już nic dalej nie będzie przesyłana.
`StartFrameOut`	`2'b11`	`8'b11mmmmmm`	Żądanie transmisji wyjściowej w ilości słów **M** określonej wartością `6'bmmmmmm` rozpoczynając od początku pamięci wewnętrznej.

Wielkość **M** jest przechowywana w zmiennej `maxd` i stanowi ograniczenie pracy generatora adresu `maddr`.

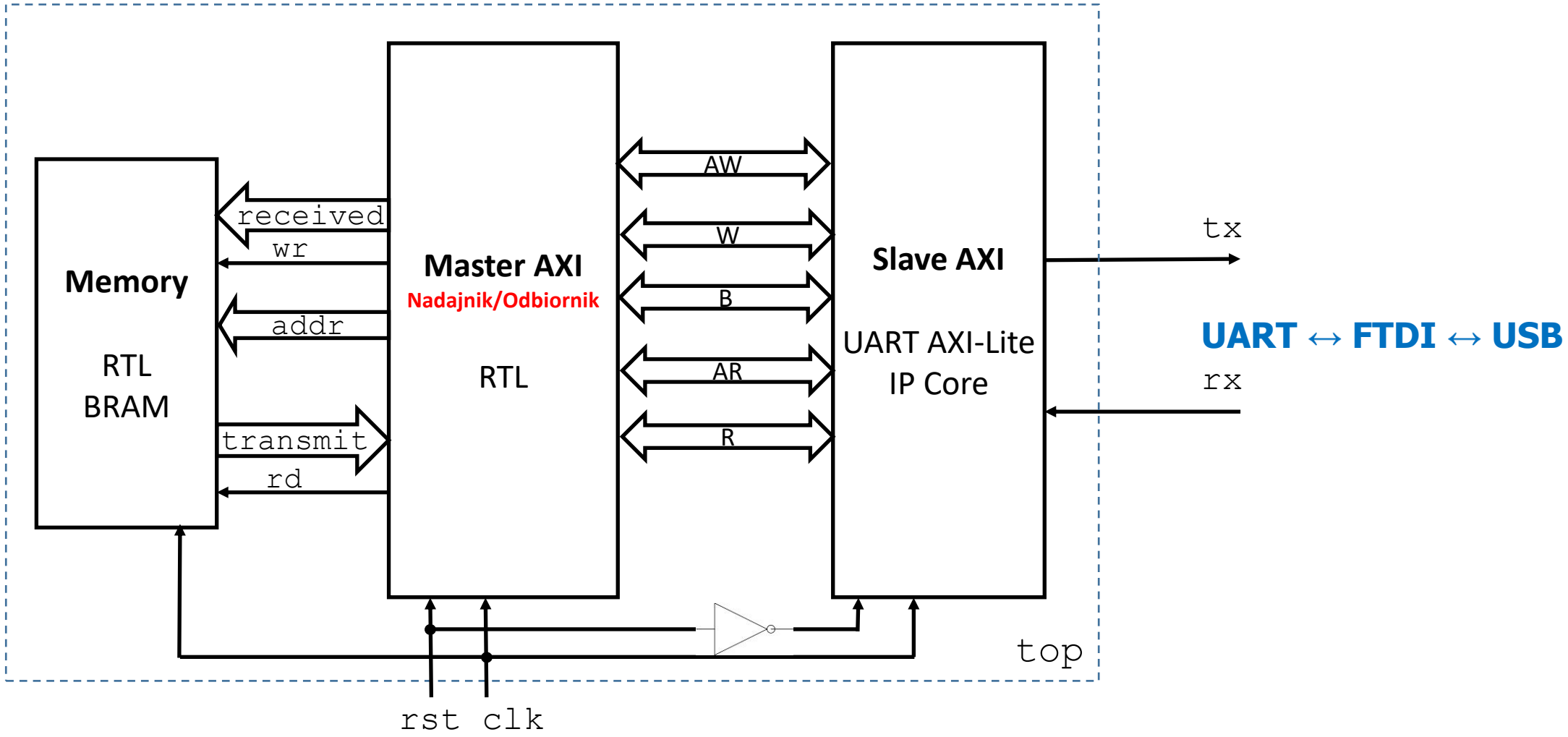
Rejestry wewnętrzne AXI UART Lite

Magistrale i rejestry są 32-bitowe ale tylko najmłodsze bity są używane.
Kolejki FIFO mają głębokość 16 słów. Rozmiar słowa można ustawić w kreatorze generacji IP Core'a.

Nazwa	**Adres** [4'h]	**Opis**
Rx FIFO	0	Tylko do odczytu. Próba odczytu gdy kolejka jest pusta powoduje błąd
Tx FIFO	4	Tylko do zapisu. Próba zapisu gdy kolejka jest pełna powoduje błąd
STAT_REG	8	Tylko do odczytu. Znaczenie bitów rozpoczynając od najmłodszego: [0] - "ważne dane w kolejce Rx FIFO", [1] - "pełna kolejka Rx FIFO", [2] - "pusta kolejka Tx FIFO", [3] - "pełna kolejka Tx FIFO", [4] - "przerwania umożliwione", [5] - "przepełnienie", [6] - "błąd ramki", [7] - "błąd parzystości"
CTRL_REG	C	Tylko do zapisu. Znaczenie stanu wysokiego w trzech bitach: [0] – "wyczyść kolejkę Tx FIFO", [1] – "wyczyść kolejkę Rx FIFO", [4] – "umożliwienie generacji przerwań"

Pełny opis: Product Guide PG142, Xilinx

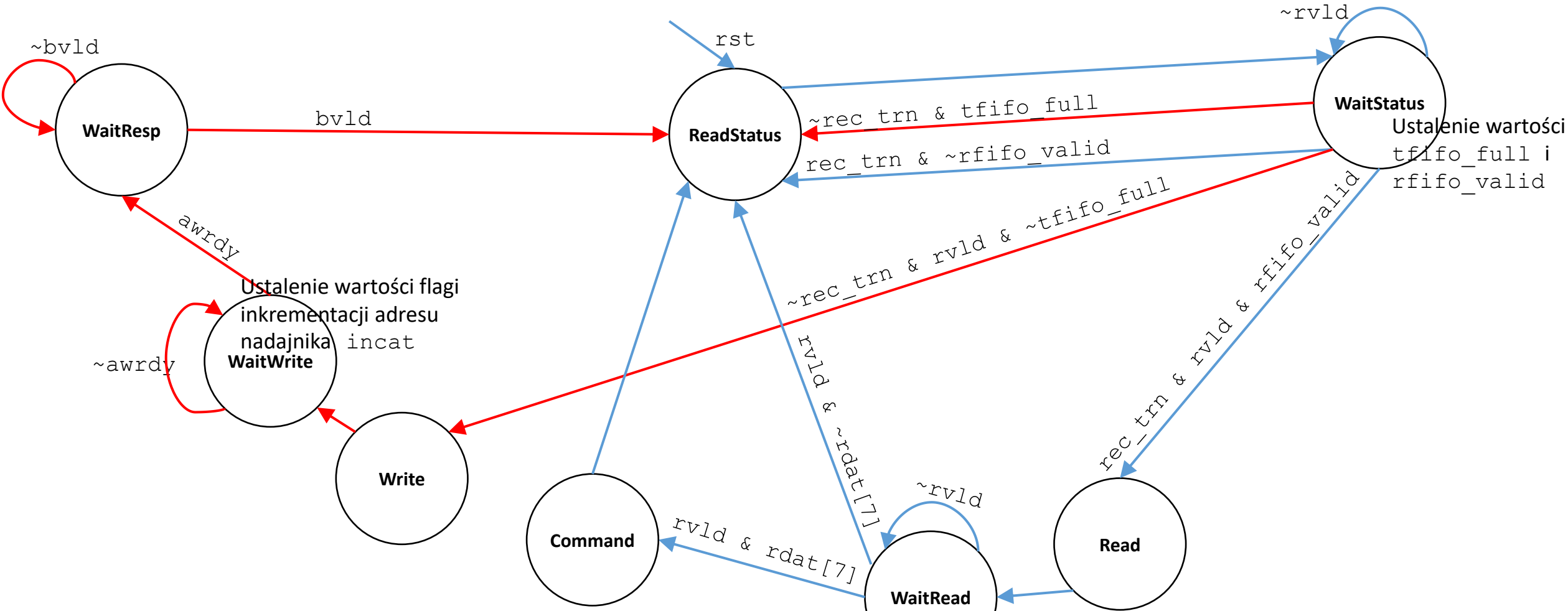
Schemat blokowy projektu



Koncepcja modułu nadrzędnego AXI4-Lite

```
module master_axi #(parameter nb = 4, deep = 16) (input clk, rst,
    output logic [3:0] wadr, output logic awvld, input awrdy,
    output [31:0] wdat, output logic wvld, input wrdy,
    input [1:0] brsp, input bvld, output logic brdy,
    output logic [3:0] radr, output logic arvld, input arrdy,
    input [1:0] rdat, input rvld, output logic rrdy,
    input [7:0] data_tr, output logic [7:0] data_rec,
    output logic [nb-1:0] addr, output logic wr, rd);
//Rx FIFO valid flag
//Tx FIFO full flag
//flip-flop to distinguish transmit and receive
//FSM: state register and next state logic
//command decoder
//transaction counter (memory address generator) and flags
// Receiver control ------------------------------------------
//channel AR
//channel R
//memory write
// Transmitter control ---------------------------------------------
//channel AW
//channel W
//channel B
//memory read
endmodule
```

Graf automatu sterującego nadajnikiem/odbiornikiem



Tryb **Komendy** `cmdm = 1'b1`
Zmiana w stanie **Command** gdy rozpoznano komendę `StopFrameIn`
Tryb **Dane** `cmdm = 1'b0`

Tryb **Odbiornik** `rec_trn = 1'b1`
Zmiana w stanie **Command** gdy rozpoznano komendę `StartFrameOut`
Tryb **Nadajnik** `rec_trn = 1'b0`

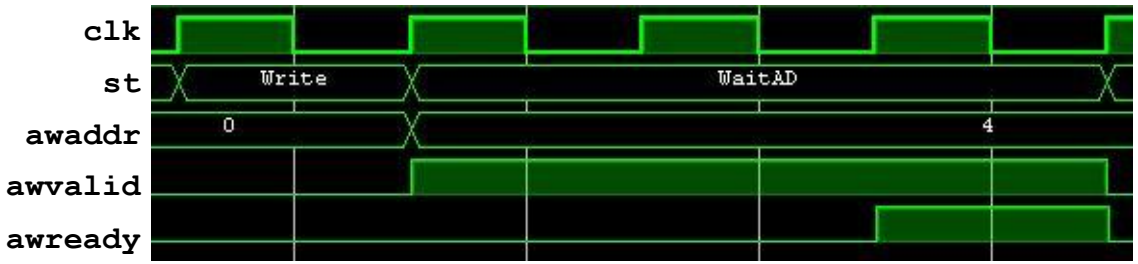
Stany automatu nadajnika /odbiornika

Stan	Akcja
`readstatus`	Początkowy stan domyślny. Ustawienie adresu odczytu AR rejestru STATUS, ustawienie sygnałów ważności adresu AR
`waitstatus`	Oczekiwanie na sygnał ważności danych w kanale R; po otrzymaniu przejście do `read` jeśli bit **zerowy** w rejestrze STATUS jest aktywny (`rfifo_valid`), a do `write` gdy bit **trzeci** w rejestrze STATUS jest nieaktywny (`tfifo_full`), a w przeciwnym razie powrót do `readstatus`
`read`	Bezwarunkowe przejście do `waitread`
`waitread`	Odczyt z kolejki Rx FIFO z wykorzystanie *handshake* w kanale AR i R
`command`	Dekodowanie komendy
`write`	Bezwarunkowe przejście do `waitwrite`
`waitwrite`	Sprawdzenie licznika odczytów; po osiągnięciu zadanej wartości przejście do `waitresp` w przeciwnym razie pozostanie w `waitwrite`
`waitresp`	Oczekiwanie na sygnał zezwolenia (valid) w kanale odpowiedzi B; po otrzymaniu przejście do `ReadStatus`

Ważne znaczniki

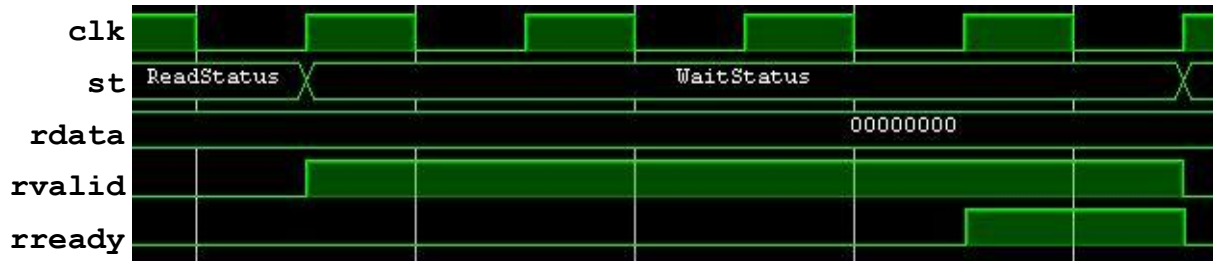
`tfifo_full`	***Znacznik zapełnienia kolejki wyjściowej*** Tx FIFO jest zależny od trzeciego bitu w STAT_REG. Powtarza jego wartość w stanie WaitStatus gdy przychodzi potwierdzenie kanału odczytu danych R (`rvld`).
`rfifo_valid`	***Znacznik obecności danych wejściowych*** w kolejce Rx FIFO jest zależny od zerowego bitu w STAT_REG. Powtarza jego wartość w stanie WaitStatus gdy przychodzi potwierdzenie kanału odczytu danych R (`rvld`).
`incar`	***Warunek inkrementacji adresu w czasie odbioru***. Aktywny w stanie WaitRead gdy adres jest mniejszy od wartości maksymalnej określonej komendą StartFrameIn i jest koniunkcją zaprzeczenia znacznika komendy, znacznika kierunku transmisji i sygnału potwierdzenia kanału odczytu danych R (`rvld`).
`incat`	***Warunek ikrementacji adresu w czasie nadawania***. Aktywny w stanie WaitWrite gdy adres jest mniejszy od wartości maksymalnej określonej komendą StartFrameOut i jest koniunkcją znacznika komendy, zaprzeczenia znacznika kierunku transmisji i sygnału gotowości kanału zapisu danych W (`wrdy`).
`cmdm`	***Znacznik komendy***: "1" - komenda, "0" – dane Znacznik ten domyślnie jest w stanie komendy i zmienia stan po odebraniu komendy StartFrameIn na czas wskazanej ilości odebranych bajtów. Przy pracy w kierunku nadawania pozostaje wysoki.
`rec_trn`	***Znacznik kierunku transmisji***: "1" - odbiór, "0" – nadawanie. Znacznik ten domyślnie jest w stanie odbioru i tylko komenda StartFrameOut przełącza go w stan nadawania. Znacznik ten pozostaje w stanie niskim tylko przez czas potrzebny na wysłanie paczki danych na magistrali AXI4-Lite. Długość tej paczki jest co najwyżej równa głębokości kolejki wyjściowej Rx FIFO.

Generacja sygnałów Master AXI w języku Verilog




```
always @(posedge clk, negedge rst)
        if(~rst)
                awaddr <= 32‘h0;
        else if (st == Write)
                awaddr <= 32'h89ac6471;
always @(posedge clk, negedge rst)
        if(~rst)
                awvalid <= 1'b0;
        else if (st == Write)
                awvalid <= 1'b1;
        else if (awready)
                awvalid <= 1'b0;
```

handshake dla kanałów: AW (W, AR)




```
always @(posedge clk, negedge rst)
        if(~rst)
                rready <= 1'b0;
        else if (st == Read)
                rready <= 1'b1;
        else if (rvalid)
                rready <= 1'b0;
assign received = (rvalid)?rdata:32’h0;
```

handshake dla kanałów: R (B)

Dokładniejsze informacje w prezentacji z wykładu 10 slajdy od 10 do 15

Symulacja

```
module tb();
localparam d = 40, hp = 5, fclk = 100_000_000, br = 230400, size = 8;
localparam ratio = fclk / br - 1;
//liczba komend/danych w pamięci nadajnika behawioralnego
localparam nr_trn = 9;
//rozmiar pamięci danych odbiornika behawioralnego
localparam nr_rec = 11;
logic clk, rst;
//start nadajnika i odbiornika
logic strt, strr;
//zakończenie pracy nadajnika i odbiornika
wire fint, finir;
//instacja projektu testowanego
//instacja nadajnika behawioralnego
//instacja odbiornika behawioralnego
```

Zawartość pliku `tr_init.mem` koniecznego dla nadajnika behawioralnego

86 – StartFrameIn dla **nr_trn** = 9 – 3 komendy = 6 bajtów
32
31
34
33 } dane, 6 bajtów
37
36
80 – StopFrameIn
cb – StartFrameOut dla **nr_rec** = 11 bajtów

Pominięto generację zegara i resetu systemowego

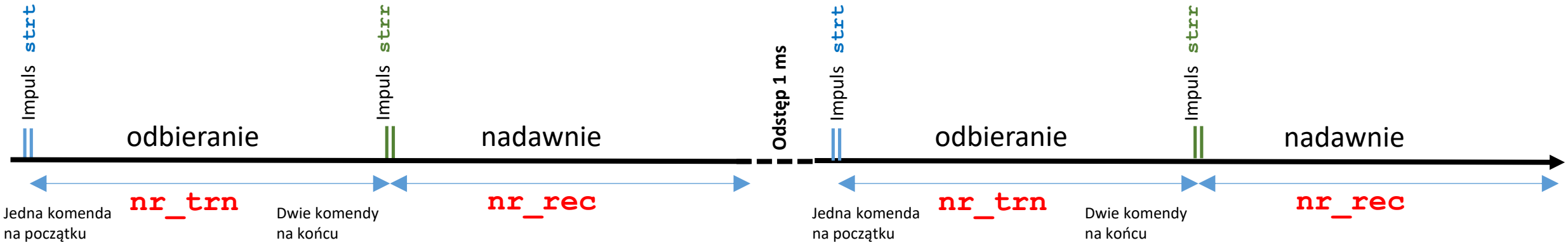
Moduły nadajnika i odbiornika dla wykorzystania w symulacji są dostępne na UPeL w osobnym archiwum .tgz

```
initial begin
    repeat(2*nr_rec+1) @(negedge finr);
    #200000 $finish();
end
endmodule
```

Zakończenie symulacji

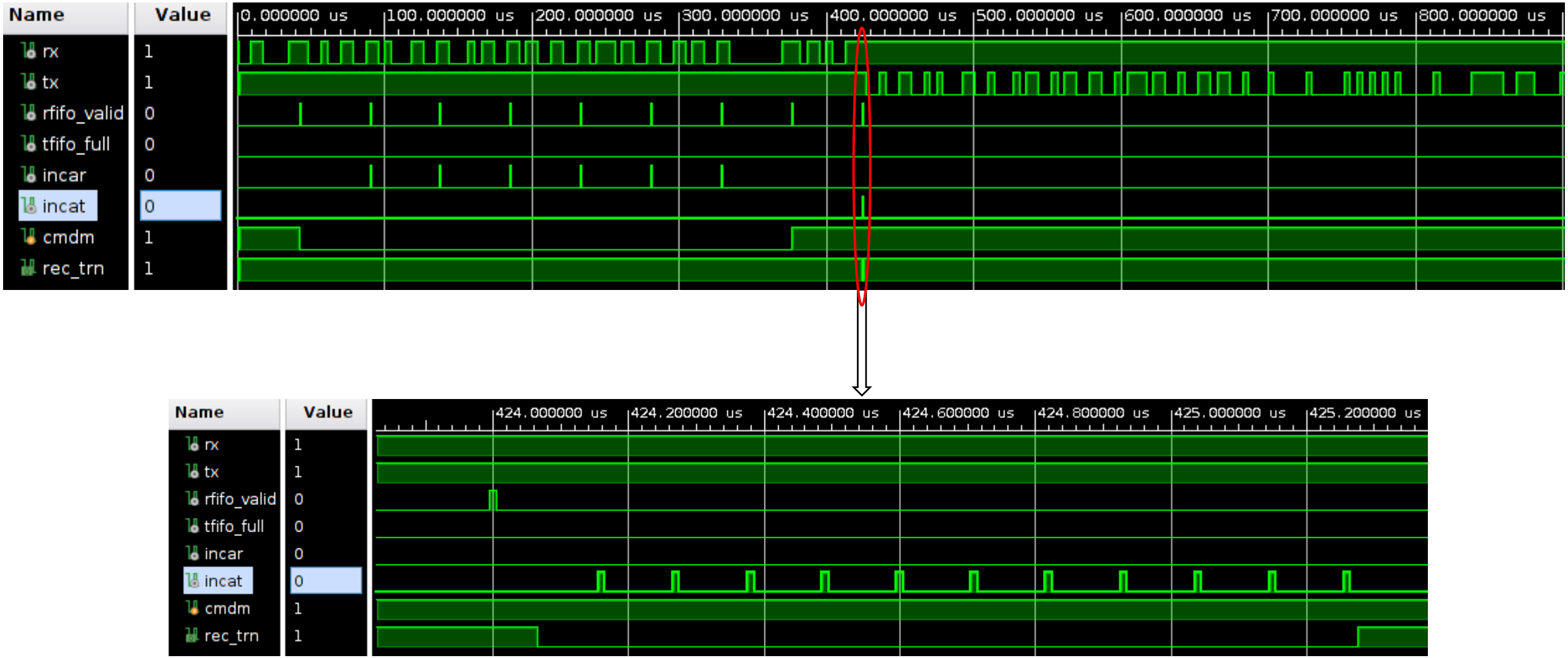
Symulacja

Przykładowy przebieg symulacji




```
initial begin
    strt = 1'b0;
    strr = 1'b0;
    @(negedge rst);
    repeat(ratio/8) @(posedge clk);
    strt = 1'b1;
    $display("Start sending at: %t ns", $time);
    @(negedge clk);
    strt = 1'b0;
    repeat(nr_trn) @(negedge fint);
    strr = 1'b1;
    repeat(2) @(negedge clk);
    strr = 1'b0;
    repeat(nr_rec) @(negedge finr);
    #1000000 strt = 1'b1;
    $display("Start sending second time at: %t ns", $time);
    @(negedge clk);
    strt = 1'b0;
    repeat(nr_trn) @(negedge fint);
    strr = 1'b1;
    repeat(2) @(negedge clk);
    strr = 1'b0;
    repeat(nr_rec+1) @(negedge finr);
end
```

Symulacja


Name
Value
rx
tx
rfifo_valid
tfifo_full
incar
incat
cmdm
rec_trn
1
1
0
0
0
0
1
1
0.000000 us
100.000000 us
200.000000 us
300.000000 us
400.000000 us
500.000000 us
600.000000 us
700.000000 us
800.000000 us
424.000000 us
424.200000 us
424.400000 us
424.600000 us
424.800000 us
425.000000 us
425.200000 us

Od strony komputera

Proponuję rozwiązanie za pomocą pakietu `pyftdi`:

```
pip3 install pyftdi
```

Połączenie do UART jest opisane tutaj

Ten adres urządzenia jest zwracany przez skrypt: `ftdi_urls.py`

```
# Enable pyserial extensions
import pyftdi.serialext

# Open a serial port on the FTDI device
brate = 230400
url = 'ftdi://ftdi:232:AQ00RVZ8/1'
port = pyftdi.serialext.serial_for_url(url, baudrate=brate, bytesize=8, stopbits=1, parity='N', xonxoff=False, rtscts=False)

# Send bytes
print("Transmition at", brate)
b = bytes([0x33, 0x35, 0x86, 0x32, 0x4c, 0x31, 0x36, 0x35, 0x37, 0x80, 0xc6])
print("-", b)
port.write(b)

# Receive bytes
nb = 6
print("Receiving at", brate)
print(nb, "bytes")
data = port.read(nb)
print('-', data)
port.close()
```

6 bajtów danych

StartFrameIn M=6

StopFrameIn M=0

StartFrameOut M=6

Przed pierwsza komendą
nie są interpretowane

Musi być ta sama wartość !!!

Reset synchroniczny LogiCORE IP AXI UART Lite

Dla rozpoczęcia poprawnej pracy skryptu może być konieczne naciśnięcie przycisku reset, gdyż IP-core wymaga kasowania synchronicznego i działa tylko w obecności zegara.

Może też być potrzebne wykonanie ponownego podłączenia podsystemu USB w systemie operacyjnym Linux poprzez wyjęcie i włożenie etyczki USB.

Od strony komputera

Polecenie:

Przeanalizuj skrypt i wykonaj kolejne zmiany w danych komendach aby przetestować projekt sprzętowy. Pamiętaj o zasadach składniowych komend.

Literatura:

1. AXI UART Lite v2.0 LogiCORE IP, Product Guide, Vivado Design Suite, PG142, April 5, 2017
2. FT232R USB UART IC Datasheet Version 2.07 Clearance No.: FTDI# 38, Document No.: FT_000053, FTDI Ltd., 2010 Waveshare
3. PyFtdi Documentation, https://eblot.github.io/pyftdi/