BAB 4

UART

Pada bab ini, praktikan akan mempelajari konsep dasar dan fungsi dari UART (Universal Asynchronous Receiver-Transmitter), yang merupakan komponen penting dalam komunikasi serial. Praktikan akan mempelajari cara kerja UART dan bagaimana mengimplementasikannya pada FPGA untuk mendeskripsikan proses transmisi dan penerimaan data secara asinkron. Selain itu, praktikan akan memahami peran FTDi chip dalam konversi sinyal USB ke sinyal serial. Asisten praktikum atau praktikan diharapkan membaca tujuan dan persyaratan bab ini agar praktikum dapat berjalan sesuai dengan prosedur.

Tujuan

Tujuan	Penjelasan
Memahami Fungsi dan Cara Kerja UART	Praktikan diharapkan dapat memahami fungsi UART sebagai modul komunikasi serial yang bertugas mengonversi data paralel menjadi data serial dan sebaliknya.
Mengimplementasikan UART pada FPGA	Praktikan diharapkan mampu mengimplementasikan modul UART pada FPGA menggunakan bahasa pemrograman VHDL atau Verilog.
Memahami Fungsi dan Cara Kerja FTDi Chip	Praktikan akan memahami cara kerja FTDI chip, serta bagaimana chip ini digunakan dalam proyek FPGA untuk mempermudah koneksi serial dengan perangkat eksternal seperti komputer.

Persyaratan

Disarankan praktikan menggunakan hardware dan software sesuai pada dokumentasi ini. Apabila terdapat versi hardware atau software yang cukup lama dari versi yang direkomendasikan maka sebaiknya bertanya kepada Asisten Mengajar Shift.

HARDWARE YANG DIBUTUHKAN PRAKTIKUM

PC / Laptop

FPGA Board Nexys A7 dan Nexys 4

Kabel Power USB

SOFTWARE YANG DIBUTUHKAN PRAKTIKUM

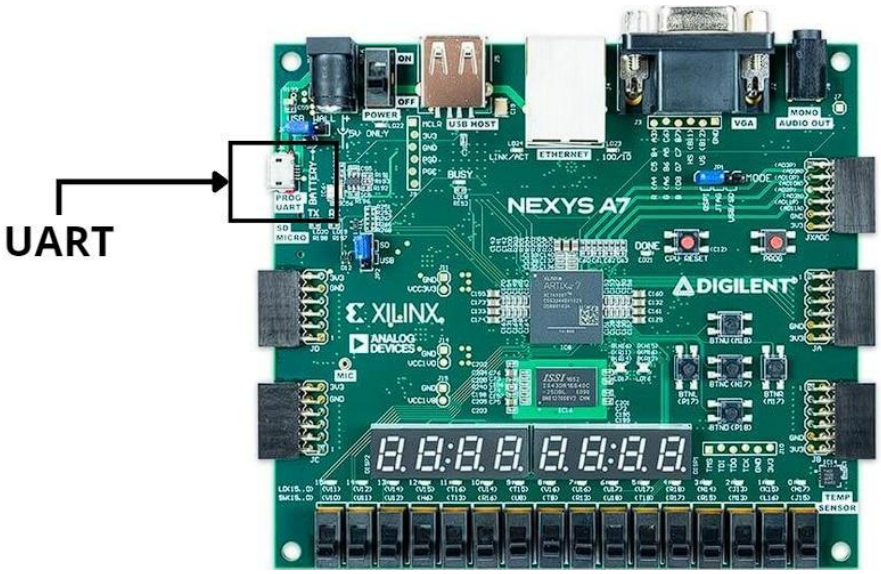
Vivado Design Suite

⚠ Diusahakan untuk memakai **Versi** dan **Aplikasi** yang sama agar tidak terjadinya kesalahan yang tidak diinginkan!

Teori

4.1. Pengertian

UART adalah kepanjangan dari *Universal Asynchronous Receiver/Transmitter* adalah perangkat keras yang menerjemahkan antara bit- bit paralel data dan serial. UART umumnya digunakan sebagai protokol komunikasi serial asinkron dalam pengiriman data antara perangkat satu dengan yang lainnya. Sebagai contoh, komunikasi antara sesama mikrokontroler atau mikrokontroler dengan PC.



Gambar 4. 1 Port UART pada Nexys A7

4.1.1. Cara Kerja UART

Dalam pengiriman data menggunakan UART, *baud rate* antara pengirim dan penerima harus sama karena paket data dikirim tiap bit mengandalkan *clock* tersebut. Inilah salah satu keuntungan menggunakan model *asynchronous* dalam pengiriman data karena dengan hanya satu kabel transmisi maka data dapat dikirimkan. Berbeda dengan model *synchronous* yang terdapat pada protocol SPI dan I2C karena protokol tersebut membutuhkan minimal dua kabel dalam transmisi data, yaitu transmisi *clock* dan data. Namun, kelemahan model *asynchronous* adalah dalam hal kecepatannya dan jarak transmisi. Semakin cepat dan jauhnya jarak transmisi, membuat paket-paket bit data menjadi terdistorsi sehingga data yang dikirim atau diterima bisa mengalami error.

Transmitter/Pengiriman Data:

Data paralel dari perangkat sumber dikonversi menjadi data serial oleh UART. Bit data dikirim secara berurutan melalui jalur komunikasi serial.

- Data paralel dari perangkat sumber dikonversi menjadi data serial oleh UART dan dibagi menjadi bit-bit individu.
- Bit individu dari data yang dikirim dimulai dari yang terkecil pertama (LSB).
- Bit start (0) dikirim untuk menandai awal transmisi.
- Setelah bit *start*, dikirimkan bit-bit data.
- Bit *parity* (opsional) dikirim untuk deteksi kesalahan.
- Setelah semua bit data dikirim, dikirimkan bit *stop*.
- Bit *stop* (1) dikirim untuk menandai akhir transmisi.

Setiap bit yang ditransmisikan serupa dengan jumlah bit lainnya, dan penerima mendeteksi jalur disekitar pertengahan periode setiap bit untuk menentukan apakah bit adalah 1 atau 0. Misalnya, jika dibutuhkan dua detik untuk mengirim setiap bit, penerima akan memeriksa sinyal untuk menentukan apakah itu adalah 1 atau 0 setelah satu detik telah berlalu.

Receiver/Penenerimaan Data:

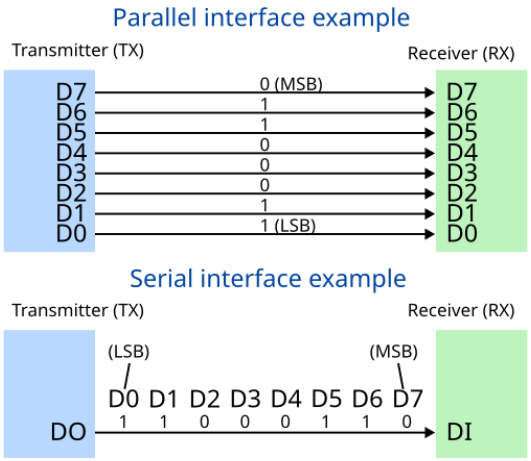
UART menerima data serial dan mengonversinya kembali menjadi data paralel untuk perangkat tujuan.

- Bit *start* dideteksi untuk menyinkronkan penerima dengan kecepatan data.
- Ketika terdeteksi bit *start*, perangkat penerima mulai mengukur waktu.
- Pada setiap batas waktu, perangkat penerima membaca nilai bit data.
- Bit data diterima dan dirakit sesuai urutan.
- Bit *parity* diperiksa untuk deteksi kesalahan.
- Bit *stop* dideteksi untuk menandai akhir transmisi.

4.2 Komunikasi Serial

UART pada dasarnya merupakan protokol komunikasi serial. Artinya, data dikirimkan bit demi bit melalui satu saluran komunikasi. Komunikasi serial adalah proses pengiriman bit data satu per satu, secara berurutan, melalui sebuah saluran komunikasi. Hal ini berbeda dengan komunikasi paralel, dimana beberapa bit dikirim secara keseluruhan, *link* dengan beberapa saluran paralel. UART menggunakan komunikasi serial karena hanya membutuhkan dua kabel utama (TX dan RX) untuk komunikasi dua arah, sehingga lebih hemat biaya dan mudah diimplementasikan.

Metode komunikasi serial sering disebut dengan TTL Serial (*transistor-transistor logic*). Komunikasi serial pada tingkat TTL akan selalu tetap antara batas 0V dan VCC, yang sering digunakan yaitu 5V atau 3.3V. Sebuah logika tinggi (1) diwakili oleh VCC sedangkan logika rendah (0) adalah 0V.



Gambar 4. 2 Komunikasi Serial dan Paralel

Pada saat berkomunikasi, semakin tinggi frekuensi pengiriman data, maka semakin tinggi juga gangguan elektromagnetik. Setiap kabel dapat diperlakukan sebagai antena untuk menangkap *noise* yang ada di sekitarnya yang mengganggu

data yang sedang ditransmisikan. Dalam komunikasi paralel, karena banyaknya kabel yang digunakan, masalah gangguan elektromagnetik menjadi lebih serius. Komunikasi serial dibutuhkan jumlah kabel yang lebih sedikit, bisa hanya menggunakan tiga kabel, yaitu saluran *Transmit Data* (TX), saluran *Receive Data* (RX) dan saluran *Ground* (GND).

4.2.1. Jenis-jenis Komunikasi Serial

a. Synchronous

Sebuah pengiriman data serial yang disertai dengan clock sebagai pengatur waktu untuk mengindikasi bahwa ada bit siap untuk dibaca. Contoh : I2C, USRT, SPI, dll.

b. Asynchronous

Sebuah pengiriman data dimana clock tidak dikirim bersamaan dengan data sehingga masing-masing perangkat keras yang berkomunikasi harus membuat clock-nya sendiri yang sama. Contoh : UART, RS- 232, USB, dll.

c. Full duplex

Jenis komunikasi serial yang menyatakan hubungan antara dua perangkat keras. Misalnya ada sebuah perangkat A dan B, jika A sedang melakukan pengiriman data, pada saat yang sama A dapat menerima data dari B, begitupun

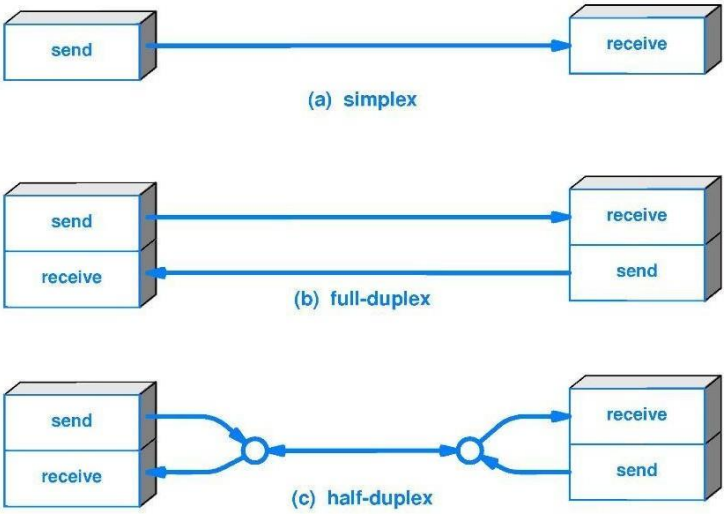
sebaliknya. Kondisi ini dinamakan full duplex atau komunikasi dua arah, contohnya adalah telepon.

d. Half duplex

Kondisi ketika proses pengiriman dan penerimaan data tidak dapat dilakukan secara bersamaan, namun secara bergantian. Contohnya adalah walkie talkie.

e. Simplex

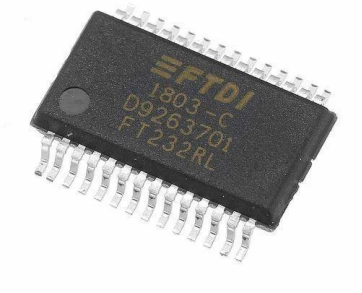
Jenis komunikasi dimana pengiriman hanya terjadi satu arah saja kepada penerima. Contohnya seperti broadcast sebuah pesan atau seperti TV.



Gambar 4. 3 Simplex, Full-Duplex, Half-Duplex

4.3. FTDI Chip

FTDI (*Future Technology Devices International*) adalah perusahaan yang memproduksi chip konverter *USB-to-serial*. Komunikasi UART tidak bisa diterapkan melalui jalur USB secara langsung tanpa menggunakan sebuah modul FTDI. Modul ini mengelola transaksi data dengan menerapkan protokol USB dari data yang ditransfer melalui serial asinkron. Ini memungkinkan perangkat modern dengan port USB untuk berkomunikasi dengan perangkat yang menggunakan antarmuka serial tradisional seperti RS-232, RS-485, atau TTL. Tanggung jawab perangkat ini adalah untuk memberitahu PC bahwa perangkat yang digunakan dengan menambahkan beberapa informasi identifikasi, sehingga PC dapat memuat *driver* yang tepat.



Gambar 4. 4 FTDI Chip

4.3.1. Cara Kerja FTDI Chip

- Ketika data serial diterima, FTDI Chip mengubahnya menjadi format USB yang dapat dipahami oleh komputer.
- Sebaliknya, ketika komputer mengirimkan data USB, FTDI Chip mengubahnya menjadi format serial yang dapat dipahami oleh perangkat yang terhubung.

FTDI Chip juga melakukan konversi level logika antara 3.3V atau 5V (umumnya digunakan oleh perangkat serial) dan 5V (umumnya digunakan oleh port USB).

4.4. ASCII

ASCII (American Standard Code for Information Interchange) merupakan standar pengkodean karakter yang dikembangkan pada tahun 1960-an untuk memudahkan pertukaran informasi antar perangkat elektronik. ASCII menggunakan representasi bilangan biner untuk setiap huruf, angka, simbol, maupun karakter kontrol. Setiap karakter dalam ASCII direpresentasikan dengan 7 bit, sehingga mampu menampung 128 karakter (kode 0–127). Pada implementasi modern, ASCII biasanya disimpan dalam 8 bit (1 byte) dengan tambahan 1 bit yang tidak digunakan atau digunakan untuk memperluas tabel ke Extended ASCII (0–255).

Decimal - Binary - Octal - Hex – ASCII
Conversion Chart

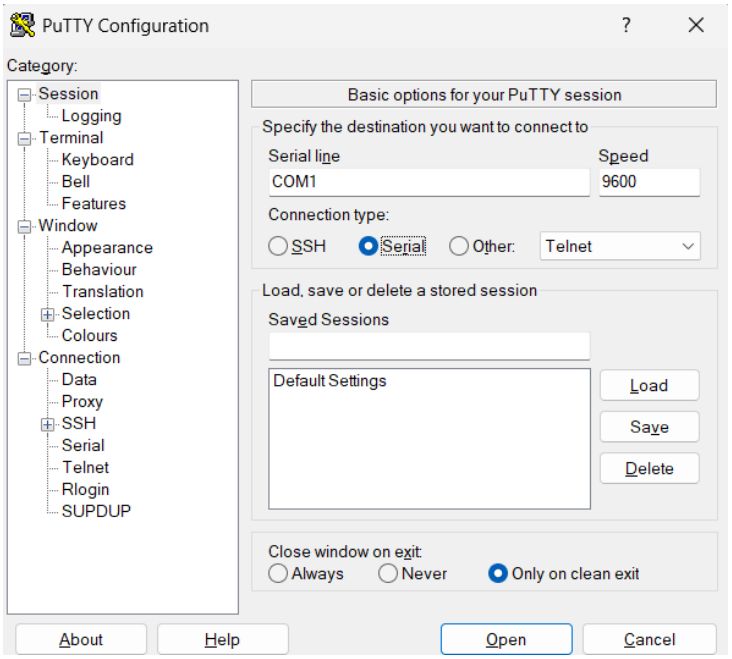
Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

4.5 Tabel Konversi ASCII

4.5. PUTTY

PuTTY adalah sebuah program open source yang dapat digunakan untuk melakukan protocol jaringan SSH, Telnet, Serial dan Rlogin. Protocol ini dapat digunakan untuk menjalankan sesi remote pada sebuah computer melalui sebuah jaringan baik itu LAN maupun internet. PuTTY pada awalnya dibuat untuk Microsoft Windows, tetapi telah mendukung berbagai system operasi lainnya. Aplikasi PuTTY digunakan ketika ingin mentransfer sebuah data dari computer ke sebuah perangkat lain.

Pada Nexys A7, PuTTY dapat digunakan untuk menghubungkan komputer dengan FPGA melalui port UART. Dengan PuTTY, kita bisa mengirimkan dan menerima data secara langsung antara komputer dan board Nexys A7, misalnya untuk debugging atau mengontrol perangkat melalui antarmuka teks.



Gambar 4.6 Tampilan Awal PuTTY

4.6. Sensor

Sensor merupakan komponen elektronik yang berfungsi untuk mendeteksi fenomena fisik dari lingkungan lalu mengubahnya menjadi sinyal listrik yang dapat diolah oleh sistem digital, seperti FPGA. Dalam implementasi pada board Nexys A7, sensor sering digunakan untuk membaca parameter lingkungan seperti suhu, cahaya, kelembapan, atau percepatan. Jenis sensor dapat dibagi menjadi beberapa kelompok berdasarkan besaran fisik yang diukur, di antaranya:

- **Sensor analog**, yang menghasilkan sinyal tegangan/arus kontinu, contohnya sensor cahaya (LDR) atau sensor suhu analog (LM35).

  Sensor digital, yang mengeluarkan data dalam bentuk digital melalui protokol komunikasi seperti I²C, SPI, atau UART, contohnya ADT7420 (sensor suhu digital), HDC1080 (sensor suhu dan kelembapan), dan akselerometer berbasis SPI.

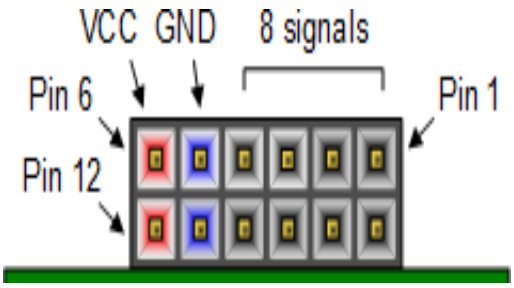
FPGA tidak dapat langsung memproses sinyal analog, sehingga jika menggunakan sensor analog, diperlukan **ADC (Analog-to-Digital Converter)** untuk mengubah sinyal analog menjadi data digital. Pada Nexys A7, ADC sudah terintegrasi melalui **XADC** sehingga input analog dapat dibaca melalui pin khusus.

4.7. Pin Pmod

Nexys A7 menyediakan **Pmod (Peripheral Module)** sebagai antarmuka standar untuk menghubungkan modul eksternal, termasuk sensor. PMOD adalah konektor dengan format 2×6 pin (12 pin) yang memudahkan ekspansi board FPGA dengan berbagai perangkat tambahan.

Struktur dasar pin pada konektor Pmod adalah:

- **4 pin I/O** (GPIO) yang dapat diprogram melalui FPGA.
- **2 pin GND** sebagai ground referensi.
- **2 pin VCC** (biasanya 3,3 V) sebagai catu daya untuk modul eksternal.
- Sisanya digunakan sebagai pin tambahan untuk I/O atau sinyal kontrol.



Gambar 4.7 Pin Pmod Nexys A7

Nexys A7 memiliki beberapa port Pmod (JA, JB, JC, JD, JE, JXADC). Masing-masing port bisa digunakan untuk menghubungkan sensor dengan protokol yang sesuai. Misalnya:

- **JXADC** → khusus untuk input analog menggunakan pin XADC
- **JA–JD** → digunakan untuk komunikasi digital (SPI, I²C, UART, atau GPIO biasa).

Table 10.1. Nexys A7 Pmod pin assignments.

Pmod JA	Pmod JB	Pmod JC	Pmod JD	Pmod XADC
JA1: C17	JB1: D14	JC1: K1	JD1: H4	JXADC1: A13 (AD3P)
JA2: D18	JB2: F16	JC2: F6	JD2: H1	JXADC2: A15 (AD10P)
JA3: E18	JB3: G16	JC3: J2	JD3: G1	JXADC3: B16 (AD2P)
JA4: G17	JB4: H14	JC4: G6	JD4: G3	JXADC4: B18 (AD11P)
JA7: D17	JB7: E16	JC7: E7	JD7: H2	JXADC7: A14 (AD3N)
JA8: E17	JB8: F13	JC8: J3	JD8: G4	JXADC8: A16 (AD10N)
JA9: F18	JB9: G13	JC9: J4	JD9: G2	JXADC9: B17 (AD2N)
JA10: G18	JB10: H16	JC10: E6	JD10: F3	JXADC10: A18 (AD11N)

Gambar 4.8 Pinout Pmod JA, JB, JC, JD dan XADC

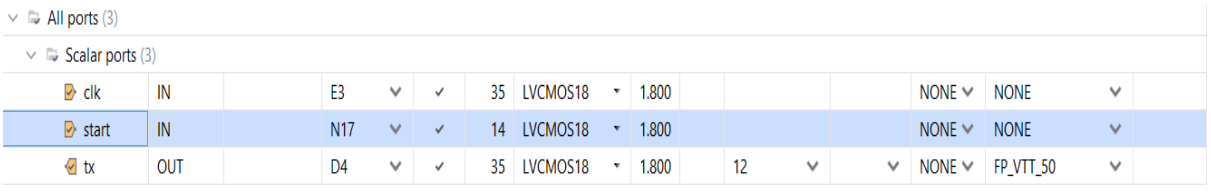
Praktek

1. Menampilkan Hello, World! pada Putty menggunakan Nexys A7

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity UART_HelloWorld is
    Port (
        clk   : in  std_logic;  -- System clock (100 MHz)
        tx    : out std_logic;  -- UART Transmit
        start : in  std_logic   -- Start signal
    );
end UART_HelloWorld;

architecture Behavioral of UART_HelloWorld is

    constant clk_freq  : integer := 100_000_000; -- 100 MHz
    constant baud_rate : integer := 9600;
    constant baud_div  : integer := clk_freq / baud_rate;

    type state_type is (IDLE, SEND);
type string_array is array (0 to 12) of std_logic_vector(7 downto 0);
signal message : string_array := (
    X"48", -- H
    X"45", -- E
    X"4C", -- L
    X"4C", -- L
    X"4F", -- O
    X"20", -- (spasi)
    X"57", -- W
    X"4F", -- O
    X"52", -- R
    X"4C", -- L
    X"44", -- D
    X"0D", -- \r (Carriage return)
    X"0A"  -- \n (Line feed)
);

    signal char_index : integer range 0 to 12 := 0;
```

```
begin

    process (clk)
    begin
        if rising_edge(clk) then
            case state is
                when IDLE =>
                    tx <= '1';  -- idle line
                    if start = '1' then
                        char_index <= 0;
                        state <= SEND;
                    end if;

                when SEND =>
                    if tx_busy = '0' then
                        tx_data   <= message(char_index);
                        -- UART frame: start(0) + data(8bit LSB first) + stop(1)
                        shift_reg <= '1' & tx_data & '0';
                        bit_cnt   <= 0;
                        tx_busy   <= '1';
                    end if;

                    if tx_busy = '1' then
                        if baud_counter = baud_div then
                            baud_counter <= 0;
                            tx <= shift_reg(0); -- kirim LSB
                            shift_reg <= '1' & shift_reg(9 downto 1);
                            bit_cnt <= bit_cnt + 1;
                            if bit_cnt = 9 then
                                tx_busy <= '0';
                                if char_index = 12 then
                                    state <= IDLE;  -- selesai
                                else
                                    char_index <= char_index + 1;
                                end if;
                            end if;
                        else
                            baud_counter <= baud_counter + 1;
                        end if;
                    end if;
            end case;
        end if;
    end process;

end Behavioral;
```

Pinout Nexys A



All ports (3)

Scalar ports (3)

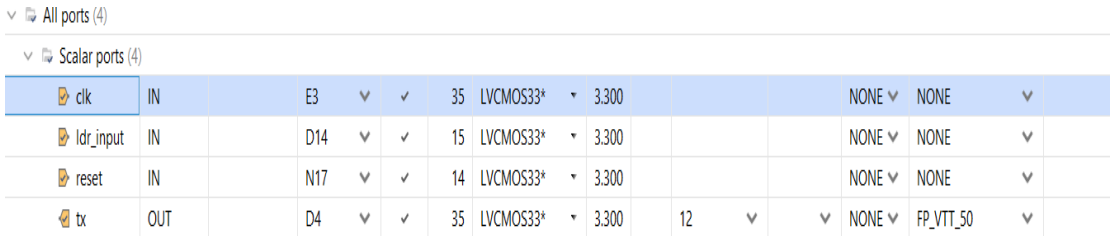
clk	IN		E3	✓	35	LVCMOS18	1.800				NONE	NONE
start	IN		N17	✓	14	LVCMOS18	1.800				NONE	NONE
tx	OUT		D4	✓	35	LVCMOS18	1.800		12		NONE	FP_VTT_50

2. Menggunakan sensor untuk menampilkan hasilnya pada PuTTY

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LDR_UART is
    Port (
        clk       : in  STD_LOGIC;
        reset     : in  STD_LOGIC;
        ldr_input : in  STD_LOGIC;
        tx        : out STD_LOGIC
    );
end LDR_UART;

architecture Behavioral of LDR_UART is
    constant clk_freq     : integer := 100_000_000; -- System clock frequency (100 MHz)
    constant baud_rate    : integer := 9600;        -- UART baud rate
    constant baud_div     : integer := clk_freq / baud_rate;
    constant delay_cycles : integer := clk_freq * 5; -- 5-second delay

    type state_type is (IDLE, DELAY, SEND);
    signal state : state_type := IDLE;

    signal tx_data : std_logic_vector(7 downto 0);
    signal tx_busy : std_logic := '0';
    signal bit_cnt : integer range 0 to 9 := 0;
    signal shift_reg : std_logic_vector(9 downto 0);
    signal baud_counter : integer range 0 to baud_div := 0;
    signal delay_counter : integer range 0 to delay_cycles := 0;

    type string_array is array (0 to 7) of std_logic_vector(7 downto 0);

    signal message_terang : string_array := (
        X"54", X"65", X"72", X"61", X"6E", X"67", X"0D", X"0A"  -- "Terang\r\n"
    );

    signal message_gelap : string_array := (
        X"47", X"65", X"6C", X"61", X"70", X"0D", X"0A", X"0A"  -- "Gelap\r\n"
    );


        signal char_index : integer range 0 to 7 := 0;
        signal current_message : string_array;

    begin
        process(clk, reset)
        begin
            if reset = '1' then
                state <= IDLE;
                tx_busy <= '0';
                char_index <= 0;
                tx <= '1';
                delay_counter <= 0;
            elsif rising_edge(clk) then
                case state is
                    when IDLE =>
                        tx <= '1';  -- Idle state of UART TX line
                        if ldr_input = '1' then
                            current_message <= message_gelap;
                        else
                            current_message <= message_terang;
                        end if;
                        char_index <= 0;
                        state <= SEND;
```

```
                when SEND =>
                    if tx_busy = '0' then
                        tx_data <= current_message(char_index);
                        shift_reg <= '1' & tx_data & '0'; -- Start bit, data, stop bit
                        bit_cnt <= 0;
                        tx_busy <= '1';
                    end if;

                    if tx_busy = '1' then
                        if baud_counter = baud_div then
                            baud_counter <= 0;
                            tx <= shift_reg(0);
                            shift_reg <= '1' & shift_reg(9 downto 1);
                            bit_cnt <= bit_cnt + 1;
                            if bit_cnt = 9 then
                                tx_busy <= '0';
                                if char_index = 7 then
                                    state <= DELAY;  -- Delay before next reading
                                    delay_counter <= 0;
                                else
                                    char_index <= char_index + 1;
                                end if;
                            end if;
                        else
                            baud_counter <= baud_counter + 1;
                        end if;
                    end if;

                when DELAY =>
                    if delay_counter < delay_cycles then
                        delay_counter <= delay_counter + 1;
                    else
                        state <= IDLE; -- Restart cycle after delay
                    end if;
            end case;
        end if;
    end process;
end Behavioral;
```

Pinout Nexys A7



All ports (4)

Scalar ports (4)

clk	IN		E3	✓	35	LVCMOS33*	3.300				NONE	NONE
ldr_input	IN		D14	✓	15	LVCMOS33*	3.300				NONE	NONE
reset	IN		N17	✓	14	LVCMOS33*	3.300				NONE	NONE
tx	OUT		D4	✓	35	LVCMOS33*	3.300		12		NONE	FP_VTT_50